

# **Oracle® Endeca Information Discovery Integrator**

Integrator ETL User's Guide

Version 3.2.0 • January 2016

## Copyright and disclaimer

Copyright © 2003, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. UNIX is a registered trademark of The Open Group.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Table of Contents

<b>Copyright and disclaimer</b> .....	<b>2</b>
<b>Preface</b> .....	<b>8</b>
About this guide .....	8
Who should use this guide .....	9
Conventions used in this guide .....	9
Contacting Oracle Customer Support .....	10
Recommended reading .....	10
<b>Chapter 1: Integrator ETL Overview</b> .....	<b>11</b>
Integrator ETL Designer .....	11
Integrator ETL Server .....	12
Configuring Integrator ETL .....	13
Setting a default time zone for incoming data .....	13
Verifying installed Web service versions .....	14
Securing graph parameters .....	14
Additional documentation .....	17
<b>Chapter 2: Building a Simple Project</b> .....	<b>19</b>
Starting Integrator ETL .....	19
Creating a project .....	19
Adding the sample data .....	20
Building your first graph .....	21
Adding a new component .....	21
Adding data to the component .....	22
Defining metadata for the geography data .....	22
Adding a Trash component .....	25
Connecting two components with an edge .....	26
Assigning metadata to the edge .....	27
Running the graph .....	28
Checking the output .....	28
Debugging the graph .....	29
Viewing the XML source for the graph .....	29
Sending data to the Endeca data domain .....	30
<b>Chapter 3: Working with Data Domains</b> .....	<b>34</b>
About working with data domains .....	34
Recommended data domain creation and configuration process .....	34
About data domain data .....	36
Supported data types .....	36
Supported languages .....	37
Default values for new attributes .....	38

Creating mdexType Custom properties . . . . .	39
Specifying multiple record delimiters . . . . .	43
Creating data domains . . . . .	44
Loading and maintaining data in a data domain . . . . .	45
Choosing a data loading strategy . . . . .	46
Choosing a loader . . . . .	47
Processing multi-value data . . . . .	49
Load graphs . . . . .	49
Source data format . . . . .	50
Configuring a Reformat component to generate a primary key . . . . .	52
Configuring the Bulk Add/Replace Records component for initial load . . . . .	52
Adding records and assignments . . . . .	53
Merge Records input . . . . .	54
Merge Records graph . . . . .	54
Configuring the Merge Records component . . . . .	55
Modifying records . . . . .	56
Modify Records input . . . . .	57
Modify Records graph . . . . .	58
Specifying the Record Set Specifier Attributes . . . . .	58
Backing up and restoring data domains . . . . .	59
Deleting data from the data domain . . . . .	59
Deleting records from the data domain . . . . .	59
Delete Records graph . . . . .	61
Configuring the Delete Records component . . . . .	61
<b>Chapter 4: Configuring Data Domains . . . . .</b>	<b>63</b>
About configuring data domains . . . . .	63
Global Configuration Record . . . . .	64
Sample GCR input file . . . . .	65
Creating the GCR graph . . . . .	65
GCR Web service code . . . . .	65
Configuring attributes . . . . .	66
Loading the standard attribute schema . . . . .	67
Loading the managed attribute schema . . . . .	69
Loading and modifying managed attribute values (MVals) . . . . .	72
Configuring a Merge Managed Values component to load or modify managed attribute values . . . . .	75
Configuring record search . . . . .	76
Configuring value search . . . . .	77
Configuring relevance ranking . . . . .	78
Configuring stop words . . . . .	78
Configuring the thesaurus . . . . .	79
Configuring precedence rules . . . . .	80
Precedence rule schema . . . . .	81
Format of the precedence rules input file . . . . .	82
Building a precedence rules graph . . . . .	83
XML-based configuration graphs . . . . .	84

Generating a single XML string	84
Creating an aggregation edge for configuration loading graphs	85
CSV-based configuration graphs	86
Exporting and importing data domain configurations	86
Building an export graph	87
Building an import graph	87
Configuring edges for export and import graphs	88
<b>Chapter 5: Managing View Definitions</b>	<b>91</b>
About view definitions	91
Exporting view definitions	91
Importing view definitions	93
<b>Chapter 6: Working with Outer Transaction Graphs</b>	<b>96</b>
About transactions	96
About the Transaction RunGraph component	97
When to use outer transactions	97
Setting up outer transactions	97
Creating an outer transaction graph using the Transaction RunGraph component	98
Format of the steps input file	98
Creating an outer transaction graph	99
Committing or rolling back an outer transaction	99
Performance impact of transactions	101
<b>Chapter 7: Loading Data from Special Repositories</b>	<b>102</b>
Loading data from an IAS Record Store	102
Configuring the Record Store Reader	103
Generating edge metadata with the Record Store Wizard	103
Loading Data from Oracle Business Intelligence Server	107
Using the Load Data from OBI Server wizard	107
Adding collections to a Load data from OBI project	119
Loading Oracle Business Intelligence Server data to Endeca Server	123
Configuring communication security for OBI graphs	123
Configuring OBI graphs for encrypted passwords	124
Configuring OBI Server graphs for Integrator ETL Server	124
<b>Chapter 8: Enhancing Text</b>	<b>126</b>
Choosing Text Enrichment or Text Tagging	126
Using Text Tagger components	126
Overwriting and appending target values	127
Text Tagger input and output metadata	127
Using the Text Tagger Whitelist component	127
Text Tagger Whitelist input tags-rules	128
Adding the Text Tagger Whitelist component to a graph	128
Search term lengths	129
Text Tagger Whitelist edges	129
Using the Text Tagger Regex component	130

Text Tagger Regex input patterns file .....	131
Adding Text Tagger Regex to a graph .....	131
Text Tagger Regex edges .....	132
Using Text Enrichment .....	132
Text Enrichment prerequisites .....	134
Text Enrichment properties file .....	135
Query topics definition file .....	144
Adding the Text Enrichment component to a graph .....	144
Text Enrichment component edges .....	145
Creating metadata for an example Text Enrichment edge .....	147
Processing text formatted in all caps .....	148
Normalizing themes .....	148
Adding foreign language processing to Text Enrichment .....	148
Processing multiple languages .....	149
Adding Social Media processing to Text Enrichment .....	149
Detecting text language .....	149
Language Detector edges .....	150
<b>Chapter 9: Component Reference .....</b>	<b>151</b>
Add KVPs component .....	152
Bulk Add/Replace Records component .....	156
Delete Records component .....	163
Export Config component .....	170
Import Config component .....	171
Language Detector component .....	173
Merge Managed Values component .....	175
Merge Records component .....	178
Modify Records component .....	183
Record Store Reader .....	189
Reset Data Domain component .....	192
Text Enrichment component .....	194
Text Tagger Regex component .....	196
Text Tagger Whitelist component .....	197
Transaction RunGraph component .....	200
Visual properties of components .....	205
Common configuration properties of components .....	206
Multi-assign delimiter .....	207
Batch size adjustments by components .....	208
Valid characters for ingest .....	208
<b>Appendix A: Implementing SSL .....</b>	<b>209</b>
Configuring SSL for Information Discovery components .....	210
Configuring SSL for the Web Service Client component .....	211
Configuring Integrator ETL Server to support SSL .....	211
<b>Appendix B: Troubleshooting Problems .....</b>	<b>213</b>
OutOfMemory errors .....	213

Transaction-related errors . . . . .	215
Connection errors . . . . .	215
Multi-assign delimiter error . . . . .	216
<b>Appendix C: Managing Studio Metadata . . . . .</b>	<b>217</b>
Managing metadata for physical attributes and attribute groups . . . . .	220
Creating Information Discovery metadata properties . . . . .	220
Assigning values to attribute metadata properties . . . . .	221
Assigning values to attribute group configurations . . . . .	222
Managing metadata for views . . . . .	222
Studio features and their associated metadata . . . . .	225
General attribute metadata . . . . .	225
General attribute group metadata . . . . .	226
Aggregation method metadata properties . . . . .	227
Display formats for attributes . . . . .	227
Display format metadata properties . . . . .	228
About the formatting specification . . . . .	228
Boolean display format settings . . . . .	228
Date/Time display format settings . . . . .	229
Duration display format settings . . . . .	230
Geocode display format settings . . . . .	231
Number display format settings . . . . .	231
Time display format settings . . . . .	233
Configuring Studio usage and display of attributes . . . . .	233
Localization in Studio . . . . .	235
Localizing display names and descriptions of attributes and predefined metrics . . . . .	236
Localizing attribute group display names . . . . .	237
Localizing view display names and descriptions . . . . .	238
Localizing attributes . . . . .	238
Date and time metadata properties . . . . .	242
<b>Appendix D: Managing Collections for Studio . . . . .</b>	<b>243</b>
Creating and updating collections . . . . .	243
Collection key and attribute names . . . . .	244
Prepending collection keys to attribute names in metadata . . . . .	244

## Preface

Oracle® Endeca Information Discovery Integrator is a powerful visual data integration environment that includes:

The Integrator Acquisition System (IAS) for gathering content from delimited files, file systems, JDBC databases, and Web sites.

Integrator ETL, an out-of-the-box ETL purpose-built for incorporating data from a wide array of sources, including Oracle BI Server.

In addition, Oracle Endeca Web Acquisition Toolkit is a Web-based graphical ETL tool, sold as an add-on module. Text Enrichment and Text Enrichment with Sentiment Analysis are also sold as add-on modules. Connectivity to data is also available through Oracle Data Integrator (ODI).

## About this guide

This guide describes how to use Oracle Endeca Information Discovery Integrator ETL to integrate data into an Endeca data domain.

Integrator ETL loads records, taxonomies, and configuration documents into the Endeca data domain.

The guide assumes that you are familiar with Endeca concepts and Endeca application development, as well as the interface of the Data Ingest Web Service.

Both data architects and system administrators should be familiar with the chapter "Building a Simple Project".

If you are a data architect, the following sections will be of particular interest to you:

- Working with Data Domains
- Working with Outer Transactions
- Loading Data from Special Repositories
- Enhancing Text

If you are a system administrator, the following sections will be of particular interest to you:

- Configuring Data Domains
- Managing View Definitions



## Who should use this guide

This guide is intended for data architects who are responsible for loading source data and configuration documents into an Endeca data domain, and for system administrators who are responsible for backing up and maintaining the configurations and loaded data.

## Conventions used in this guide

The following conventions are used in this document.

### Typographic conventions

The following table describes the typographic conventions used in this document.

#### *Typographic conventions*

Typeface	Meaning
<b>User Interface Elements</b>	This formatting is used for graphical user interface elements such as pages, dialog boxes, buttons, and fields.
Code Sample	This formatting is used for sample code phrases within a paragraph.
<Variable Name>	This formatting is used for variable values, such as <install path>.
File Path	This formatting is used for file names and paths.

### Symbol conventions

The following table describes symbol conventions used in this document.

#### *Symbol conventions*

Symbol	Description	Example	Meaning
>	The right angle bracket, or greater-than sign, indicates menu item selections in a graphic user interface.	File > New > Project	From the File menu, choose New, then from the New submenu, choose Project.

## Contacting Oracle Customer Support

Oracle Customer Support provides registered users with important information regarding Oracle software, implementation questions, product and solution help, as well as overall news and updates from Oracle.

You can contact Oracle Customer Support through Oracle's Support portal, My Oracle Support at <https://support.oracle.com>.

## Recommended reading

In addition to this document and associated Integrator ETL documentation, Oracle recommends that Integrator ETL users read the Oracle Endeca Server documentation.

In particular, Oracle recommends read and become familiar with the *Oracle Endeca Server Developer's Guide*. You may also find the information in the *Oracle Endeca Server Data Loading Guide* valuable as well.



## Chapter 1

---

# Integrator ETL Overview

Oracle Endeca Information Discovery Integrator ETL is a high-performance platform that lets you extract source records from a variety of source types, and load them into an Endeca data domain.

[Integrator ETL Designer](#)

[Integrator ETL Server](#)

[Configuring Integrator ETL](#)

[Additional documentation](#)

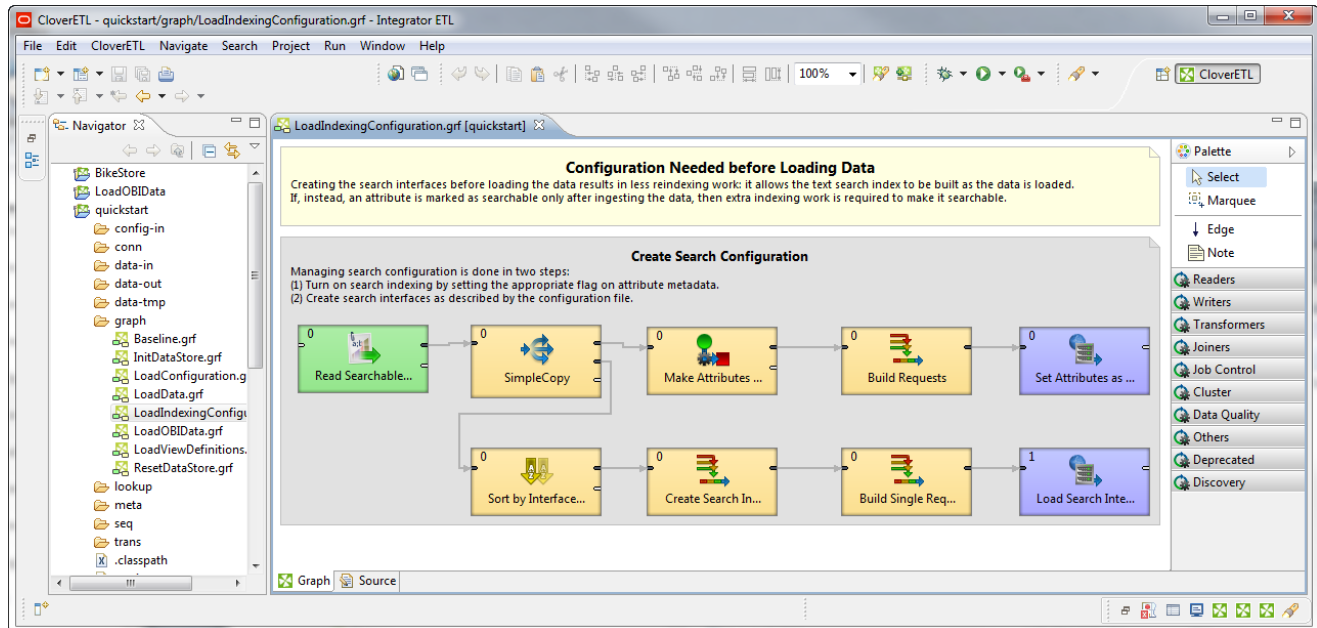
## Integrator ETL Designer

Integrator ETL Designer provides an easy-to-use interface you can use to create graphs for loading and updating your data quickly.

A graph is essentially a pipeline of components that processes the data. The simplest graph has one Reader component to read in the source data and one of the Information Discovery components to write (send) the data to the Endeca data domain. More complex graphs will use additional components, such as Transformer and Joiner components.

Integrator ETL, with its powerful graphical interface, provides an easy way to graphically lay out even complex graphs. You drag and drop the components from the Palette and then configure them by clicking the component icon.

The Integrator ETL perspective consists of four panes and the Palette tool, as shown in this example:



These panes are:

- The **Navigator** pane lists your projects, their folders (including the graph folders), and files.
- The **Outline** pane lists all the components of the selected graph.
- The **Tab** pane consists of a series of tabs (such as the Properties tab and the Console tab) that provide information about the components and the results of graph executions. The illustration shows the Log tab listing the output of a successful record loading operation.
- The **Graph Editor** pane allows you to create a graph and configure its components.
- The **Palette** allows you to select a component and drag it to the Graph Editor.

For more information on the Integrator ETL user interface, see the *Oracle Endeca Information Discovery Integrator ETL Designer Guide*.

## Integrator ETL Server

Information Discovery Integrator ETL Server provides a runtime environment for the graphs.

Integrator ETL Server is not required in order to load data into the Endeca data domains. In other words, you can run Integrator ETL Designer independently, and it does not require Integrator ETL Server to do its work.

You use Integrator ETL Server only if you are running graphs in an enterprise-wide environment. In this environment, different users and user groups can access and run the graphs. In addition, you can schedule the graphs to run at designated times, and monitor their execution progress.

Integrator ETL Server runs on an Apache Tomcat web application server or a WebLogic enterprise application server.

Because Integrator ETL Server is not a mandatory component for loading data into the Endeca data domains, it is not documented in this guide. For information on the setup and use of Integrator ETL Server, see the *Oracle Endeca Information Discovery Integrator ETL Server Guide*.

## Configuring Integrator ETL

This section provides information about configuration options for Integrator ETL.

[Setting a default time zone for incoming data](#)

[Verifying installed Web service versions](#)

[Securing graph parameters](#)

### Setting a default time zone for incoming data

You can specify the default time zone to use when the incoming data does not have time zone information on the dates.

By setting a default time zone, you can avoid the following scenario where you are reading date/time values from a database. The values in the database might indicate midnight of various dates. But when you look at the values in the Dgraph (for example, through Studio), you might see the same dates being shown with a 4am time stamp. This time difference may affect your application logic and your EQL statements.

The reason for this is that Integrator ETL parses time values using current time by default, unless the values contain an explicit time zone specifier. The Information Discovery component was correctly sending the time values to the Dgraph, which was storing them internally as UTC values. The Dgraph's query service only returns values in UTC, causing Studio to show the 4am values.

In this use case, the important factor is an end-to-end consistency in the time stamps. Therefore, the solution is to interpret these time stamps as UTC. There are two ways to do this.

#### Method 1: Modify the source data

The first method is to modify the source data to include a timestamp and change the format string in Integrator ETL to reflect that (e.g., from `dd.MM.yyyy HH:mm:ss` to `dd.MM.yyyy HH:mm:ss z`).

The advantage of this method is that you do not have to add components to your existing graph. However, this approach is not as appealing as the next two because the data comes from a database and the changes have to be made there.

#### Method 2: Use a Reformat component

The second method is, in Integrator ETL, to treat the timestamp as a string (i.e., change the metadata definition), and then write a CTL expression (such as in a **Reformat** component) to append an explicit time zone ("UTC") and parse it into a date value.

A sample of the CTL code would be:

```
$.OrderDate = str2date($.OrderDate + " UTC", "dd.MM.yyyy HH:mm:ss Z");
```

## Verifying installed Web service versions

Before making any calls to the Oracle Endeca Server with the **WebServiceClient** component of Integrator ETL, verify the version of the particular Web service you are going to use. The namespace of the Web service, which contains its version, must be included in the **WebServiceClient** component's configuration.

Web service namespaces include major and minor version numbers particular to each service (see the WSDLs for the exact formats).

The following example shows how a version number of 3.0 is represented in the namespace for the Configuration Web Service:

```
xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/3/0"
```

Note that the version of any Web service you are going to use may differ from the version shown in this example.

Namespaces affect requests as follows:

- Requests not using these namespaces are rejected.
- Requests using a different major version than was released with the Oracle Endeca Server are rejected.
- Requests using a minor version that is the same or lower than the version packaged with the server are accepted. For example, if a Transaction Web Service packaged with the Oracle Endeca Server has a version 1.1, and you use a version 1.0 that is listed in the namespace, then this request is accepted. All supported minor versions are listed in the WSDL. Any minor versions higher than supported are rejected.

For the **WebServiceClient** component, specify a string similar to the following in the **Operation name** field for the component:

```
{http://www.endeca.com/MDEX/config/services/config/3}Config#ConfigPort#DoConfigTransaction
```

Notice that this string includes a target namespace for the Web service that accurately reflects the major version of the Web service as /3 in this example.

## Securing graph parameters

You can secure graph parameters in Integrator ETL by setting up a Master Password.

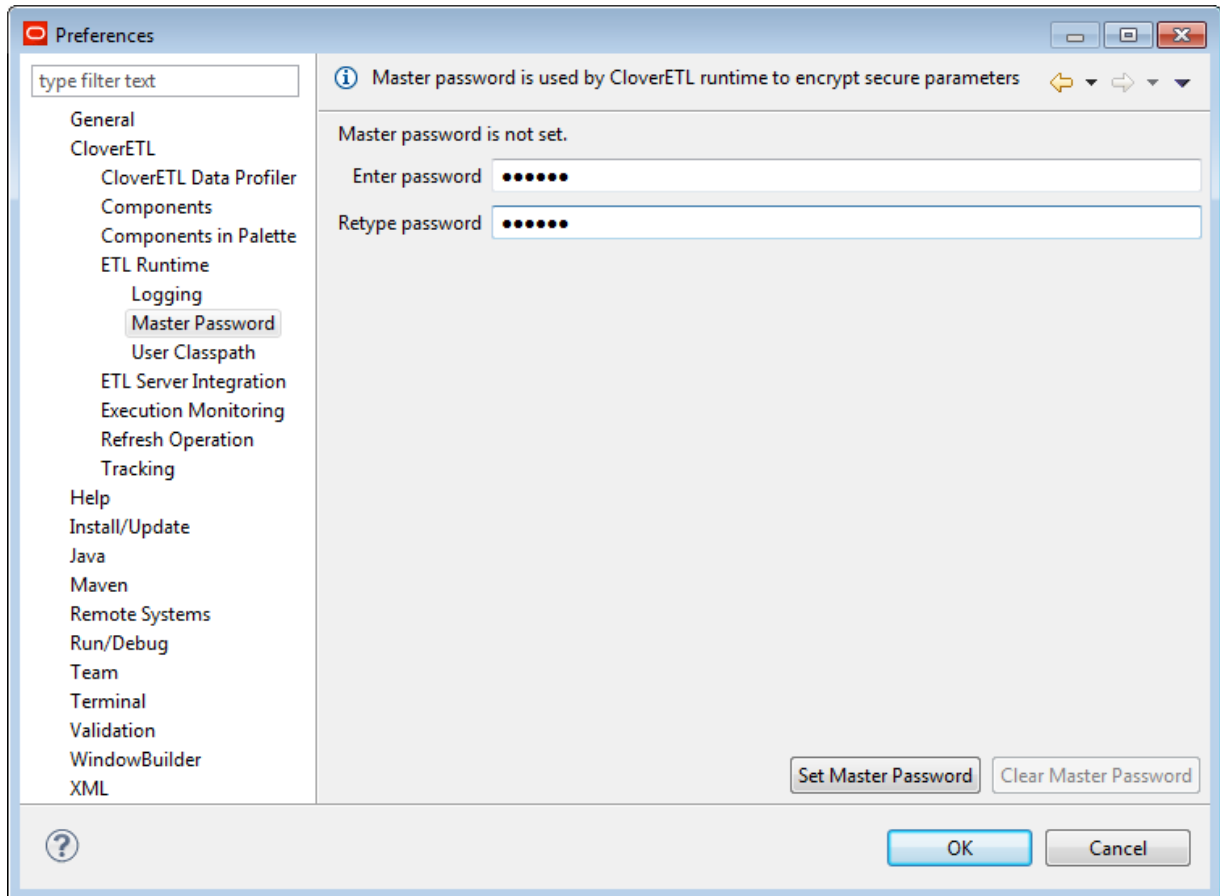
After setting up the Master Password, you can make a given graph parameter as secure. For example, your graph can read data from an Oracle database using an encrypted database connection password.

The sensitive information in secure parameters is persisted in encrypted form on file system. Decryption of secure parameters is automatically performed in graph runtime.

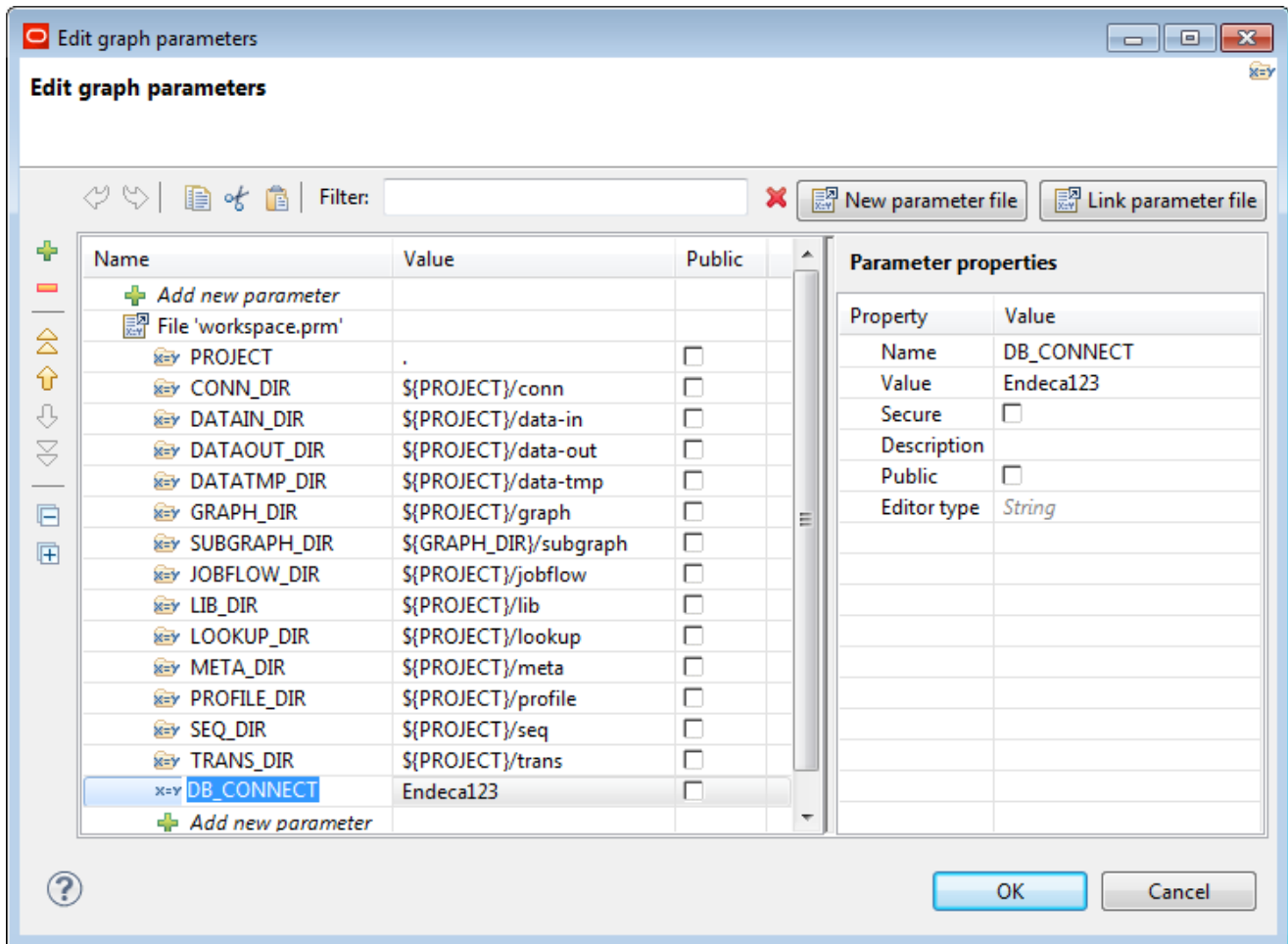
To configure the Secure Graph Parameters feature:

1. Open **Window>Preferences>CloverETL>ETL Runtime>Master Password**.

2. Enter your password, re-type it, and click **OK**.

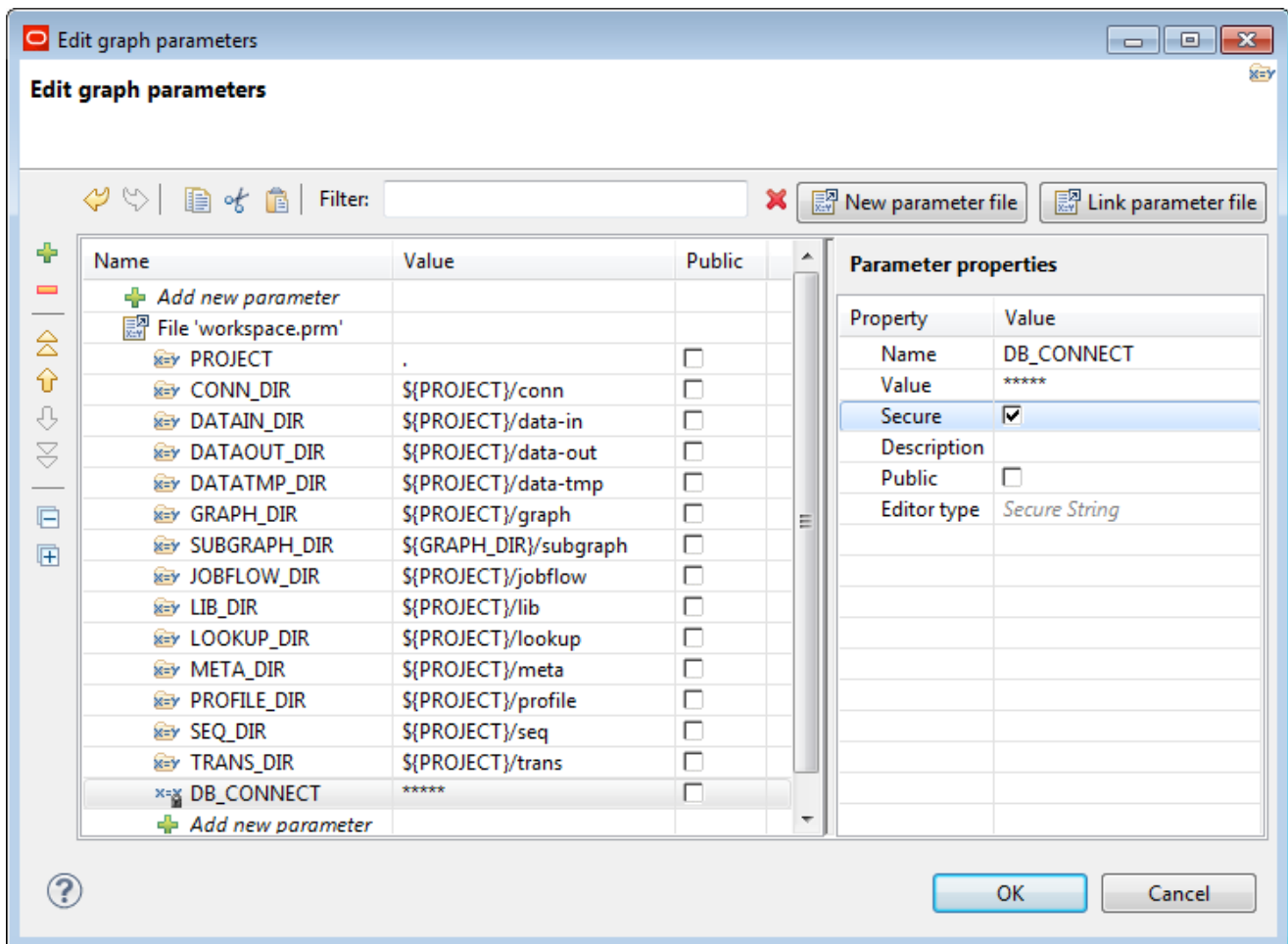


After the Master Password has been set, you can encrypt parameter properties. For example, assume you have just created a parameter named `DB_CONNECT` that contains a database password. The parameter displays its value as unencrypted in the Graph Parameters Editor:



If you then click the **Secure** checkbox in the **Parameter properties** pane, the parameter value will be encrypted:





For more information on the Secure Graph Parameters feature, see the *Integrator ETL Designer Guide*.

## Additional documentation

Additional Integrator ETL documentation is available online and as part of the Integrator documentation set.

### Integrator ETL documentation set

The following PDF documents are shipped as part of the Integrator documentation set:

- *Oracle Endeca Information Discovery Integrator ETL Designer Guide* – a comprehensive user's guide for Integrator ETL.
- *Oracle Endeca Information Discovery Integrator ETL Server Guide* – a comprehensive user's guide for Integrator ETL Server.

### Documentation online

You can access online documentation from within Integrator ETL by clicking **Help Contents** from the **Help** menu. This action returns the online Help system. Among the documents available in the online help is the

*CloverETL Designer User's Guide*, which is the online version of the *Oracle Endeca Information Discovery Integrator ETL Designer Guide*.



## Chapter 2

# Building a Simple Project

---

This section describes how to build a simple project.

[Starting Integrator ETL](#)

[Creating a project](#)

[Adding the sample data](#)

[Building your first graph](#)

[Running the graph](#)

[Checking the output](#)

[Debugging the graph](#)

[Viewing the XML source for the graph](#)

[Sending data to the Endeca data domain](#)

## Starting Integrator ETL

This topic describes how to start Integrator ETL.

To start Integrator ETL:

1. Go to the directory where you installed Integrator ETL and run the Integrator application.
2. Depending on how your Integrator ETL is configured, you may be asked to select or confirm your workspace.

The workspace is the directory where Integrator ETL creates and stores your projects.

3. The first time you launch Integrator ETL, a **Welcome** screen appears. Use this screen to navigate to launch Integrator ETL.



**Note:** You can return to the **Welcome** screen at any time by clicking **Help>Welcome**.

## Creating a project

This topic describes how to create a new Integrator ETL project.

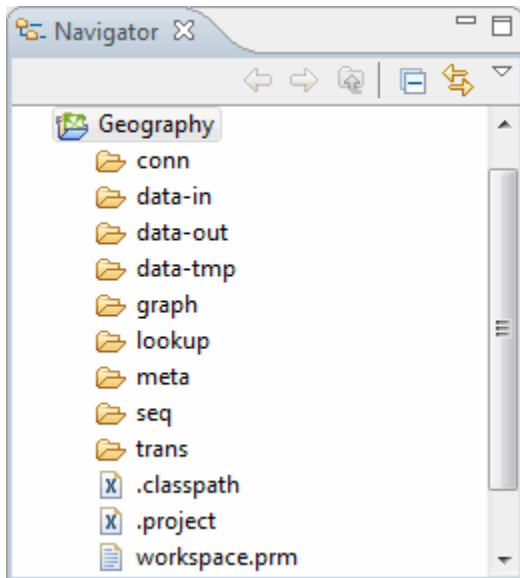
To create a new Integrator ETL project:

1. In Integrator ETL, select **File>New >Clover ETL Project**.

If you are running Integrator ETL for the first time, you may not see **Clover ETL Project** on the menu. In this case, select **Other>CloverETL>Clover ETL Project**.

2. In the **New CloverETL project** dialog box, type the project name (in this example we use **Geography**), set the directory location, and then click **Next**.
3. In the **Configure Clover ETL project subdirectories** dialog box, accept the default project directory locations, and then click **Finish**.
4. If you are asked about using the Clover ETL Perspective, click **Yes**.

A project called **Geography** appears in the **Navigator** pane. You can expand this to see the folders beneath it, all of which are currently empty.



## Adding the sample data

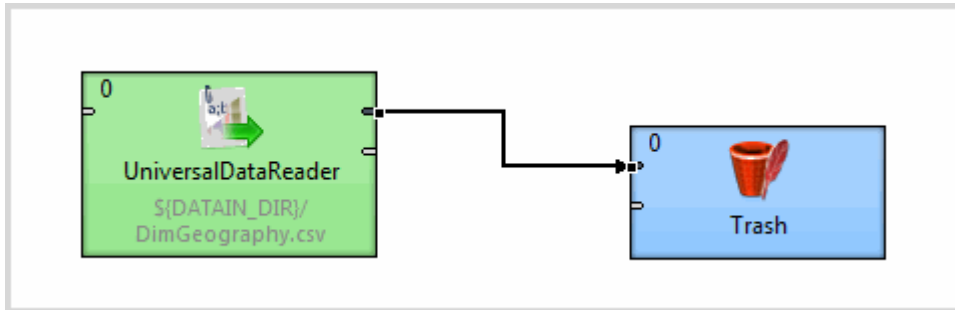
Sample data is available in the Getting Started application.

For details about obtaining and installing the Getting Started application, see the *Oracle Endeca Information Discovery Getting Started Guide*.

## Building your first graph

We are now ready to start building a transformation graph.

This simple graph contains two components connected by a single edge.



*Adding a new component*

*Adding data to the component*

*Defining metadata for the geography data*

*Adding a Trash component*

*Connecting two components with an edge*

*Assigning metadata to the edge*

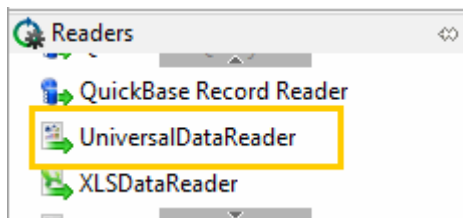
## Adding a new component

This topic illustrates how to add a component to read the geography data.

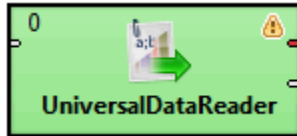
1. In the Navigator, right-click **Geography** and select **New>ETL Graph**.
2. Name the graph **LoadGeography**.
3. Click **Finish**.

An empty graph called **LoadGeography.grf** appears in the **Graph** editor.

4. In the **Palette**, click the section called **Readers** to open it.
5. Select **Universal Data Reader** and drag it onto the **Graph** editor.



The **LoadGeography.grf** now contains a single **UniversalDataReader** component.



## Adding data to the component

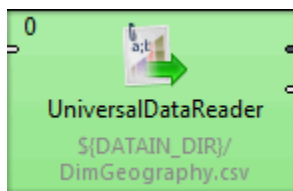
After creating the **UniversalDataReader** component, we need to associate data with it.

To add data to the component:

1. Double-click the **UniversalDataReader** component to open the **Edit Component** dialog box.
2. Click the **File URL** property.
3. Click the browse (...) button to the right of the **File URL** property.
4. In the **URL Dialog** dialog box, double-click the **data-in** folder to open it, and then select `DimGeography.csv`.
5. Click **OK** to return to the **Edit Component** dialog box.
6. Check the **Quoted strings** property to set it to true.

This step is necessary because the `DimGeography.csv` data contains quoted strings.

7. Locate the **Number of skipped records** property and set this to 1.  
Setting this value for the **Number of skipped records** ensures that header field names are not read in as proper data.
8. Click **OK** to return to the graph. The **UniversalDataReader** contains a reference to its data source.



9. Save the **LoadGeography** graph.

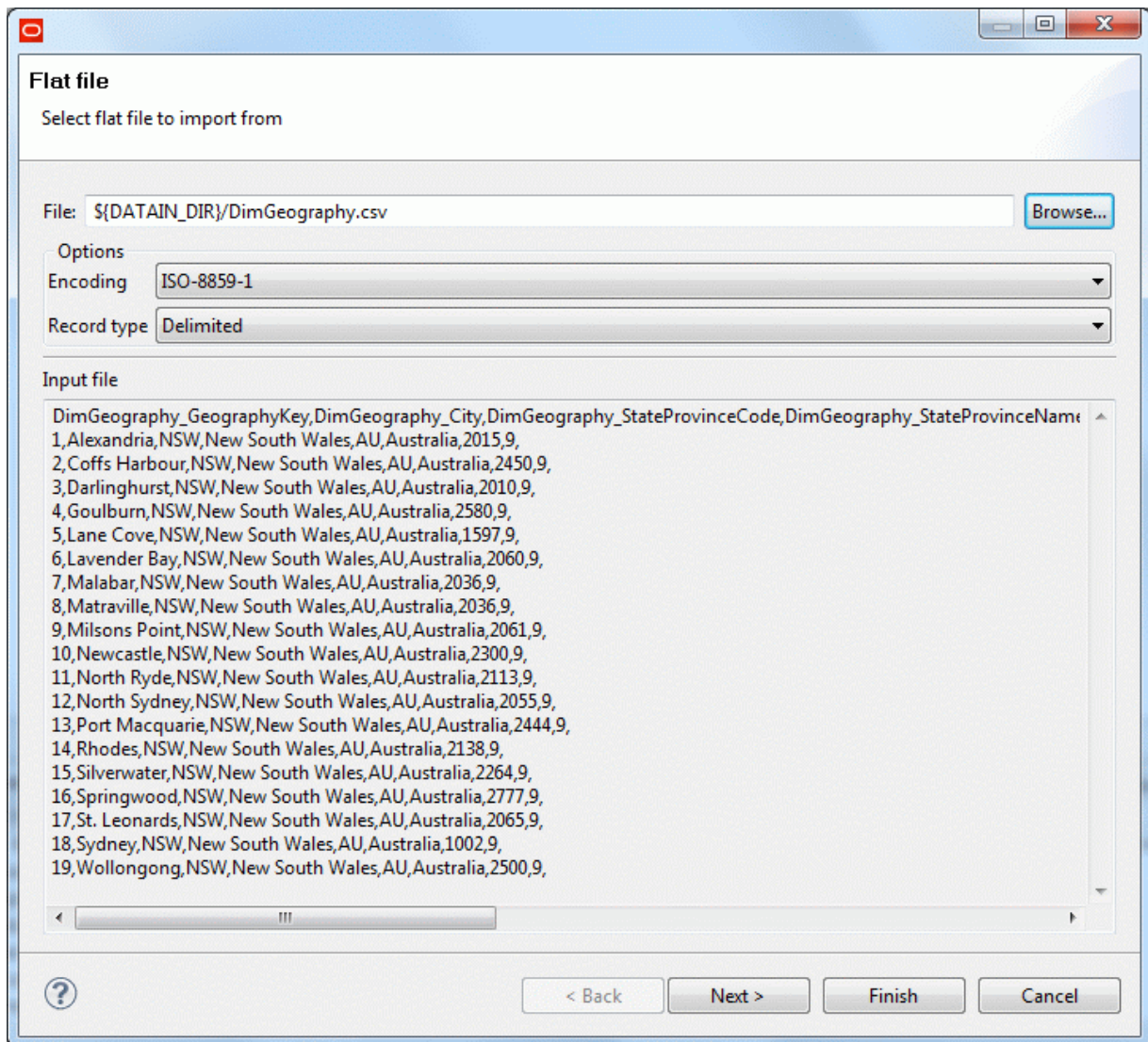
## Defining metadata for the geography data

In order to pass data from the **UniversalDataReader** to another component, you must define metadata that can be assigned to the edge that will join them together.

To define metadata:

1. In the **Outline**, right-click **Metadata** and select **New metadata > Extract from flat file**.

2. In the **File** text box, type or browse to the full path to your `DimGeography.csv` file, and then press **Enter**.



- Click **Next** to see the **Metadata** editor, where you can edit metadata properties.

The screenshot shows the 'Metadata editor' window with the following components:

- Field List Table:**

#	Name	Type	Delimi
1	Record: DimGeography_...	delimited	,
1	DimGeography_Geograp...	string	,
2	DimGeography_City	string	,
- Field Properties:**
  - ISO-8859-1
  - Comma ","
  - Quote char: [ ]
  - Extract names
  - Normalize names
  - Repa
- Data Preview Table:**


DimGeography_Geog...	DimGeograp...	DimGeography_StatePr...	DimGeography_StatePr..
1	Alexandria	NSW	New South Wales
2	Coffs Harbour	NSW	New South Wales
- Preview Text:**

```
DimGeography_GeographyKey,DimGeography_City,DimGeography_StateProvinceCode,Di
1,Alexandria,NSW,New South Wales,AU,Australia,2015,9,
2,Coffs Harbour,NSW,New South Wales,AU,Australia,2450,9,
3,Darlinghurst,NSW,New South Wales,AU,Australia,2010,9,|
```
- Buttons:** < Back, Next >, Finish, Cancel

- To give the metadata a useful name, rename the topmost record in the **Fields** list to **Geography**.

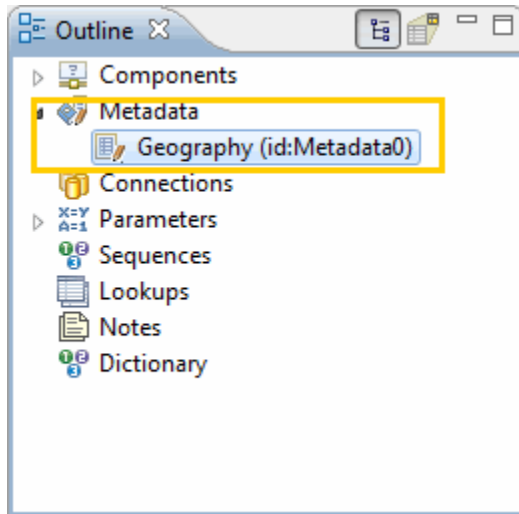
The screenshot shows the 'Fields' list table with the first record highlighted and renamed:

#	Name	Type	Delimiter
1	Record: <b>Geography</b>	delimited	,
1	DimGeography_Geograp...	integer	,
2	DimGeography_City	string	,
3	DimGeography_StatePro	string	,

 **Note:** Make sure you tab out of this field. Otherwise it will not be saved correctly.



5. Click **Finish**.



The **Geography** metadata item now appears in the **Metadata** collection. To edit the metadata, double-click it.

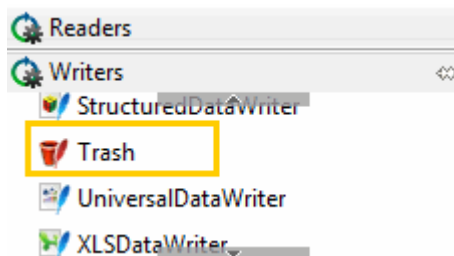
## Adding a Trash component

In this topic, you add a **Trash** component to your simple graph.

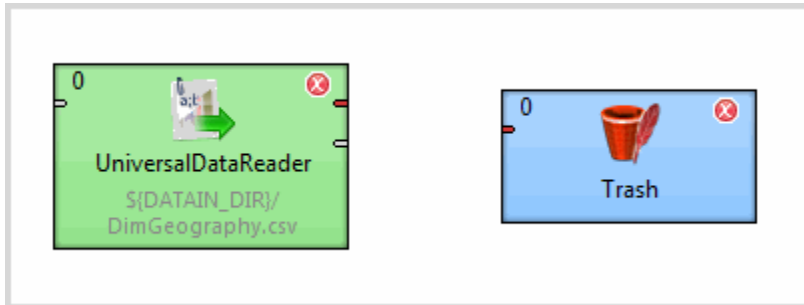
The **Trash** component tests the end points in a graph. Any data that arrives in the **Trash** component is discarded, which means that there is no need to create a file or database output. The **Trash** component also allows you to use some of the debugging capabilities of the Integrator ETL to monitor graph execution.

To add a **Trash** component:

1. In the **Palette**, click the section called **Writers** to open it.
2. Select **Trash** and drag it onto the **Graph** editor.



The **LoadGeography.grf** now contains two unconnected components.

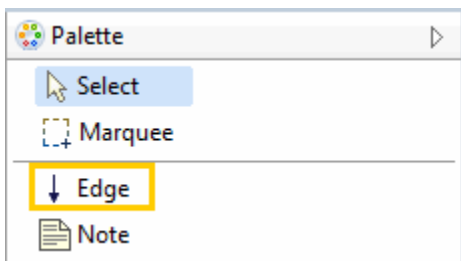


## Connecting two components with an edge

In this topic, you connect the **Trash** component to the **UniversalDataReader** component with an edge.

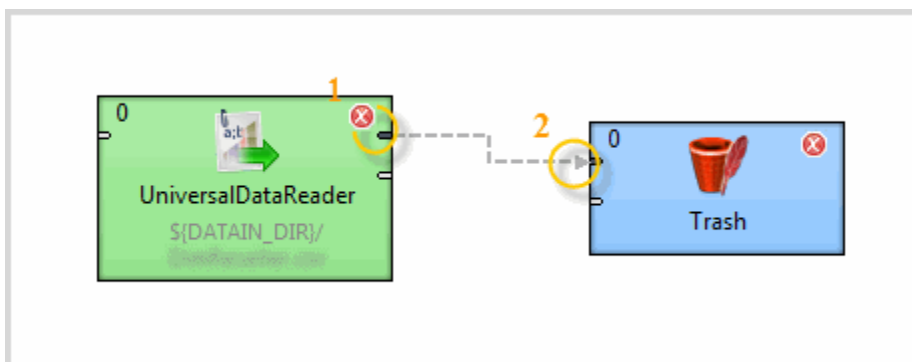
To connect two components with an edge:

1. In the **Palette**, select the **Edge** tool.



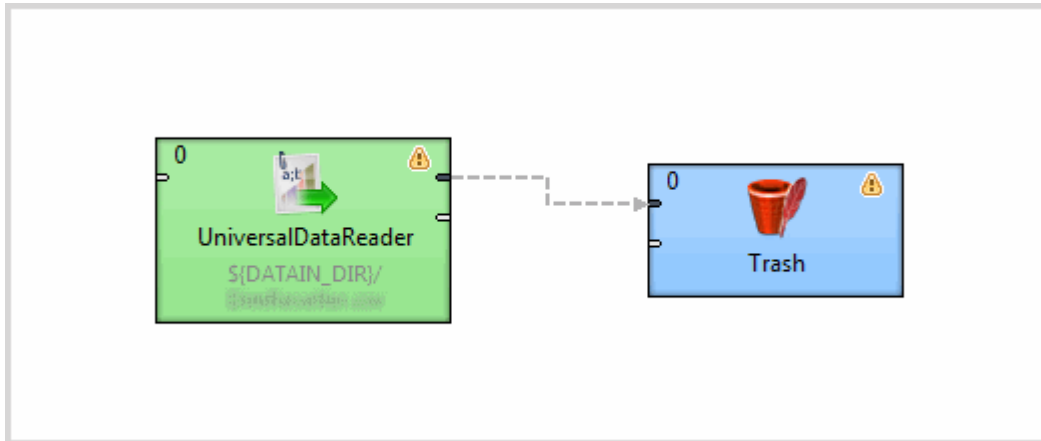
2. Click on the upper output port of the **UniversalDataReader** (number 1 in the image below) and drag across to the upper input port of the **Trash** component (number 2 in the image below).

You have to click on the target component to connect the edge.



3. Press **Esc** to change from Edge mode back to Select mode.
4. Save the graph file.

The **LoadGeography.grf** now contains two components connected by an edge.



## Assigning metadata to the edge

This topic illustrates how to apply metadata to an edge.

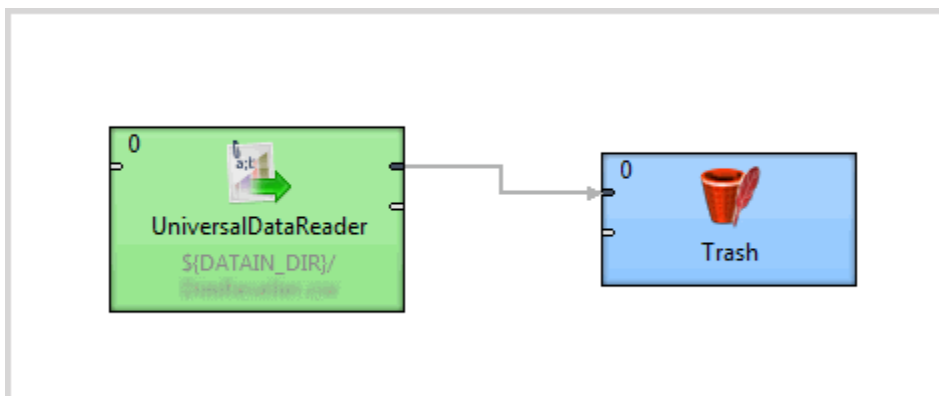
To assign metadata to an edge:

1. Right-click on the edge connecting the **UniversalDataReader** and **Trash** components.
2. Choose **Select Metadata > Geography**.

The edge becomes a solid, rather than a dashed, line, which indicates that metadata is associated with it.

3. Save your graph.

The **LoadGeography.grf** is now ready to be run.



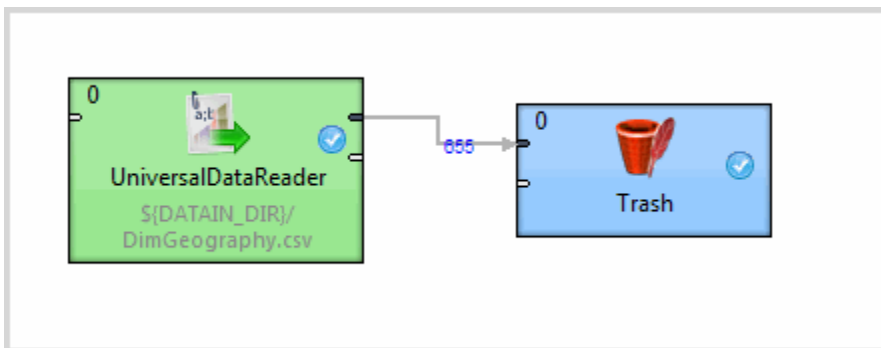
## Running the graph

After creating the graph and configuring the components, you can run the graph.

To run your Integrator ETL graph:

1. Run your graph in one of three ways:
  - Select **Run>Run As>CloverETL graph** from the main menu.
  - Right-click in the **Graph** editor, and then select **Run As>CloverETL graph**.
  - In the toolbar, click the run icon .

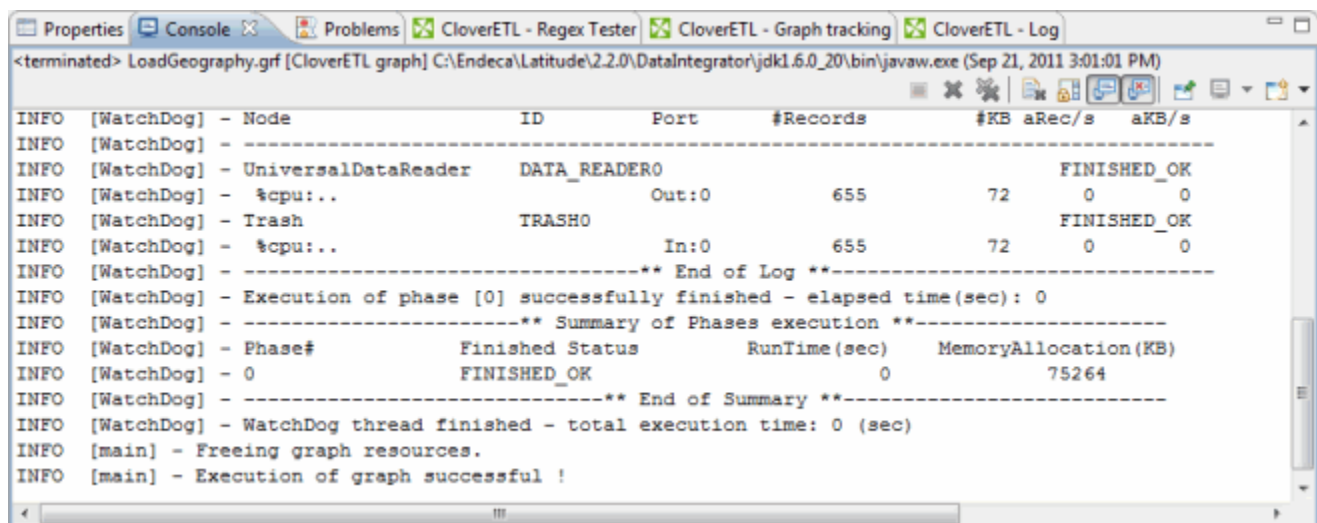
Upon successful execution, the components are flagged with a check mark, and the edge displays the number of records processed.



## Checking the output

Using the **Console** and the **Clover Log**, you can check the details of the graph execution.

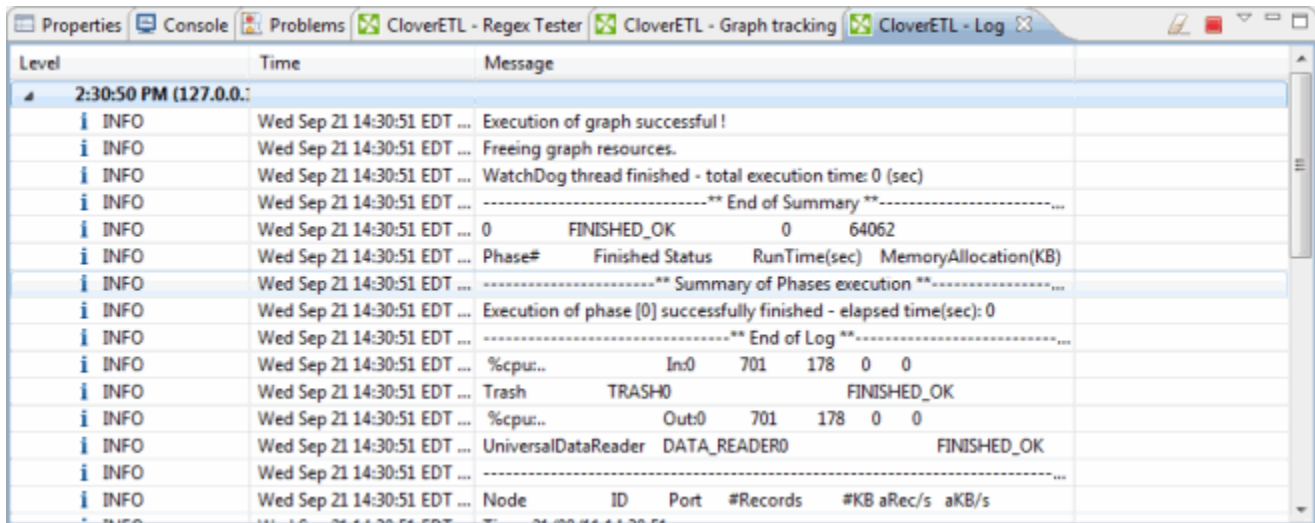
When a job runs, a **Console** window opens up at the bottom of the Eclipse window below the graph workspace. The **Console** logs the output for graph execution. The image below shows the **Console**:



```

<terminated> LoadGeography.grf [CloverETL graph] C:\Endeca\Latitude\2.2.0\DataIntegrator\jdk1.6.0_20\bin\javaw.exe (Sep 21, 2011 3:01:01 PM)
INFO [WatchDog] - Node ID Port #Records #KB aRec/s aKB/s
INFO [WatchDog] - -----
INFO [WatchDog] - UniversalDataReader DATA_READERO
INFO [WatchDog] - %cpu:.. Out:0 655 72 0 0
INFO [WatchDog] - Trash TRASHO
INFO [WatchDog] - %cpu:.. In:0 655 72 0 0
INFO [WatchDog] - -----** End of Log **-----
INFO [WatchDog] - Execution of phase [0] successfully finished - elapsed time(sec): 0
INFO [WatchDog] - -----** Summary of Phases execution **-----
INFO [WatchDog] - Phase# Finished Status Runtime(sec) MemoryAllocation(KB)
INFO [WatchDog] - 0 FINISHED_OK 0 75264
INFO [WatchDog] - -----** End of Summary **-----
INFO [WatchDog] - WatchDog thread finished - total execution time: 0 (sec)
INFO [main] - Freeing graph resources.
INFO [main] - Execution of graph successful !
  
```

Alternately, click the **CloverETL - Log** tab to view more concise output. The image below shows the **CloverETL - Log**:



Level	Time	Message
2:30:50 PM (127.0.0.1)		
INFO	Wed Sep 21 14:30:51 EDT ...	Execution of graph successful !
INFO	Wed Sep 21 14:30:51 EDT ...	Freeing graph resources.
INFO	Wed Sep 21 14:30:51 EDT ...	WatchDog thread finished - total execution time: 0 (sec)
INFO	Wed Sep 21 14:30:51 EDT ...	-----** End of Summary **-----
INFO	Wed Sep 21 14:30:51 EDT ...	0 FINISHED_OK 0 64062
INFO	Wed Sep 21 14:30:51 EDT ...	Phase# Finished Status RunTime(sec) MemoryAllocation(KB)
INFO	Wed Sep 21 14:30:51 EDT ...	-----** Summary of Phases execution **-----
INFO	Wed Sep 21 14:30:51 EDT ...	Execution of phase [0] successfully finished - elapsed time(sec): 0
INFO	Wed Sep 21 14:30:51 EDT ...	-----** End of Log **-----
INFO	Wed Sep 21 14:30:51 EDT ...	%cpu:.. In:0 701 178 0 0
INFO	Wed Sep 21 14:30:51 EDT ...	Trash TRASH0 FINISHED_OK
INFO	Wed Sep 21 14:30:51 EDT ...	%cpu:.. Out:0 701 178 0 0
INFO	Wed Sep 21 14:30:51 EDT ...	UniversalDataReader DATA_READERO FINISHED_OK
INFO	Wed Sep 21 14:30:51 EDT ...	-----
INFO	Wed Sep 21 14:30:51 EDT ...	Node ID Port #Records #KB aRec/s aKB/s



**Note:** If you cannot see both of these tabs, go to the **Window** menu option, and then select **Reset Perspective**.

## Debugging the graph

Debugging is a vital (and easy-to-use) feature that lets you see exactly what data was passed along any edge in a graph.



**Note:** When debugging an Integrator ETL graph, keep in mind that all components in the same phase run in parallel and are multi-threaded. Therefore, make sure you start with components that are flagged as errors (🔴) and not with warnings (🟡), even if the warnings appear to occur logically before the errors.

To debug a graph:

1. Right-click the edge and then select **Enable Debug**.

The debug icon  appears on the edge.

2. Re-run the graph so that Integrator ETL can generate the debug data.
3. Right-click the edge and then select **View data**.

The **View data** window shows all of the data fields correctly parsed and loaded.


## Viewing the XML source for the graph

When you create a graph, it is saved as XML. You can view and edit this XML source.

To see the XML source for the graph:

1. At the bottom of the **Graph** editor, click the **Source** tab.

2. In the XML version of **LoadGeography.grf** (shown below), scroll to view the data. Any changes you make are automatically applied to the graphical version.



```

<Metadata id="Metadata0" previewAttachment="{DATAIN_DIR}/DimGeography.csv" previewAttachmentCharset="
<Record fieldDelimiter="," name="Geography" previewAttachment="{DATAIN_DIR}/DimGeography.csv" preview
<Field name="DimGeography_GeographyKey" type="integer"/>
<Field name="DimGeography_City" type="string"/>
<Field name="DimGeography_StateProvinceCode" type="string"/>
<Field name="DimGeography_StateProvinceName" type="string"/>
<Field name="DimGeography_CountryRegionCode" type="string"/>
<Field name="DimGeography_CountryRegionName" type="string"/>
<Field name="DimGeography_PostalCode" type="string"/>
<Field name="DimGeography_SalesTerritoryKey" type="integer"/>
</Record>
</Metadata>
<Property fileURL="workspace.prm" id="GraphParameter0"/>
<Dictionary/>
</Global>
<Phase number="0">
<Node enabled="enabled" fileURL="{DATAIN_DIR}/DimGeography.csv" guiHeight="93" guiName="UniversalData
<Node enabled="enabled" guiHeight="67" guiName="Trash" guiWidth="128" guiX="311" guiY="152" id="TRASH0
<Edge fromNode="DATA_READER0:0" guiBendpoints="" guiRouter="Manhattan" id="Edge1" inPort="Port 0 (in)"
</Phase>
</Graph>

```

## Sending data to the Endeca data domain

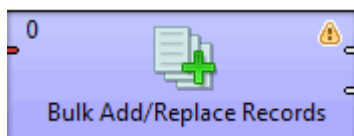
In this topic, you will replace the **Trash** component with a component that sends records to the Endeca data domain.

The procedure below assumes that the Endeca Server is running (using the default 7001 port) and that you have created a running Endeca data domain named **geography**. Creating an Endeca data domain is documented in the *Oracle Endeca Server Administrator's Guide*. The data will be loaded into a collection whose key is **base**.

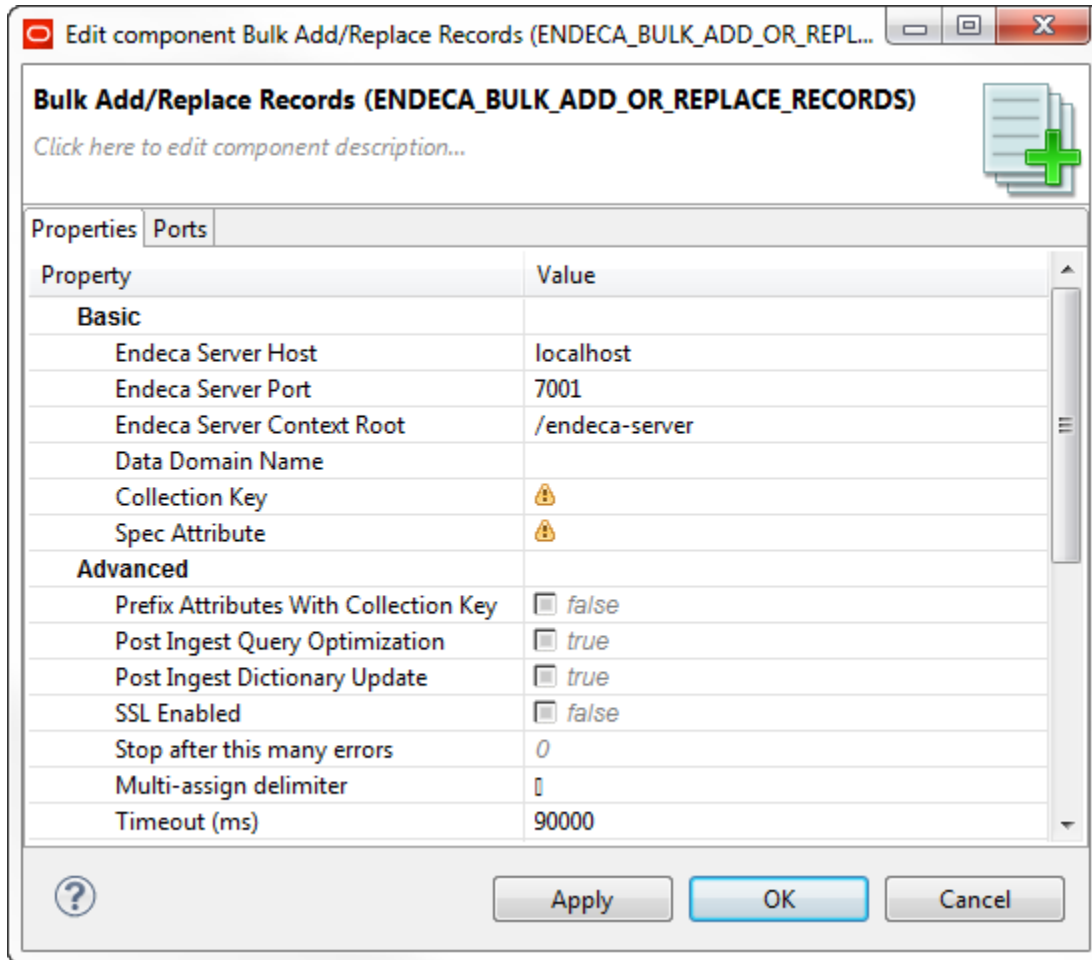
For details about the Endeca data domain load process, see [Recommended data domain creation and configuration process on page 34](#).

To send data to an Endeca data domain:

1. In the **Navigator**, select the **graph** folder in the **Geography** project.
2. In the **Palette**, click the section called **Discovery** to open it.
3. Select **Bulk Add/Replace Records** and drag it onto the **Graph** editor.



4. Double-click the **Bulk Add/Replace Records** component to open the **Edit Component** dialog box.



5. Set the following mandatory properties in the **Basic** section, and then click **OK**:
- Endeca Server Host:** The name or IP address of the machine. The default value (`localhost`) can be used as the name.
  - Endeca Server Port:** 7001 (the value of the default Endeca Server port)
  - The **Endeca Server Context Root** defaults to `/endeca-server`. Do not change the value of this property.
  - Data Domain Name:** The name of the Endeca data domain. In this example, `geography` is the name.
  - Collection key:** The key of the collection to which you want to load the data. In this example `base` is the Collection key.



**Note:** In Oracle Endeca Information Discovery Studio, collections are called "data sets".

- (f) **Spec Attribute:** The name of the primary key. In this example, DimGeography\_GeographyKey is the name.

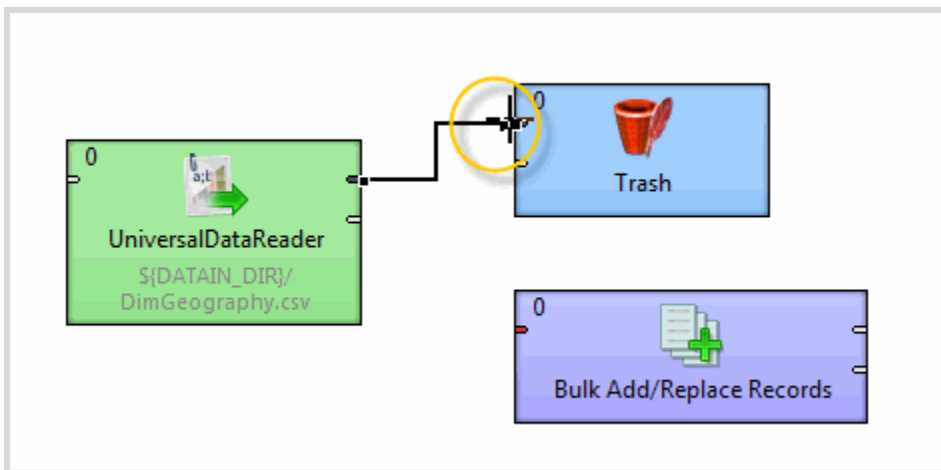


**Note:** Even though we are going to prepend the collection key to all attribute names, we should not prepend the collection key now. When we check the **Prefix Attributes With Collection Key** box, the collection key will be prepended automatically.

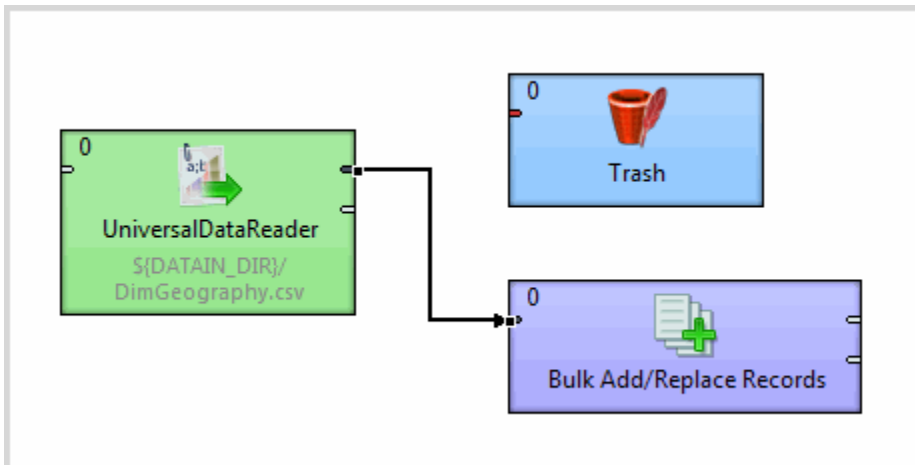
- (g) Check the **Prefix Attributes With Collection Key** box.

Studio requires that attribute names include the collection key. Check this box to prepend the collection key to the attribute names when loading data.

- 6. In the **Graph** editor, position the cursor over the input port of the **Trash** component so that the hand cursor becomes a +.



- 7. Drag the edge endpoint from the **Trash** component to the **Bulk Add/Replace Records** component that you just configured.



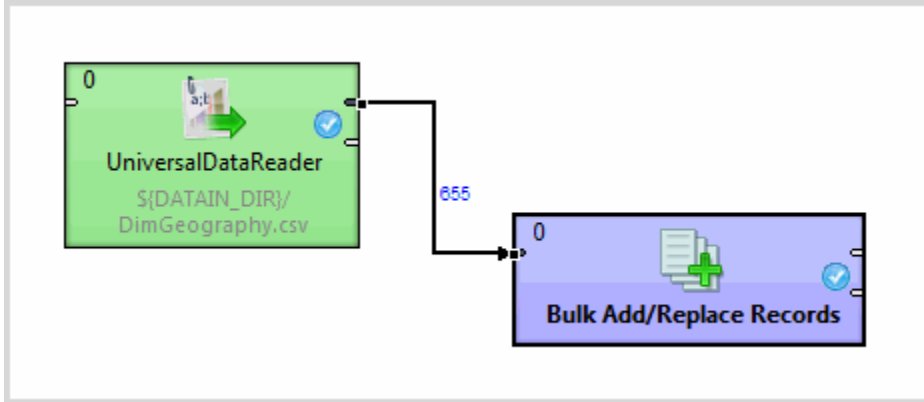
When you move the edge, its associated metadata is also moved.

- 8. Delete the **Trash** component.
- 9. Save the graph.

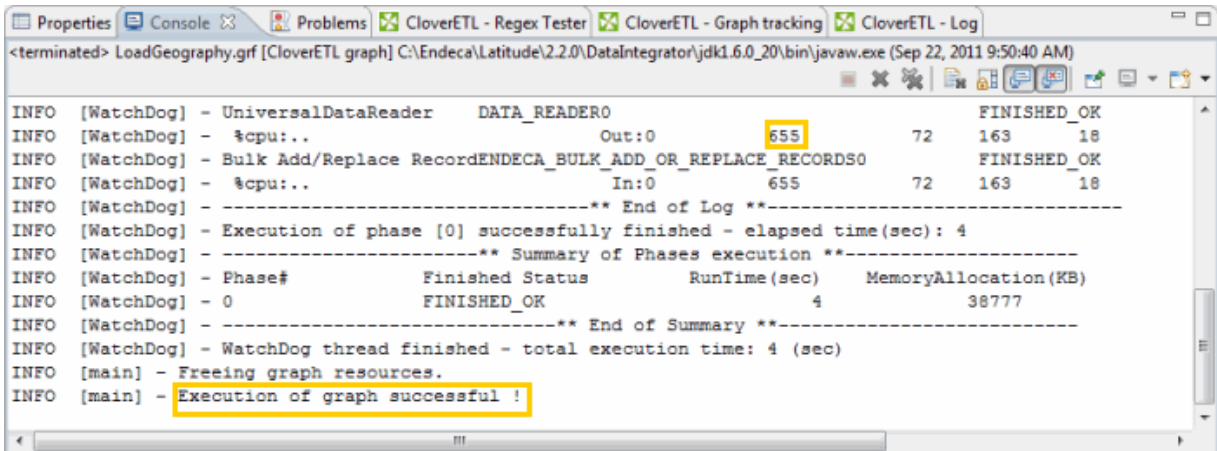


10. Run the graph.

Upon successful execution, the components are flagged with a check mark, and the edge displays the number of records processed.



In addition, you will see the following success message in the **Console**:





## Chapter 3

# Working with Data Domains

---

This section documents the creation, population, and maintenance of data domains.

[About working with data domains](#)

[Recommended data domain creation and configuration process](#)

[About data domain data](#)

[Creating data domains](#)

[Loading and maintaining data in a data domain](#)

[Load graphs](#)

[Adding records and assignments](#)

[Modifying records](#)

[Backing up and restoring data domains](#)

[Deleting data from the data domain](#)

## About working with data domains

This topic describes general information about working with data domains.

The configuration and loading of a data domain may be an iterative process. A data domain is created with a default set of attributes, but as you begin testing your applications, you may discover that you need a different set of standard and managed attributes, as well as other configurations, to generate the results you want in your applications.

Standard attributes, once assigned to records, cannot be modified. If you want to modify standard attributes, you must delete the data in the data domain, including the data domain configuration, modify and reload the data domain configuration, then reload the data. You may need to reconfigure and reload data multiple times before you develop a final set of configurations that meets your needs and goals.

## Recommended data domain creation and configuration process

This topic outlines the recommended process for creating, configuring, and loading a data domain.

Use the following process for creating, configuring, and populating a data domain. Note that all steps in the following process are options. So, for example, if the default global configuration meets your needs, you do

not need to update the Global Configuration Record. If you do not want to define any attribute hierarchies, you do not need to load the managed attribute schema and managed attribute values.

1. Create the data domain.
2. Update the Global Configuration Record (GCR).

The Global Configuration Record sets the global configuration settings for the Endeca data domain. See [Global Configuration Record on page 64](#).

3. Update the attribute schema configuration. See [Configuring attributes on page 66](#).

The attribute schema consists of:

1. The standard attribute schema

A standard attribute is the basic unit of information for a record, and consists of a key-value pair. When you load data into the Endeca Server, standard attributes are created with default data types. You may want to update the standard attribute schema in the following circumstances:

- You want to assign a data type that is different from the default data type for the attribute. For example, by default, part numbers consisting of numeric data would be assigned a numeric data type, such as integer or long. Part numbers are usually treated as string data, however, so you would want to define the part number as a string data type before loading the data.
- You want to define hierarchy of attribute values as managed attributes. These attributes must first be loaded as standard attributes.

Standard attributes are defined by Property Description Records, or PDRs.

2. The managed attribute schema

Managed attributes define a hierarchy of attribute values. For example, a location attribute might define a hierarchy of country, state or province, and city or town. Managed attributes are defined by both Property Description Records (PDRs), which define the attributes, and Dimension Description Records (DDR), which define the relation and hierarchy between the values of the attribute.

3. Managed attribute values, or mvals.

4. Configuration documents, loaded in this order:

1. `reirank_strategies`

This document configures the relevance ranking strategies for your Information Discovery application. You must load this document before you load the `recsearch_config` and `dimsearch_config` documents. For more information, see [Configuring relevance ranking on page 78](#).

2. `recsearch_config`

This document configures the record search, including search interfaces that control record search behavior for groups of attributes. For more information, see [Configuring record search on page 76](#).

3. `dimsearch_config`

This document sets the configuration for value search. For more information, see [Configuring value search on page 77](#).

4. `stop_words`

This document sets the stop words for queries. Stop words are words that are eliminated from a query before it is processed by the Dgraph. For more information, see [Configuring stop words on page 78](#).

5. `thesaurus`

This document configures the thesaurus for your application, which allows the system to return matches for related concepts to the words and phrases included in user queries. For more information, see [Configuring the thesaurus on page 79](#).

5. Source data you want to load into the Endeca Server.

Precedence rules can be loaded at any point in the sequence. Precedence rules suppress refinements of Endeca attributes until a condition is met. Suppressing some attributes makes navigation easier and prevents display of an excessive amount of information that can overwhelm the user. For more information, see [Configuring precedence rules on page 80](#).

## About data domain data

This section describes the data loaded into the data domain.

[Supported data types](#)

[Supported languages](#)

[Default values for new attributes](#)

[Creating mdexType Custom properties](#)

[Specifying multiple record delimiters](#)

## Supported data types

This topic lists the Integrator ETL native data types and specifies which of them are supported in the Endeca data domain's Dgraph process.

The table also shows how the Integrator ETL supported data types are mapped to the Dgraph data types during an ingest operation. You will see the data types when you create the metadata definition for a component's edge.

Integrator ETL Data Types in Metadata	Maps to Dgraph Data Type
boolean	mdex:boolean
byte	Not supported
cbyte	Not supported
date	mdex:dateTime
decimal	mdex:double
integer	mdex:int
long	mdex:long
number	mdex:double

Integrator ETL Data Types in Metadata	Maps to Dgraph Data Type
string	mdex:string
string with an mdexType Custom property set to mdex:duration	mdex:duration
string with an mdexType Custom property set to mdex:geocode	mdex:geocode

As the table notes, you can create an `mdexType` Custom property type for the input property's metadata, and the Dgraph will use that type when creating the standard attribute's PDR. For details, see [Creating mdexType Custom properties on page 39](#).

## Supported languages

Endeca data stores support upload of data in a number of languages.

Languages must be specified using a valid language code. Supported languages and language identifiers include:

- Catalan: `ca`
- Chinese (simplified): `zh_CN`
- Chinese (traditional): `zh_TW`
- Czech: `cs`
- Dutch: `nl`
- English: `en`
- French: `fr`
- German: `de`
- Greek: `el`
- Hebrew: `he`
- Hungarian: `hu`
- Italian: `it`
- Japanese: `ja`
- Korean: `ko`
- Polish: `po`
- Portuguese: `pt`
- Romanian: `ro`
- Russian: `ru`
- Spanish: `es`
- Swedish: `sv`

- Thai: `th`
- Turkish: `tr`
- no language specified: `unknown`



**Note:** Language IDs are not case sensitive.

For complete details about internationalization and language support in Endeca server, see "Internationalized Data" in the *Oracle Endeca Server Developer's Guide*.

## Default values for new attributes

New standard and managed attributes created during an ingest are given a set of default values.

During any data ingest operation, if a non-existent Endeca standard attribute is specified for a record, the specified attribute is automatically created by the Dgraph. Likewise, non-existent Endeca managed attributes specified for a record are also automatically created. Note that you cannot disable this automatic creation of these attributes.

### Standard attribute default values

The PDR for a standard attribute that is automatically created will use the system default settings, which (unless they have been changed by the data developer) are:

PDR property	Default setting
<code>mdex-property_Key</code>	Set to the standard attribute name specified in the request.
<code>mdex-property_Type</code>	Set to the standard attribute type specified in the request. If no type was specified, defaults to the <code>mdex:string</code> type.
<code>mdex-property_IsPropertyValueSearchable</code>	<code>true</code> (the standard attribute will be enabled for value search)
<code>mdex-property_IsSingleAssign</code>	<code>true</code> (a record may only have one value assignment for the standard attribute)
<code>mdex-property_IsTextSearchable</code>	<code>false</code> (the standard attribute will be disabled for record search)
<code>mdex-property_IsUnique</code>	<code>false</code> (more than one record may have the same value of this standard attribute)

PDR property	Default setting
<code>mdex-property_Language</code>	unknown (the language of the attribute is not known or has not been specified; for more information about internationalization and language support, see "Internationalized Data" in the <i>Oracle Endeca Server Developer's Guide</i> )
<code>mdex-property_TextSearchAllowsWildcards</code>	<code>false</code> (wildcard search is disabled for this standard attribute)
<code>system-navigation_Select</code>	<code>single</code> (allows selecting only one refinement from this standard attribute)
<code>system-navigation_ShowRecordCounts</code>	<code>true</code> (record counts will be shown for a refinement)
<code>system-navigation_Sorting</code>	<code>record-count</code> (refinements are sorted in descending order, by the number of records available for each refinement)

## Managed attribute default values

A managed attribute that is automatically created will have both a PDR and a DDR created by the Dgraph. The default values for the PDR are the same as listed in the table above, except that `mdex-property_IsPropertyValueSearchable` will be `false` (i.e., the managed attribute will be disabled for value search).

The DDR will use the system default settings, which (unless they have been changed by the data developer) are:

DDR property	Default setting
<code>mdex-dimension_Key</code>	Set to the managed attribute name specified in the request.
<code>mdex-dimension_EnableRefinements</code>	<code>true</code> (refinements will be displayed)
<code>mdex-dimension_IsDimensionSearchHierarchical</code>	<code>false</code> (hierarchical search is disabled during value searches)
<code>mdex-dimension_IsRecordSearchHierarchical</code>	<code>false</code> (hierarchical search is disabled during record searches)

## Creating `mdexType` Custom properties

Integrator ETL allows you to create an **`mdexType`** Custom property that you can use to explicitly specify the MDEX type to which a particular Endeca standard attribute should map.

The Custom property feature can be used to specify MDEX types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`) that are not natively supported in Integrator ETL. In this case, the ETL developer has to send

a string through Integrator ETL, making sure that the string value is formatted in the way that the Dgraph expects. The new **mdexType** Custom property, in other words, overrides the Integrator ETL native property type when the records are sent to the Dgraph.

This functionality is particularly useful for non-String multi-assign properties, because Integrator ETL natively has to treat the property as a string, because it has to include a delimiter. Thus, you can include delimiters in the multi-assign property (as though it were a String) but send the property to the Dgraph with `mdex:int` (for example) as the MDEX property type.



**Important:** Although the property will be designated as Integrator ETL type String, you must make sure that the string value is formatted according to the rules of the MDEX property type to which it will be mapped. For example, if it will be created as an `mdex:duration` attribute in the Dgraph, then the String value must use the `mdex:duration` format.

You add Custom properties by invoking the Custom property editor from the Fields pane in the Metadata Editor:



The Name field must be **mdexType**, and the Value field must be one of the MDEX property types (such as `mdex:duration`). The Name and Value are used by the Information Discovery component to specify (to the Dgraph) what MDEX property type should be used for when creating the standard attribute.

The source input file used as an example is a simple one:

```
ProductKey|ProductName|Duration|Location
95000|HL Mountain Rim|P429DT2M3.25S|42.365615 -71.075647
```

It creates only one record with four standard attributes:

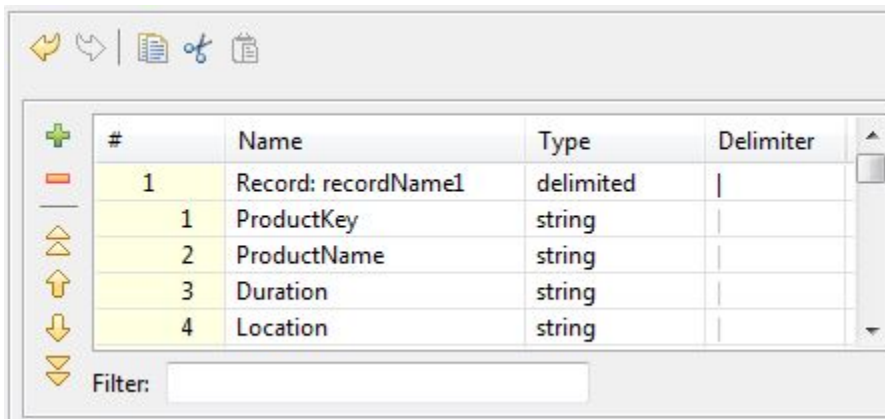
- The ProductKey attribute is the primary key and is an Integer. Its value is 9500.
- The ProductName attribute is a String type with a value of "HL Mountain Rim".
- The Duration attribute will be a String property in the Designer metadata, but will use a Custom property of `mdex:duration` in order to create a Duration standard attribute. Its value is "P429DT2M3.25S" (which specifies a duration of 429 days, 2 minutes, and 3.25 seconds).
- The Location attribute will be a String property in the Integrator ETL metadata, but will use a Custom property of `mdex:geocode` in order to create a Geocode standard attribute. Its value is "42.365615 - 71.075647" (which specifies a location at 42.365615 north latitude, 71.075647 west longitude).



To create a Custom property:

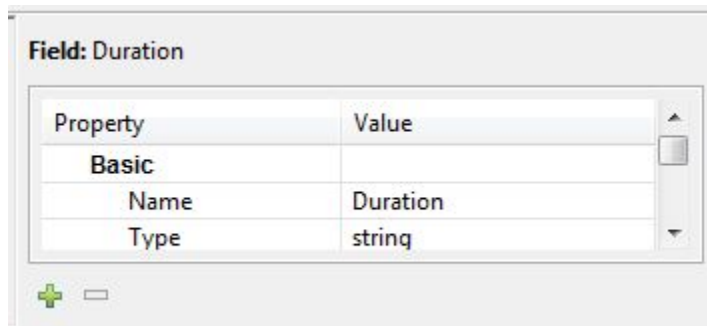
1. Create a graph with at least one reader, an Information Discovery component (such as the **Add/Update Records** component), and an Edge component.
2. Right-click on the Edge and select **New metadata>Extract from flat file**.
3. In the Flat File dialog, select the input file and then click **Next** to display the Metadata editor.
4. In the middle pane of the Metadata editor:
  - (a) Check the **Extract names** box.
  - (b) Click **Reparse**.
  - (c) Click **Yes** in the Warning message.

At this point, the Record pane of the Metadata editor should look like this:



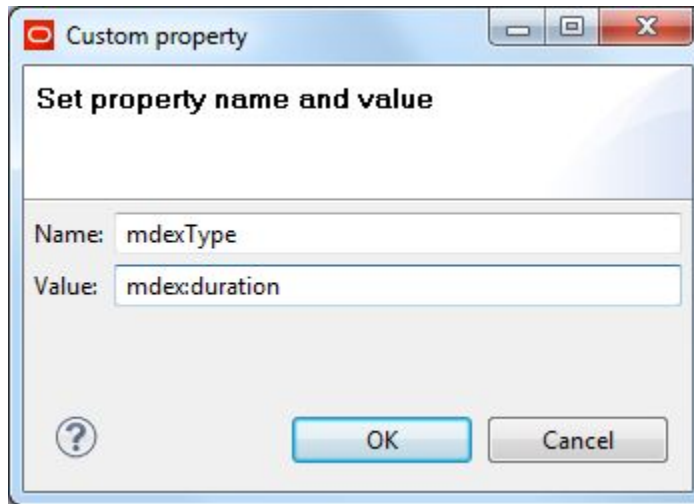
5. In the Record pane of the Metadata editor, make these changes:
  - (a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a more descriptive name.
  - (b) Change the ProductKey Type to **integer**.
  - (c) Leave the ProductName Type as **string**.
6. To create a Custom property type for the Duration property:
  - (a) In the Record pane, click the Duration property to high-light it.

The Duration property is displayed in the Field pane on the right, as in this example:



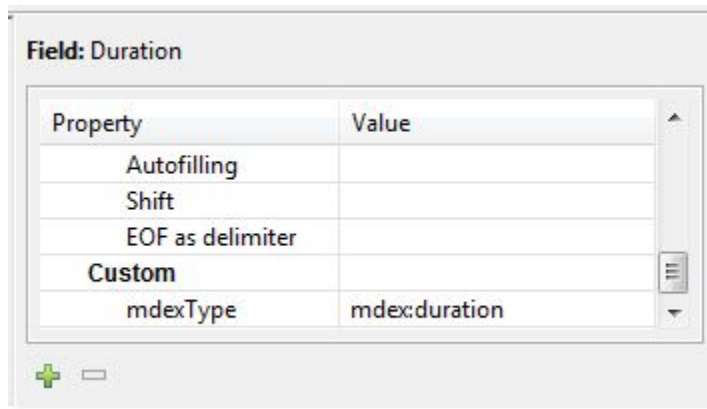
- (b) In the Field pane, click the green + icon to bring up the Custom property editor.  
 (c) Enter **mdexType** in the Name field and **mdex:duration** in the Value field.

The Custom property editor should look like this:



- (d) Click **OK** in the Custom property editor.

As a result, a Custom section (with the new **mdexType** property) is added to the Duration property in the Field pane:



7. Repeat Step 6 if you want to create another **mdexType** Custom property type for another of your source properties.

For example, for the Location attribute, you would create an **mdexType** Custom property with **mdex:geocode** in the Value field.

8. Click **OK** to apply your changes and close the Metadata editor.

As mentioned above, when the graph is run to add records, the Dgraph will use the **mdexType** Custom properties to create the standard attributes.

Keep in mind that you can create **mdexType** Custom properties for any of the MDEX property types, by setting the Value field to:

- `mdex:boolean` for Booleans

- `mdex:dateTime` to represent the date and time to a resolution of milliseconds
- `mdex:double` for floating-point values
- `mdex:duration` to represent a length of time with a resolution of milliseconds
- `mdex:geocode` to represent latitude and longitude pairs
- `mdex:int` for 32-bit signed integers
- `mdex:long` for 64-bit signed integers
- `mdex:string` for XML-valid character strings
- `mdex:time` for time-of-day values to a resolution of milliseconds

## Specifying multiple record delimiters

By using an OR operator, you can specifying multiple record delimiters in the metadata.

In the Edge metadata, the default record delimiter for a file depends on which operating system was used to create the file. For example, the default record delimiter for a Windows file is `\r\n`, while `\n` is typically used for Linux files.

However, you may have files that were created on different platforms (for example, if you have input files that you check out of a version control system, the files' line endings will vary according to the platform). In this case, you would want the record delimiter to be set to both values, so that you could use the same graph on Windows or Linux. You would then set the record delimiter to:

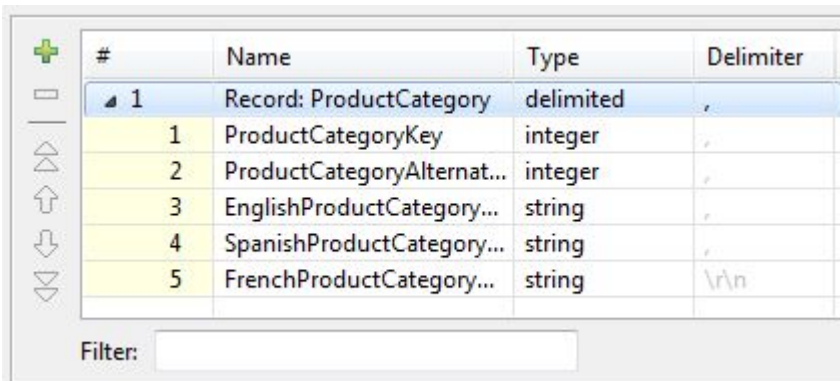
```
\r\n\\|\n
```

The `|` (pipe) character is an OR operator and the `\\` syntax is a way to escape that OR operator in the Integrator ETL interface.

To specify multiple record delimiters in the metadata:

1. In the Record pane of the Metadata Editor, click the first row (the Record row).

In this example, you would click the `Record:ProductCategory` row.



#	Name	Type	Delimiter
1	Record: ProductCategory	delimited	,
1	ProductCategoryKey	integer	,
2	ProductCategoryAlternat...	integer	,
3	EnglishProductCategory...	string	,
4	SpanishProductCategory...	string	,
5	FrenchProductCategory...	string	\r\n

Filter:

- In the Details pane (to the right of the Record pane), check the **Record delimiter** property to see the default setting.

In this example, `\r\n` is set as the record delimiter.

The screenshot shows a window titled "Field: ProductCategory" with a table of properties. The "Record delimiter" property is set to "\r\n".

Property	Value
<b>Basic</b>	
Name	ProductCategory
Type	delimited
Record delimiter	\r\n
Default delimiter	,
Skip source rows	1
Description	

- Place the cursor in the Value field of the **Record delimiter** property, and select `\r\n\\n` from the drop-down menu.

The Details pane should now look like this:

The screenshot shows the same window as before, but the "Record delimiter" property is now set to "\r\n\\n".

Property	Value
<b>Basic</b>	
Name	ProductCategory
Type	delimited
Record delimiter	\r\n\\n
Default delimiter	,
Skip source rows	1
Description	

- Click OK to save your changes made in the Metadata Editor.

## Creating data domains

This section describes options for creating a data domain.

Two options are available for creating data domains:

- server command
- Web services

For complete details about creating a data domain, see "Adding a new data domain" in the *Oracle Endeca Server Cluster Guide*.

After creating the data domain, you may want to configure it. See [Configuring Data Domains on page 62](#).

## Loading and maintaining data in a data domain

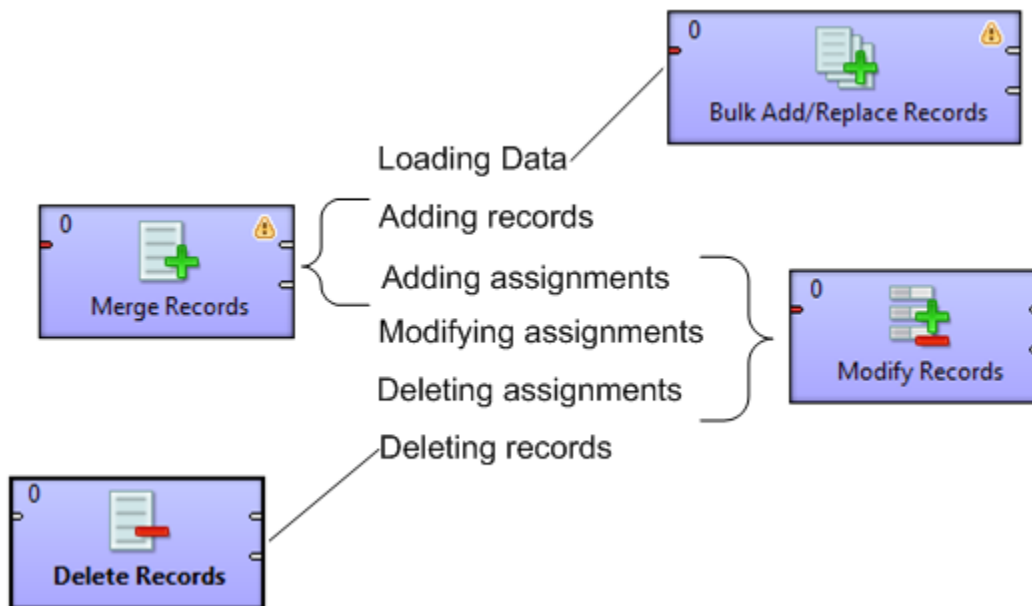
Integrator ETL provides four components to load and maintain data in a data domain.

The four data-loading components are:

- **Bulk Add/Replace**
- **Merge Records**
- **Modify Records**
- **Delete Records**

Before populating the data domain, you may want to configure it. See [Configuring Data Domains on page 62](#).

The following graphic illustrates data loading and maintenance tasks and the Integrator ETL component you should use for each:



- Use the **Bulk Add/Replace Records** component to load new records into a data domain, or to replace existing records.
- Use the **Merge Records** component to:
  - Add new records to a data domain
  - Add new assignments to existing records in a data domain

Note that this component can only operate on one Endeca record per input record; in other words, new assignments can only be added to one Endeca record per input record.

- Use the **Modify Records** component to:
  - Add new assignments to records
  - Replace the values in assignments of records with new values
  - Delete assignments from records

This component can modify multiple Endeca records per input record. In other words, multiple records can be deleted

- Use the **Delete Records** component to delete specific records from a data domain.

[Choosing a data loading strategy](#)

[Choosing a loader](#)

[Processing multi-value data](#)

## Choosing a data loading strategy

Strategies for loading data into Endeca Server fall into two general categories: baseline updates and incremental updates.

A baseline update loads data into an empty data domain. Baseline updates include full initial load of data and subsequent baseline updates. An incremental update adds records to an existing data domain, while an incremental update with a delta also modifies existing records in a loaded data domain.

The following table describes these strategies in detail.

Table 3.1: Data loading strategies

Data loading strategy	When to use this strategy
<p>A <b>full initial load</b> is a simple load of data into an empty data domain. This update is also known as a baseline update. Its basic idea is the simple loading of data, without the need to preserve any previously configured settings. It includes loading data into an empty Endeca data domain. This update assumes that the Endeca data domain is empty, and that the configuration and schema have only their default values acquired when the Endeca data domain was first created.</p> <p>As an example, the Baseline graph from the Getting Started project performs an initial data load, after loading a basic attribute schema.</p>	<p>Use this strategy the first time you load data into a new data domain. This strategy typically uses the <b>Bulk Add/Update Records</b> component to load new records quickly.</p>

Data loading strategy	When to use this strategy
<p>Subsequent baseline, also known as a <b>re-baseline</b>, completely replaces the existing contents of an existing data domain while preserving the configuration and schema.</p> <p>A typical subsequent baseline includes the following graphs:</p> <ul style="list-style-type: none"> <li>• An export configuration graph to preserve the existing configuration of the data domain. This graph uses the <b>Export Config</b> component.</li> <li>• A reset graph to reset the data domain to a pristine state. This graph uses the <b>Reset</b> component.</li> <li>• An import configuration graph that imports the previously exported configuration into the data domain after reset. This graph uses the <b>Import Config</b> component.</li> <li>• A set of graphs to reload the data to the reconfigured data domain.</li> </ul> <p>Typical practice is to run these graphs within an outer transaction using the <b>Transaction RunGraph</b> component.</p>	<p>Use this strategy to refresh the data domain periodically to ensure data is up to date and not corrupt.</p>
<p>An <b>incremental update</b> adds new records to a data domain, and new properties to existing records.</p>	<p>Use this strategy when you want to add new records to a data domain, or to add new assignments to existing records. This strategy typically uses the <b>Merge Records</b> component. Note that this component adds or modifies only one Endeca record per input record.</p>
<p>An <b>incremental update with delta</b>:</p> <ul style="list-style-type: none"> <li>• Adds new assignments to existing records</li> <li>• Modifies existing assignments on existing records</li> <li>• Deletes assignments from a record</li> <li>• Deletes complete records</li> </ul>	<p>Use this strategy when you want to modify or remove specific records. This strategy uses either the <b>Modify Records</b> component (to add, modify, or remove assignments) or the <b>Delete Records</b> component (to delete complete records). Note that both components can modify or delete multiple Endeca records per input record.</p>

## Choosing a loader

Integrator ETL provides two loader components to support the loading of data into an Endeca data domain: The **Bulk Add/Update Records** component (often called the "Bulk Loader") and the **Merge Records** component.

When loading data, at least a portion of processing resources is devoted to load processing; as a result, query performance is reduced while data is loading. The **Merge Records** component uses the Data Ingest Web

Service, which consumes only a portion of processing resources. Thus, while query performance is reduced, some query processing can continue. The Bulk Loader uses the Endeca server's Bulk Load API, which consumes all processing resources while running. Query processing is essentially placed on hold during bulk loading and resumes once the bulk load processing is complete.

Use the **Bulk Add/Replace Records** component to load data in bulk when it is acceptable for the visibility of updates to be delayed and for query processing to stop during the load processing.

Specific situations when you would choose this component include:

- You are performing the initial load of records into the data domain, whether or not an attribute schema has been configured. If the schema has not been configured (in other words, if no PDRs have been loaded) and no user data has been loaded previously, all new properties are created with default system values.
- You are adding new records to the data domain any time after the initial upload. Any new standard attributes that do not exist in the data domain are automatically created with default system values.
- You want to replace existing records in the data domain. When you use the **Bulk Add/Replace Records** component, if a loaded record matches an existing record, the loaded record overwrites (completely replaces) the existing record.

Note that this component cannot be used to load:

- The Global Configuration Record (GCR)
- Property Description Records (PDRs)
- Dimension Description Records (DDRs)
- Managed attribute values (mvals)
- Any data domain configuration documents

Use the **Merge Records** component to add or update small numbers of records, or to add or modify records when you want query processing to continue during the load (with reduced performance) and can accept a longer loading time in exchange.

Specific situations when you would choose this component include:

- You are loading the attribute schema.
- You are incrementally updating the data domain with new records after the initial load of records. Any new attributes that do not exist in the data domain are automatically created with system defaults.
- You are incrementally updating existing records in the data domain following initial upload. The behavior depends on the multi-assign configuration of the standard attribute. If the single-assign property is configured as false, uploaded data is totally additive; in other words, the loaded key-value pair will be merged into the existing record. If the single-assign property is configured as true (which is the default), and additional values are uploaded for an existing assignment, the operation fails.
- You are adding new records to the data domain any time.

Note that this component cannot be used to load:

- The Global Configuration Record (GCR)
- Managed attribute values (mvals)
- Any data domain configuration documents



In general:

- Choose the **Bulk Add/Replace Records** component to load, add, or replace a large number of records, and it is acceptable for query processing and the visibility of changes to be delayed. (Thus, you may want to consider scheduling such operations outside of business hours or during weekends.)
- Choose the **Merge Records**, or **Modify Records** components to load, add, or update smaller numbers of records, when you want query processing to continue during load processing (as noted above, query processing performance will still be reduced) and you want the changes to be visible immediately.

## Processing multi-value data

You can load multi-value data into a data domain by specifying the delimiter used to separate multiple values. But if you want to process multi-value data before loading it, convert it to a list data type

Two options are available for loading multi-value attributes:

- If the multi-value field is string data, you can specify the delimiter character used to separate the values. This option is the recommended approach, especially when loading multi-value data from flat files (such as .csv files) or from database input.
- Multi-value data can be converted to a list container. This option is recommended when loading multi-value data from complex input (such as JSON or XML) or if you need to process the individual values in the multi-value

To convert the input data to a list data type, use a **Transform** component. Within the **Transform** component, use the `split()` CTL function to convert the input of a multi-value field into a list data type. This function takes two parameters: the input string to be converted to a list, and a regular expression. The input is split whenever a match is found.

For example, suppose the value of the `DimEmployee_DepartmentName` field is multi-value, and uses dashes as the value separator. To generate a list data type for this input, you might use the following CTL expression: `split($in.0.DimEmployee_DepartmentName, "-")`.

The configuration of new attributes defaults to single-assign. You should configure attributes to support multi-assign prior to loading data.

## Load graphs

Loading graphs share a similar implementation, whether they were developed for initial load of data or for a subsequent baseline. The Integrator ETL Sample Applications include a full load graph, named `LoadData`, that you can use as a model for building your own load graphs.

You can load data either before or after loading the attribute schema, as described in [Configuring Attributes on page 66](#). If you load data before loading the attribute schema, records will be created with default values for standard attributes. You will likely find that you go through several iterations of loading data and reconfiguring your attribute schema before you determine the configuration required to meet your needs. In each iteration, you will delete the data in the data domain, update and reload the configuration and attribute schema, and reload the data.

Load graphs generally consist of the following components:

- One or more data reader components

Reader components read the source data that you want to load. Which reader you use depends on the nature of the data source you want to read. In the LoadData graph, **Universal Data Reader** components are used to read from the source .csv files. If you are loading database data or data stored as XML, however, you should use components appropriate to those data sources.

For details about adding a reader component to a graph, see [Adding a new component on page 21](#).

- Optionally, a **Reformat** component

If your incoming data includes a usable primary key, this component is unnecessary. If you want to define a primary key as a combination of multiple incoming values, the **Reformat** component is required.

For details about configuring a **Reformat** component to create a primary key, see [Configuring a Reformat component to generate a primary key on page 52](#).

- One or more joiner components

If you are combining data from multiple incoming data sources (such as the multiple .csv files in the LoadData graph), joiner components are required. The exact number and configuration of joiners depends on the number of incoming data streams and the way you want to combine the data from the different input streams.

- One data writer component

The data writer component writes the processed data to the Endeca data domain. As illustrated in the LoadData graph, the **Bulk Add/Replace Records** component is usually used for this purpose in full initial load graphs. See [Bulk Add/Replace Records component on page 156](#).

[Source data format](#)

[Configuring a Reformat component to generate a primary key](#)

[Configuring the Bulk Add/Replace Records component for initial load](#)

## Source data format

Integrator ETL reader components can read a variety of formats, including delimited, JDBC, and XML.

Production Information Discovery applications usually read data directly from databases or from database extracts.

The FullLoad sample application uses a two-dimensional format similar to the tables found in typical database management systems; the data is organized into tables, which consist of rows of records. Each row consists of a set of columns that represent the source properties and property values for each record. (This type of format is often called a "rectangular data format".)

The following image illustrates how the source data for the FullLoad graph is organized in a two-dimensional format:

	A	B	C	D	E
1	FactSales_ProductKey	FactSales_OrderDateKey	FactSales_DueDateKey	FactSales_ShipDateKey	FactSales_ResellerKey
2	349	20050701	20050713	20050708	676
3	350	20050701	20050713	20050708	676
4	351	20050701	20050713	20050708	676
5	344	20050701	20050713	20050708	676
6	345	20050701	20050713	20050708	676
7	346	20050701	20050713	20050708	676
8	347	20050701	20050713	20050708	676
9	229	20050701	20050713	20050708	676
10	235	20050701	20050713	20050708	676
11	218	20050701	20050713	20050708	676

## Primary key attribute

In the Endeca Server, the primary key is also called the “record spec”. You can use a standard attribute as a primary key for your records if your records include a property that will be unique for each record. (For more information on primary keys, see the *Oracle Endeca Server Data Loading Guide*.)

In the example LoadData graph, none of the source files includes a field with unique values for all records. Thus, the graph includes a **Transform** component named **CreateSpec** that creates the primary key (named FactSales\_RecordSpec) by concatenating the values of two attributes.

The name of the primary key must be added to the metadata definition applied to the edge that joins the **Transform** component to the next component in the graph flow.

## Use of hyphens in input property names

Although the Dgraph will accept attribute names with hyphens (because hyphens are valid NCName characters), Integrator ETL will not accept source property names with hyphens as metadata. Therefore, if you have a source property name such as "Ship-Date", make sure you remove the hyphen from the name.

## Using multi-assign data

Source data may include properties that have more than one value; such properties are known as multi-assign properties. For example, instead of having two properties (such as Color1 and Color2), the data may include one property (Color) with multiple values, as in the following example:

```
ComponentID|Color|Size
123|Blue|Medium
456|Blue;Red|Small
789|Red;Black;Silver|Large
```

In the example, the pipe character (|) is the delimiter between the properties, while the semi-colon (;) is the delimiter between multiple values in a given property. For example, the Color property for record 789 has values of "Red", "Black", and "Silver".

When configuring the **Bulk Add/Replace Records** component, you can then specify that the semi-colon is to be used as the delimiter for multi-assign properties.

Keep in mind that an Endeca attribute that is multi-assign must have the `mdex-property_IsSingleAssign` property set to `false` in its PDR. The default value of the property is `false`, which means the attribute is enabled for multi-assign by default.

## Configuring a Reformat component to generate a primary key

The typical practice when generating a primary key is to use CTL to concatenate the value of two or more standard attributes to generate the key value.

The following code illustrates a simple example derived from the LoadData graph:

```
//#CTL2
// Transforms input record into output record.
function integer transform() {
    $0.* = $0.*;
    $0.FactSales_RecordSpec = $0.FactSales_SalesOrderNumber+"-"+$0.FactSales_SalesOrderLineNumber;

    return ALL;
}
```

The first line:

```
$0.* = $0.*;
```

copies all of the attributes and values loaded from the data source.

The second line:

```
$0.FactSales_RecordSpec = $0.FactSales_SalesOrderNumber+"-"+$0.FactSales_SalesOrderLineNumber;
```

generates the primary key (here named `FactSales_RecordSpec`) by concatenating the values of the `FactSales_SalesOrderNumber` and `FactSales_SalesOrderLineNumber`.

The last line outputs all results.

## Adding a primary key to metadata

When you generate a primary key, you must add it to the metadata added to the edge that connects the **Reformat** component to the following component in the graph flow. Add a new property in the metadata editor to include the primary key.

## Configuring the Bulk Add/Replace Records component for initial load

This topic describes the configuration of the **Bulk Add/Replace Records** component when used in an initial load graph.

When configuring the **Bulk Add/Replace Records** component, you must configure the following properties:

- **Spec Attribute**

Enter the name of the standard attribute that you are using as a primary key. This may be the name of a property derived from the input data if any properties have a unique value for all records; otherwise, it will be the name of the primary key you defined in the **Reformat Transformation** component. See [Configuring a Reformat component to generate a primary key on page 52](#).



**Note:** When you specify the Spec Attribute, do not prepend the Collection key. When you check the **Prefix Attributes With Collection Key** box, the collection key will be prepended automatically. If you prepend the collection key when specifying the Spec attribute, it will be prepended twice.

- **Collection key**

Enter the collection key of the collection to which you want to add the uploaded data. If the key does not exist, it will be created. Note that if you are loading data to a collection that already exists, the value of the **Spec Attribute** property must match the spec attribute of the existing collection.

- **Prefix Attributes With Collection Key**

Studio requires that all attributes have unique names. To ensure that the name of each property is unique to its collection, recommended practice is to prepend the collection name to the property name. Check this box to prepend the collection name to the attributes loaded into the data domain.

- **Post Ingest Query Optimization**

This field determines when data merging occurs. If the value of the field is `true` data merge is performed immediately after the ingest operation finishes. If the value is `false`, data merge is disabled and the data will be merged later according to the merge policy of the data domain. For additional details, see the "Post ingest behavior" subsection of [Bulk Add/Replace Records component on page 156](#).

Defaults to `true`.

- **Post Ingest Dictionary Update**

This field determines when the spelling dictionary will be updated. It defaults to `true`. If the value of the field is `true`, the dictionary is updated immediately after the ingest operation finishes. If the value of the field is `false`, updating of the dictionary is disabled; you can manually update the dictionary later, by using the `updateSpellingDictionaries` operation of the Data Ingest Web Service.

- **SSL Enabled**

This field toggles communication with the Server and data domain via SSL. Only set this property to `true` if SSL has been enabled on both the Endeca Server and the data domain. For details, see the *Oracle Endeca Server Administrator's Guide*.

Defaults to unchecked.

- **Multi-assign delimiter**

If any of the attributes being loaded allow and contain multi-assign values, specify the character that separates the multi-assign values in this field. Note that this delimiter is a different delimiter from the one used to separate records.

## Adding records and assignments

Use the **Merge Records** component to add new records to the data domain, or to add new assignments to existing records.

The **Merge Records** component uses the Data Ingest Web Service (DIWS) to load data. Thus, query processing can continue while you load records, although query performance is diminished. Only one Endeca record is modified per input record, however, and loading processing is not as fast as the Bulk Load Interface.

The following table describes the behavior of this component:

Table 3.2: Merge Records component behavior

Does input record exist?	Are any of the input properties single-assign in the data domain?	Result
Input record does not exist	Not applicable.	New record is created with the data defined in the input.
Input record does exist	No	New values are appended to the assignments in the record.
Input record does exist	Yes	Error; the record is not modified.

You can configure the **Merge Records** component to add records only. In this configuration, if an input record matches an existing record, the operation fails.

[Merge Records input](#)

[Merge Records graph](#)

[Configuring the Merge Records component](#)

## Merge Records input

The input to the **Merge Records** component is an arbitrary array of property names and values that you want to use to select the records you want to modify, and the values you want to add to the associated properties in the file.

You can use a delimited file, such as a comma-separated value file, or derive the input data from a database or similar source. The following code illustrates a simple example:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-6738|White|500|375
CA-7457|Black|500|375
CB-2903|Silver|1000|750
CN-6137|Silver|1000|750
CR-7833|Gold|1000|750
FH-2981|Silver|500|375
```

## Merge Records graph

A Merge Records graph usually consists of two components: a reader component and the **Merge Records** component.

Choose the appropriate reader component for your input. For example, if you derive your input from a database, use the **DBInputTable** component. If you use a delimited file, the **Universal Data Reader** component is a good choice.

For details about the **Merge Records** component, see [Merge Records component on page 178](#).

The two components are joined by a basic edge. For details, see [Connecting two components with an edge on page 26](#).

## Configuring the Merge Records component

This topic describes how to configure the **Merge Records** component to load records and assignments.

To configure the **Merge Records** component, specify the following properties:

- **Endeca Server Host**

Enter the name of the machine on which the Endeca Server is running.

- **Endeca Server Port**

Enter the port of the Endeca Server

- **Endeca Server Context Root**

Enter the WebLogic application root context of the Endeca Server.

- **Data Domain Name**

Enter the Endeca data domain to which you want to merge record changes.

- **Collection key**

Enter the collection key of the collection to which you want to merge record changes.

- **Spec Attribute**

Specify the attribute from the input array to use to select the records to which to add assignments. If no record matching the value of this property exists, a new record is created.

- **SSL Enabled**

Set this field to true if you want to enable SSL for this component. Only enable SSL on the component if you have enabled SSL on the Endeca Server.

- **Batch Size (Bytes)**

Enter an integer greater than zero to set the batch size, in bytes. To disable batching, specify zero (0) or a negative integer.

- **Multi-assign delimiter** Enter the character you want to use to separate values when a property allows multi-assign. This delimiter must be different from the delimiter that separates property fields on the source record.

- **Maximum number of failed batches**

Enter a positive integer specifying the largest number of batches that can fail before the operation as a whole is terminated. To configure the operation to fail if any batch fails, enter zero (0).

- **Disable Updates**

Check this box to prevent updates to existing records. If an input record matches an existing record, the operation fails.

## Modifying records

Use the **Modify Records** component to modify the assignments on a record. You can add new assignments, replace existing assignments, or delete assignments from the record.

Each instance of the **Modify Records** component can perform one operation. If you want to perform multiple operations on your records, you need to implement a different instance of the component for each operation. Recommended practice is not to run different operations concurrently; in other words, do not run an add operation and a delete operation at the same time. Different operations should be run in different phases or in different graphs.

The input to the component specifies the properties you want to modify, and the values you want to add to the records or delete from them.

Two options are available to select the records to modify:

- You can include the spec attributes in the input to the component to specify the records to modify. Note that you cannot modify an attribute you use to select records to modify. For example, if you use an attribute to select a record, you cannot delete that attribute's assignment from the record.
- You can enter an Endeca Query Language (EQL) record specifier (`WHERE` clause) to select the records to delete.

### Add Assignments

When the value of the **Operation** property is `Add Assignments`, if an attribute specified in the input array does not have a value, the attribute is added to the matching record with the value specified in the input array. If an attribute specified in the input array does have a value, the behavior depends the attribute is single-assign. If the attribute is not single-assign, the new value is appended to the existing values of the property. If the attribute is single-assign, an error occurs and the operation fails; this causes the whole batch to fail.

### Replace Assignments

When the value of the **Operation** property is `Replace Assignments`, if an attribute specified in the input array does not have a value, the attribute is added to the record with the value specified in the input array. If an attribute specified in the input does have a value, any existing values are replaced by the values derived from the input array. If the input value is null, the null value overwrites all existing values for the assignment. (In effect, replacing with a null value is a wildcard delete.)

### Delete Assignments

When the value of the **Operation** property is `Delete Assignments`, and an attribute in the input array includes one or more values, those values are deleted from the specified attribute in all records that match the Record Filter. If no value is specified for a property in the input array, no values are deleted.

[Modify Records input](#)

[Modify Records graph](#)

[Specifying the Record Set Specifier Attributes](#)



## Modify Records input

The input to the **Modify Records** component is an arbitrary array of property names and values that you want to use to select the records you want to modify, and the values you want to add to the associated properties in the file.

You can use a delimited file, such as a comma-separated value file, or derive the input data from a database or similar source. The metadata on the output edge from the reader component defines the names of the properties.

### Using an input array

The input array for the **Modify Records** component is an arbitrary array of property names and values that specify the records to modify and the values to add to the records or delete from them. Note that null values are added to the specified records; null values overwrite all existing values.

The examples below use the following input array:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-6738|White|500|375
CA-7457|Black|500|375
CB-2903|Silver|1000|750
CN-6137|Silver|1000|750
CR-7833|Gold|1000|750
FH-2981|Silver|500|375
```

Assume `ProductKey` is the spec attribute.

If the value of the **Operation** property is `Add Attributes`, and the records do not include values for the `Color`, `SafetyStockLevel`, and `ReorderPoint` properties, then the input values for these properties are added to the records specified by the `ProductKey` properties. If the records already include values for these properties, and the properties support multi-assign, the input values are appended to the existing values for these properties. If the properties do not support multi-assign, the operation fails; as a result, the entire batch fails.

If the value of the **Operation** property is `Replace Attributes`, the values in the input array replace any existing values of the specified properties in the records specified by the `ProductKey`. If the value of any input property is null, the null value overwrites all existing values in the property. For example, suppose the input array included the following record:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-6738|||
```

Then the values of the `Color`, `SafetyStockLevel`, and `ReorderPoint` properties are overwritten with null values. (In other words, this operation is effectively a wildcard delete operation for these properties.)

If the value of the **Operation** property is `Delete Attributes` and the following array is input (`ProductKey` is the spec attribute):

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-7457|Black||
```

then:

- For the record where `ProductKey=CA-5965`, the input values are deleted from the specified properties.
- For the record where `ProductKey=CA-7457`, the assignment `Black` is deleted from the `Color` property.

## Using EQL to select the records to modify

You can use an EQL record specifier (the `WHERE` clause of an EQL query) to select the records to modify. Use variables to specify the input properties to use to select. For example, if you input the following array:

```
ProductCategory|ProductSubcategory|SafetyStockLevel|ReorderPoint
Components|Hardware|250|185
Components|Wheels|25|18
Clothing|Hats|40|35
Clothing|Shirts|25|20
```

you could enter the following EQL: `"EnglishProductCategoryName"=$input.ProductCategory` and `"EnglishProductSubcategoryName"=$input.ProductSubcategory`

For full details about Endeca Query Language, see the *Oracle Endeca Server EQL Guide*.

## Modify Records graph

A Modify Records graph usually consists of two components: a reader component and the **Modify Records** component.

Different operations should not be run at the same time in a data domain. Recommended practice is to implement different graphs for each operation (`Add Attributes`, `Replace Attributes`, `Delete Attributes`) rather than combining operations in one graph. If combined in one graph, different operations should be configured to run in different phases.

Choose the appropriate reader component for your input. For example, if you derive your input from a database, use the **DBInputTable** component. If you use a delimited file, the **Universal Data Reader** component is a good choice.

For details about the **Modify Records** component, see [Modify Records component. on page 183](#)


The two components are joined by a basic edge. For details, see [Connecting two components with an edge on page 26](#).

## Specifying the Record Set Specifier Attributes

This topic describes how to specify the **Record Set Specifier Attributes** property on components that include it.

The component must be joined to an edge and metadata must be assigned to the edge before you can specify **Record Set Specifier Attributes**.

To define the **Record Set Specifier Attributes**:

1. Click in the **Record Set Specifier Attributes** field.  
Integrator ETL displays a browse button.
2. Click the browse button.  
Integrator ETL displays the **Edit key** dialog.
3. In the **Fields** table, select the input property that matches the name of the spec attributes.
4. Click the  button.

The attribute you selected is moved to the **Key Parts** table.

5. Repeat Steps 3 and 4 to move specifying additional attributes.
6. Click **OK**.

## Backing up and restoring data domains

Use operations in the Manage Web Service to back up and restore data domains.

To back up a data domain, use the `exportDataDomain` operation of the Manage Web Service.

To restore a data domain backup, use the `importDataDomain` operation of the Manage Web Service.

For details about these operations, see "Using the Manage Web Service" in the *Oracle Endeca Server Cluster Guide*.

Use the **Web Service Client** component to run these operations.

## Deleting data from the data domain

You can delete all data from the data domain, delete complete records from the data domain, or delete specific assignments from a record.

If you want to delete all data from a data domain, including data domain configurations, use the **Reset Data Domain** component to delete the data and reset the data domain to a pristine state. Create a graph and add the **Reset Data Domain** component. No other components are necessary. In addition to the Endeca Server Host, Endeca Server Port, and Endeca Server Context Root, specify the Data Domain Name of the data domain you want to reset. Also see [Reset Data Domain component on page 192](#).

If you want to delete individual records from a data domain, use the **Delete Records** component. For details, see [Deleting records from the data domain on page 59](#).

If you want to delete specific record assignments from individual records, use the **Update Records** component. For details see [Modifying Records on page 56](#).

[Deleting records from the data domain](#)

[Delete Records graph](#)

[Configuring the Delete Records component](#)

## Deleting records from the data domain

Use the **Delete Records** component to delete records from a data domain.

The **Delete Records** component deletes complete records that match criteria you define. Three options are available to select the records to delete:

- You can input a data array that defines the properties to use to select the records to delete.
- You can enter an Endeca Query Language (EQL) record specifier (`WHERE` clause) to select the records to delete.
- You can delete all records in a Collection.

These options are mutually exclusive. If you choose to use an EQL record specifier to select the records, for example, you cannot also use an input array, or delete records from a Collection.

If you want to delete all data from the data domain, including the configuration, use the **Reset Data Domain** component. For details about resetting a data domain, see [Deleting data from the data domain on page 59](#). If you want to delete specific record assignments from individual records, use the **Modify Records** component. For details, see [Modifying Records on page 56](#)

## Using an input array to select records to delete

The input to the **Delete Records** component is an arbitrary array of property names and values that you want to use to select the records you want to delete. You can use a delimited file, such as a comma-separated value file, or derive the input data from a database or similar source. The metadata on the output edge from the reader component defines the names of the properties.

Attributes are joined using an AND operator. Thus, the fewer properties included in the input array, the more general the selection and the more records will be deleted. Conversely, the more properties included in the input array, the more specific the selection and the fewer records will be deleted.

For the examples below, assume the following attributes have been defined in the data domain:

```
DimGeography_GeographyKey|DimGeography_City|DimGeography_StateProvinceCode|DimGeography_StateProvinceName|DimGeography_CountryRegionCode|DimGeography_CountryRegionName|DimGeography_PostalCode
```

- If the following array is input, Integrator ETL deletes all records where the value of the DimGeography\_StateProvinceName property is Alabama:

```
DimGeography_StateProvinceName
Alabama
```

- If the following array is input, Integrator ETL deletes records where the value of the DimGeography\_City property is Newton and the value of the DimGeography\_StateProvinceName is British Columbia

```
DimGeography_City|DimGeography_StateProvinceName
Newton|British Columbia
```

- If the following array is input, Integrator ETL deletes records where the value of the DimGeography\_City property is Newton and the value of the DimGeography\_StateProvinceName is British Columbia and the value of the DimGeography\_PostalCode is V2M1P1

```
DimGeography_City|DimGeography_StateProvinceName|DimGeography_PostalCode
Newton|British Columbia|V2M1P1
```

## Using EQL to select records to delete

You can use an Endeca Query Language (EQL) record specifier (the `WHERE` clause of an EQL query) to select the records to delete. For example: `"DimGeography_City"='Newton' AND "DimGeography_StateProvinceName"='British Columbia'` selects records where the value of the DimGeography\_City property is Newton and the value of the DimGeography\_StateProvinceName is British Columbia.



**Note:** Standard practice in EQL is to use double quotation marks around attribute names and single quotation marks around attribute values.

You can also use an input array to support an EQL record specifier. Use variables to specify the input properties. For example, if you input the following array:

```
City|StateProvince
```

```
Newton|British Columbia  
Townsville|Queensland  
Longmont|Colorado
```

You could enter the following EQL: "DimGeography\_City"=\$input.City and "DimGeography\_StateProvinceName"=\$input.StateProvince.

For full details about Endeca Query Language, see the *Oracle Endeca Server EQL Guide*.

## Deleting records from a Collection

To delete records from a Collection, check the **Truncate Collection** box and enter the **Collection key**.

Deleting records from a Collection does not take any input. Use the **Delete Records** component by itself. It does not take any input when deleting Collection records. If you add input when the **Truncate Collection** box is checked, the graph fails and error is returned.

Note that truncating a Collection only deletes records from the Collection. The Collection itself still exists.

## Delete Records graph

A Delete Records graph can include either one or two components.

If you choose to use an input array to select the records to delete, the graph consists of two components: a reader component and the **Delete Records** component. Choose the appropriate reader component for your input. For example, if you derive your input from a database, use the **DBInputTable** component. If you use a delimited file, the **Universal Data Reader** component is a good choice. The two components are joined by a basic edge. For details, see [Connecting two components with an edge on page 26](#).

If you choose to use EQL to select the records to delete, the graph consists of only the **Delete Records** component.

For details about the **Delete Records** component, see [Delete Records component on page 163](#).

## Configuring the Delete Records component

This topic describes how to configure the **Delete Records** component to delete records from the data domain.

To configure the **Delete Records** component, specify the following properties:

- **Endeca Server Host**  
Enter the name of the machine on which the Endeca Server is running.
- **Endeca Server Port**  
Enter the port of the Endeca Server.
- **Endeca Server Context Root**  
Enter the WebLogic application root context of the Endeca Server.
- **Data Domain Name**  
Enter the Endeca data domain from which you want to delete records.
- **SSL Enabled**

Set this field to true if you want to enable SSL for this component. Only enable SSL on the component if you have enabled SSL on the Endeca Server.

- **Batch Size (Bytes)**

Enter an integer greater than zero to set the batch size, in bytes. To disable batching, specify zero (0) or a negative integer.

- **Maximum number of failed batches:**

Enter a positive integer specifying the largest number of batches that can fail before the operation as a whole is terminated. To configure the operation to fail if any batch fails, enter zero (0).

- Do one of the following:

- If you want to use an input array to specify the records to delete, specify the name of the spec attribute in the **Record Set Specifier Attributes** field. For details, see [Specifying the Record Set Specifier Attributes on page 58](#).
- If you want to use an EQL statement to specify the records to delete:
  1. Check the **Use EQL Record Set Specifier** box
  2. Enter the EQL record specifier (*WHERE* clause) in the **EQL Record Set Specifier** field.
- If you want to delete all records from a collection, check the **Truncate Collection** box and enter the **Collection Key** of the collection whose records you want to delete.



## Chapter 4

---

# Configuring Data Domains

This section describes how to build graphs to configure data domains.

[About configuring data domains](#)

[Global Configuration Record](#)

[Configuring attributes](#)

[Configuring record search](#)

[Configuring value search](#)

[Configuring relevance ranking](#)

[Configuring stop words](#)

[Configuring the thesaurus](#)

[Configuring precedence rules](#)

[XML-based configuration graphs](#)

[CSV-based configuration graphs](#)

[Exporting and importing data domain configurations](#)

## About configuring data domains

All data domain configuration procedures are optional.

Each data domain is created with a set of defaults. When you load records, default configurations are also created. You only need to load data domain configurations if the default configurations do not meet your needs. At that time, you only need to load the specific configurations required to meet your specific needs. For example, if you do not need additional stop words beyond the defaults, you do not need to load a stop words configuration document.

For additional information on data domain configurations, see the following sections of the *Oracle Endeca Server Developer's Guide*:

- "About the data model", particularly the sections on "Attributes", "Managed Attributes", and "System records"
- "Search Interfaces"
- "Dgraph Configuration Reference"

## Global Configuration Record

The Global Configuration Record (GCR) stores global configuration settings for the Endeca data domain.

The GCR sets the configuration for wildcard search enablement, search characters, merge policy, and spelling correction settings. A full description of its properties and their default values is available in the *Oracle Endeca Server Developer's Guide*.

Note the following requirements of the GCR:

- The `mdex-config_Key` property must be unique and single-assign. The value of this property must be `global`.
- The GCR input file must contain valid values for all of GCR properties. No property can be omitted. If you omit any property, the load graph fails with an error.
- The GCR input file cannot include arbitrary, user-defined properties. If you add any arbitrary values, the load graph fails with an error.



**Note:** If you change any of the spelling settings, rebuild the Aspell dictionary using the `updateSpellingDictionaries` service of the Data Ingest Web Service. Run the following Web Service request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ingest="http://www.endeca.com/MDEX/ingest/2/0">
  <soapenv:Header/>
  <soapenv:Body>
    <ingest:updateSpellingDictionaries>
      <ingest:OuterTransactionId>txId</ingest:OuterTransactionId>
    </ingest:updateSpellingDictionaries>
  </soapenv:Body>
</soapenv:Envelope>
```

You can run this request manually or use a Web Services Client to run it as part of a graph.

### Sample GCR input file

The following code illustrates a sample GCR:

```
<mdex:record>
  <mdex-config_Key>global</mdex-config_Key>
  <mdex-config_EnableValueSearchWildcard>true</mdex-config_EnableValueSearchWildcard>
  <mdex-config_MergePolicy>aggressive</mdex-config_MergePolicy>
  <mdex-config_SearchChars>+_</mdex-config_SearchChars>
  <mdex-config_SpellingRecordMinWordOccur>2</mdex-config_SpellingRecordMinWordOccur>
  <mdex-config_SpellingRecordMinWordLength>4</mdex-config_SpellingRecordMinWordLength>
  <mdex-config_SpellingRecordMaxWordLength>24</mdex-config_SpellingRecordMaxWordLength>
  <mdex-config_SpellingDValMinWordOccur>5</mdex-config_SpellingDValMinWordOccur>
  <mdex-config_SpellingDValMinWordLength>3</mdex-config_SpellingDValMinWordLength>
  <mdex-config_SpellingDValMaxWordLength>20</mdex-config_SpellingDValMaxWordLength>
</mdex:record>
```

This GCR:

- Enables wildcard search by setting the `mdex-config_EnableValueSearchWildcard` property to `true`.
- Sets the merge policy to `aggressive` via the `mdex-config_MergePolicy` property.
- Adds the plus (+) and underscore (\_) characters as search characters for value search and record search operations.

You can create the file in a text editor.



[Sample GCR input file](#)

[Creating the GCR graph](#)

[GCR Web service code](#)

## Sample GCR input file

The following code illustrates an example GCR input file.

```
<mdex:record>
  <mdex-config_Key>global</mdex-config_Key>
  <mdex-config_EnableValueSearchWildcard>true</mdex-config_EnableValueSearchWildcard>
  <mdex-config_MergePolicy>aggressive</mdex-config_MergePolicy>
  <mdex-config_SearchChars>+_</mdex-config_SearchChars>
  <mdex-config_SpellingRecordMinWordOccur>2</mdex-config_SpellingRecordMinWordOccur>
  <mdex-config_SpellingRecordMinWordLength>4</mdex-config_SpellingRecordMinWordLength>
  <mdex-config_SpellingRecordMaxWordLength>24</mdex-config_SpellingRecordMaxWordLength>
  <mdex-config_SpellingDValMinWordOccur>5</mdex-config_SpellingDValMinWordOccur>
  <mdex-config_SpellingDValMinWordLength>3</mdex-config_SpellingDValMinWordLength>
  <mdex-config_SpellingDValMaxWordLength>20</mdex-config_SpellingDValMaxWordLength>
</mdex:record>
```

This GCR:

- Enables wildcard search by setting the `mdex-config_EnableValueSearchWildcard` property to `true`.
- Sets the merge policy to aggressive via the `mdex-config_MergePolicy` property.
- Adds the plus (+) and underscore (\_) characters as search characters for value search and record search operations.

You can create the file in a text editor.

## Creating the GCR graph

The GCR input file is an XML file.

Use an XML-based configuration graph to load the GCR. No Transformer is necessary in the graph to load a GCR configuration. The **Denormalizer** component is still necessary. See [XML-based configuration graphs on page 84](#).

## GCR Web service code

Use the `putGlobalConfigRecord` operation of the Configuration Web Service to load the GCR.

The following code illustrates a typical example Web service request:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:putGlobalConfigRecord
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    $xmlString
  </config-service:putGlobalConfigRecord>
</config-service:configTransaction>
```

## Configuring attributes

The attribute schema of a data domain is defined by its Property Description Records (PDRs) and Dimension Description Records (DDR). Standard attributes are defined by PDRs, while managed attributes are defined by both PDRs and DDRs.

If you load data before defining an attribute schema, the loading interfaces (the Bulk Load Interface and Data Ingest Web Service [DIWS]) automatically create PDRs, and thus standard attributes, based on system defaults.

If you load managed attribute values before defining a managed attribute schema, the loading interface, the DIWS, automatically creates both the PDR (and thus a standard attribute) and the DDRs (and thus the managed attributes) based on system defaults.

Therefore, if the default configurations of the PDR and the DDR meet your needs, you may not need to load attribute configurations.

- If the default attribute schemas (the default configuration of both the PDR and the DDRs) meet your needs, you do not need to load either the standard attribute configuration or the managed attribute configuration. You can load the managed attribute values and the PDRs and DDRs will be created automatically.

For details about loading managed attribute values, see [Loading and modifying managed attribute values \(MVALs\) on page 72](#).

- If the default standard attribute schema (the configuration of the PDR) meets your needs but you need to customize the managed attribute schema (configuration of the DDRs):
  1. Load the managed attribute schema (DDR).  
For details, see [Loading the managed attribute schema on page 69](#).
  2. Load managed attribute values.  
For details, see [Loading and modifying managed attribute values \(MVALs\) on page 72](#).
- If you need to customize the standard attribute schema (the PDR configuration) but the default managed attribute schema (DDR configuration) meets your needs:
  1. Load the standard attribute schema (PDRs).  
For details see [Loading the standard attribute schema on page 67](#).
  2. Load managed attribute values.  
For details, see [Loading and modifying managed attribute values \(MVALs\) on page 72](#).
- If you need to customize both the standard attribute schema (the PDR configuration) and the managed attribute schema (the DDR configuration):
  1. Load the standard attribute schema (PDRs).  
For details see [Loading the standard attribute schema on page 67](#).
  2. Load the managed attribute schema (DDR).  
For details, see [Loading the managed attribute schema on page 69](#).
  3. Load managed attribute values.  
For details, see [Loading and modifying managed attribute values \(MVALs\) on page 72](#).

Note that these operations should be performed in the specified order.

1. If you want to customize the standard attribute schema (customize the PDR configuration), you should load the custom standard attribute schema first, to ensure that the standard attributes have been created and configured.
2. If you want to customize the managed attribute schema (customize the DDR configuration), you should load the managed attribute schema after loading the standard attribute schema (if applicable), and before loading managed attribute values.
3. Load managed attribute values after loading the custom standard attribute and managed attribute schemas (configuring the PDR and the DDRs), but before loading data.
4. Load data after all attribute configuration (as well as any other data domain configuration) is complete.

[Loading the standard attribute schema](#)

[Loading the managed attribute schema](#)

[Loading and modifying managed attribute values \(MVals\)](#)

## Loading the standard attribute schema

This topic describes implementation details of graphs to load the Property Description Records (PDRs) to create the standard attribute schema.

Use a CSV-based configuration graph to load the standard attribute schema. See [CSV-based configuration graphs on page 86](#).

### Input file

The PDR input file defines one or more Endeca standard attributes, with the specific settings of required and optional PDR properties. The input file resembles the following illustration:

	A	B	C	D
1	Key	DisplayName	TextSearch	SortOrder
2	FactSales_ProductKey	Product Key	FALSE	lexical
3	FactSales_OrderDateKey	Order Date Key	FALSE	lexical
4	FactSales_DueDateKey	Due Date Key	FALSE	lexical
5	FactSales_ShipDateKey	Ship Date Key	FALSE	lexical
6	FactSales_ResellerKey	Reseller Key	FALSE	lexical
7	FactSales_EmployeeKey	Employee Key	FALSE	lexical
8	FactSales_PromotionKey	Promotion Key	FALSE	lexical

The first line of the sample file is the header row, which specifies the properties defined in the input file. You must specify the value for the `mdex-property_Key`. Other `mdex-*` and system navigation properties are optional. Attribute properties you do not specify either use defaults or are not defined. For information on the system default values for standard attributes, see [Standard attribute default values on page 38](#).

In this example, the following properties are defined:

```
Key,DisplayName,TextSearch,SortOrder
```

The names of these properties are arbitrary; you can use different names in your input file if you choose (for example, you can use `AttrName` instead of `Key`). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).



**Note:** You cannot use hyphens in the names of standard attributes. Although the data domain will accept standard attribute names that include hyphens, Integrator ETL does not. If your input includes an attribute whose name includes a hyphen, change the name in the input file. For example, if you want to input an attribute named "Sales-Type", change it in the input file to something like "Sales\_Type".

The header properties map to these PDR properties:

Input Header Property	Maps to PDR Property
Key	mdex-property_Key
DisplayName	mdex-property_DisplayName
TextSearch	mdex-property_IsTextSearchable
SortOrder	system-navigation_Sorting

The second and following rows in the input file contain the values for the configuration properties.

## Standard attribute Transformer

In the configuration graph, use a **Reformat** component to build the XML to submit to the server to define your standard attribute schema. The XML consists of a set of PDRs (`<mdex:record>`), each of which comprises attribute properties (for example, `<mdex-property_Key>`, `<mdex-property_DisplayName>`).

The following code builds an XML from the input file illustrated above:

```
integer n = 1;
integer aggrKey = 0;

// Transforms input record into output record.
function integer transform() {
    string searchBool = "";
    string saRecord = "<mdex:record xmlns=\"\">";
    saRecord = saRecord + "<mdex-property_Key>" + $0.Key + "</mdex-property_Key>";
    saRecord = saRecord + "<mdex-property_DisplayName>" + $0.DisplayName + "<
/mdex-property_DisplayName>";

    // Lower case the boolean in the CSV file
    searchBool = lowerCase($0.TextSearch);
    saRecord = saRecord + "<mdex-property_IsTextSearchable>" + searchBool + "<
/mdex-property_IsTextSearchable>";

    saRecord = saRecord + "<system-navigation_Sorting>" + $0.SortOrder + "</system-navigation_Sorting>";
};

$0.xmlString = saRecord + "</mdex:record>";

// Batch up the web service requests.
$.singleAggregationKey = aggrKey;
n++;
if (n % 15 == 0) {
    aggrKey++;
}
```

```

}
return ALL;
}

```

First, the code defines two integer variables, the first as a loop counter, the second as the aggregation key (aggKey).

Next, the `transform()` function builds the PDRs by creating the `<mdex:record>` container, and populating it with nodes corresponding to the properties defined in the input file (`<mdex-property_Key>`, `<mdex-property_DisplayName>`). The value of each node is derived from the corresponding value in the input file.

Note that in this example, the value of the Text Search field, which is specified in ALL CAPS, is converted to lower case before it is added to the XML.

Finally, the records are grouped into batches of 15 for submission to the server. Submitting in batches ensures a continuous stream of processing through all components in the graph, which improves performance of both Integrator ETL and the Endeca Server.

## Web service request

Use the `updateProperties` operation of the Configuration Web Service to load the PDR configurations into the Endeca Server. This operation creates new standard attributes, and updates any affected existing standard attributes with new data. The following code illustrates a typical `updateProperties` operation to load PDRs:

```

<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/3/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:updateProperties
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    $xmlString
  </config-service:updateProperties>
</config-service:configTransaction>

```

This example includes two variables:

- `OUTER_TRANSACTION_ID`

This variable specifies the outer transaction ID for the request. The variable and its value are stored in the `workspace.prm` file of the Integrator ETL project.

- `$xmlString`

This variable contains the PDRs that have been constructed by the `Reformat` component in the graph.

This code is added to the **Web Services Client** component in the configuration graph.

## Loading the managed attribute schema

This topic describes implementation details of graphs to load the Dimension Description Records (DDR) to create the managed attribute schema.

Use a CSV-based configuration graph to load the managed attribute schema. See [CSV-based configuration graphs on page 86](#).

## Input file

The Dimension Description Record (DDR) has the same name as the associated standard attribute. It is used to create a hierarchy of standard attribute values. The input file resembles the following illustration:

	A	B	C	D
1	Key	Refinement	DimHierarchy	RecHierarchy
2	CAT_BIKES	TRUE	TRUE	TRUE
3	CAT_COMPONENTS	FALSE	FALSE	FALSE
4	CAT_CLOTHING	TRUE	TRUE	TRUE
5	CAT_ACCESSORIES	TRUE	FALSE	FALSE
6				

The first line of the sample file is the header row, which specifies the properties defined in the input file. In this example, the following properties are defined:

```
Key,Refinement,DimSearch,RecHierarchy
```

The names of these properties are arbitrary; you can use different names in your input file if you choose (for example, you can use `AttrName` instead of `Key`). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).

In this example, the header properties map to these DDR properties:

Input Header Property	Maps to PDR Property
Key	<code>mdex-dimension_Key</code>
Refinement	<code>mdex-dimension_EnableRefinements</code>
DimHierarchy	<code>mdex-dimension_IsDimensionSearchHierarchical</code>
RecHierarchy	<code>mdex-dimension_IsRecordSearchHierarchical</code>

## Managed attribute Transformer

In the configuration graph, use a **Transformer** component to build the XML to submit to the server to define your standard attribute schema. The XML consists of a set of DDRs (`<mdex:record>`), each of which is comprised of attribute properties (for example, `<mdex-dimension_EnableRefinements>`, `<mdex-dimension_IsDimensionSearchHierarchical>`, and `<mdex-dimension_IsRecordSearchHierarchical>`).

The following code builds an XML from the input file illustrated above:

```
//#CTL2

integer n = 1;
integer aggrKey = 0;

// Transforms input record into output record.
function integer transform() {
```

```

string maBool = "";
string maRecord = "<mdex:record xmlns=\"\">";
maRecord = maRecord + "<mdex-dimension_Key>" + $0.Key + "</mdex-dimension_Key>";

// Make sure to lower case the booleans in the CSV file
maBool = lowerCase($0.Refinement);
maRecord = maRecord + "<mdex-dimension_EnableRefinements>" + maBool + "<
/mdex-dimension_EnableRefinements>";

maBool = lowerCase($0.DimHierarchy);
maRecord = maRecord + "<mdex-dimension_IsDimensionSearchHierarchical>" + maBool + "<
/mdex-dimension_IsDimensionSearchHierarchical>";

maBool = lowerCase($0.RecHierarchy);
maRecord = maRecord + "<mdex-dimension_IsRecordSearchHierarchical>" + maBool + "<
/mdex-dimension_IsRecordSearchHierarchical>";

$0.xmlString = maRecord + "</mdex:record>";

// Batch up the web service requests.
$0.singleAggregationKey = aggrKey;
n++;
if (n % 15 == 0) {
    aggrKey++;
}

return ALL;
}

```

First, the code defines two integer variables, the first as a loop counter, the second as the aggregation key (aggrKey).

Next, the `transform()` function builds the DDRs by creating the `<mdex:record>` container, and adding the `<mdex-dimension_EnableRefinements>`, `<mdex-dimension_IsDimensionSearchHierarchical>`, and `<mdex-dimension_IsRecordSearchHierarchical>` nodes. The value of each node is derived from the corresponding value in the input file.

Note that in the example input file, the values are specified in ALL CAPS. Convert these values to lower case before adding them to the XML.

Finally, the records are grouped into batches of 15 for submission to the server. Submitting in batches ensures a continuous stream of processing through all components in the graph, which improves the performance of both Integrator ETL and the Endeca Server.

## Web service request

Use the `updateDimensions` operation of the Configuration Web Service to load the DDR configurations into the Endeca Server. The following code illustrates a typical `updateProperties` operation to load DDRs:

```

<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/3/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:updateDimensions
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    $xmlString
  </config-service:updateDimensions>
</config-service:configTransaction>

```

This example includes two variables:

- OUTER\_TRANSACTION\_ID

This variable specifies the outer transaction ID for the request. The variable and its value are stored in the `workspace.prm` file of the Integrator ETL project.

- `$xmlString`

This variable contains the PDRs that have been constructed by the **Reformat** component in the graph.

This code is added to the **Web Services Client** component in the configuration graph.

## Loading and modifying managed attribute values (MVals)

If you load managed attribute values for a managed attribute that has not already been created and configured in the data domain, the managed attribute is created with system defaults for the DDR, as well as for the PDR if it does not already exist.

Therefore, if the default managed attribute and standard attribute schemas (DDR and PDR configurations) meet your needs, you can load managed attribute values without loading managed attribute and standard attribute schemas (configuring the DDR and PDR). If you need to customize the managed attribute or standard attribute schema (configure the DDR or the PDR), or both, you should load the schemas (PDR and DDR configurations) before loading managed attribute values.

Use the **Merge Managed Values** component to add managed values to a managed attribute, or to modify existing managed attribute values. An instance of the **Merge Managed Values** component can only add managed values to, or modify the values of, one managed attribute. Adding values to multiple managed attributes requires multiple instances of the component, either in the same graph or in different graphs.

You can also modify the following properties of managed attributes that have already been loaded:

- Display name
- Rank

Other properties (parent, spec, and synonym) cannot be modified once the managed attribute value has been loaded.

### Managed attribute value input file

The input schema of the managed attribute input file is predefined. The first row is the header row and must define the following properties:

```
spec | displayname | parent | synonym | rank
```

where:

- *spec* is a unique string identifier for the managed value. This ID is used to associate child values with parent values.

This field is required.

- *displayname* is the name displayed in the user interface for the managed value.

This field is optional.

- *parent* is the parent ID for this managed value. If the value is a root value (in other words, if the value has no parent, but only children), enter a forward slash (/) for this property. If the value is a child managed value, enter the unique ID of the parent managed value.



This field is required when load managed attribute values. It is optional when updating managed attribute values, and should be null. If the value of this property is not null when updating managed attribute values, the graph will fail and an error will be returned.

- *synonym* optionally defines one or more synonyms for the managed value. When you add a synonym to a managed attribute value, the value will be returned when users search on the synonym. You can add synonyms to both root and child managed values. You can add multiple synonyms to a single managed value. Specify the delimiter for multiple synonyms on the **Edit Add Managed Values** dialog.

This field is optional.

- *rank* is an integer that specifies the rank order of the value in the set of values for the managed attribute.

This field is optional.

The following graphic illustrates a simple managed attribute input file:

	A	B	C	D	E
1	CategoryKey	CategoryDisplayName	ParentKey	Synonym	Rank
2	CAT_BIKES	Bikes	/		10
3	CAT_COMPONENTS	Components	/		20
4	CAT_CLOTHING	Clothing	/		30
5	CAT_ACCESSORIES	Accessories	/		40
6		1 Mountain Bikes	CAT_BIKES		11
7		2 Road Bikes	CAT_BIKES		12
8		3 Touring Bikes	CAT_BIKES		13
9		4 Handlebars	CAT_COMPONENTS		21
10		5 Bottom Brackets	CAT_COMPONENTS		22
11		6 Brakes	CAT_COMPONENTS		23
12		7 Chains	CAT_COMPONENTS		24

This example defines several parent categories, including CAT\_BIKES and CAT\_COMPONENTS. Each of these categories is ranked.

The following values are defined as children of the CAT\_BIKES value (with rankings):

- Mountain Bikes (Rank 11)
- Road Bikes (Rank 12)
- Touring Bikes (Rank 13)

These values are defined as children of the CAT\_COMPONENTS value (with rankings):

- Handlebars (Rank 21)
- Bottom Brackets (Rank 22)
- Brakes (Rank 23)
- Chains (Rank 24)

All values of the same managed attribute share the same ranking structure. You cannot duplicate ranking values. Consider this requirement when defining the rankings for your managed attribute values and develop a ranking practice that ensures all ranks will be unique. For example, you might use a four-digit ranking

structure, where the place of the digit indicates the level of the ranking of the managed attribute value within the tree of available values. For example:

- 0002  
A first-level managed attribute value (is a child of /), such as the CAT\_COMPONENTS managed attribute value in the example input file.
- 0021  
A second-level managed attribute value (parent is a child /), such as the Road Bikes managed attribute value in the example input file.
- 0211  
A third-level managed attribute value (three levels down from /).
- 3211  
A fourth-level managed attribute value (four levels down from /).

You could modify this structure to meet your needs, or choose to use a different ranking approach.

The input file is typically stored in the `data-in` directory of the project.

While the *display name*, *synonym*, and *rank* fields are optional, the fields must be included in the input records. For example:

```
CAT_CLOTHING| | | |
```

## Modifying managed attribute values

You can modify the *display name* and *rank* of existing managed attribute values. You cannot modify the *spec*, *parent*, or *synonym* of existing managed attribute values.

- If you attempt to change the *spec* of a managed attribute value, a new managed attribute value will be created with the properties you specify.
- If you attempt to change the *parent* of a managed attribute value, Endeca Server returns the following error, and the graph fails: `Error applying updates: Attempting to change hierarchy of existing managed attribute value "1" with assignment of "CAT_BIKES_FOO" to attribute "mdex-dimension_category_Parent"`
- If you attempt to modify the *synonym* of a managed attribute value, Endeca Server returns the following error and the graph fails: `Error applying updates: Attempting to add synonym to managed attribute value "1" with assignment of "Mtn Bike, Mountain Bicycle" to attribute "mdex-dimension-value_Synonyms"`

When modifying managed attribute values, the input file must include values for the *spec* of the managed attribute values you want to modify and the *name* or *rank*, or both, depending on which property you want to modify. While you can omit values for the *parent* and *synonym*, as well as the *name* or *rank* if you are not modifying them, each record must include these fields. The following code illustrates a simple example:

```
spec,name,parent,synonym,rank
1,Mountain Bicycles,,15
```

If this input was submitted after loading the managed attribute values illustrated elsewhere, the managed attribute named "Mountain Bikes" (spec 1) would be renamed to "Mountain Bicycles", and its rank would be updated to 15. Note that the *parent* and *synonym* fields are empty in this input record.

## Managed attribute value input graph

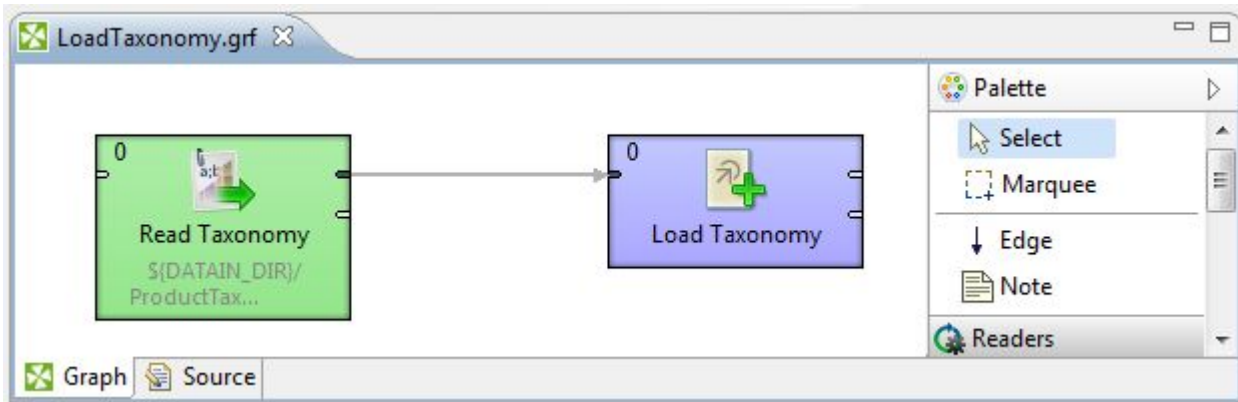
A basic managed attribute value input graph consists of two components:

- A **Universal Data Reader**

Specify the location of the input file (typically the data-in directory of the project) in the **File URL** field.

- A **Merge Managed Values** component

The two components are joined by a basic edge. See [Connecting two components with an edge on page 26](#). The following graphic illustrates a typical configuration:



## Configuring a Merge Managed Values component to load or modify managed attribute values

This topic describes how to configure a **Merge Managed Values** component in a graph to load or modify managed attribute values (mvals).

To configure a **Merge Managed Values** component:

1. In the **Palette**, open the **Discovery** section and drag a **Merge Managed Values** component into the graph.
2. Double-click on the **Merge Managed Values** component.  
Integrator ETL displays the **Edit Merge Managed Values** dialog.
3. The values of the **Endeca Server Host**, **Endeca Server Port**, and **Endeca Server Context Root** fields default to `localhost`, `7001`, and `/endeca-server` respectively. You can change these values.
4. Enter the **Data Domain Name** of the Endeca data domain to which you want to add mvals or that contains the mvals you want to modify.
5. Enter the **Managed Attribute Name** for the managed attribute whose managed attribute values you want to load or modify. You can only enter one name. If you enter more than one name, the graph fails with an error.
6. If your Endeca Server is configured to use SSL, check the **SSL Enabled** box.
7. If your input file includes multiple synonyms for any value, enter the **Synonym Delimiter**.
8. Click **OK** to save the configured component.

## Configuring record search

This topic describes how to build a graph to configure record searches.

Record search configuration controls search interfaces for groups of attributes. Features that can be included and configured in a search interface include:

- Relevance ranking
- Matching across multiple Endeca attributes
- Partial matching
- Enabling snipping for attributes

Record search is controlled by the `recsearch_config` document, which is empty by default:

```
<RECSEARCH_CONFIG/>
```

Use the `REL_RANK_STRATEGY` attribute to specify a relevance ranking strategy for the results of the record search. If you want to specify a relevance ranking strategy, you must load and configure it before configuring record search.

### Input document

The following code illustrates an example `recsearch_config` document:

```
<RECSEARCH_CONFIG>
<SEARCH_INTERFACE DEFAULT_REL_RANK_STRATEGY="All" NAME="Surveys">
  <MEMBER_NAME RELEVANCE_RANK="1">SurveyResponse</MEMBER_NAME>
</SEARCH_INTERFACE>
<SEARCH_INTERFACE DEFAULT_REL_RANK_STRATEGY="ProductRelRank" NAME="Resellers">
  <MEMBER_NAME RELEVANCE_RANK="1">DimReseller_BusinessType</MEMBER_NAME>
  <MEMBER_NAME RELEVANCE_RANK="2">DimReseller_ResellerName</MEMBER_NAME>
</SEARCH_INTERFACE>
<SEARCH_INTERFACE DEFAULT_REL_RANK_STRATEGY="All" NAME="Employees">
  <MEMBER_NAME RELEVANCE_RANK="1">DimEmployee_FullName</MEMBER_NAME>
  <MEMBER_NAME RELEVANCE_RANK="2">DimEmployee_LastName</MEMBER_NAME>
  <MEMBER_NAME RELEVANCE_RANK="3">DimEmployee_FirstName</MEMBER_NAME>
  <MEMBER_NAME RELEVANCE_RANK="4">DimEmployee_Title</MEMBER_NAME>
</SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

### Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the record search configuration document. The code entered in the **Web Services Client** resembles the following example:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
<config-service:OuterTransactionId=${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
<config-service:putConfigDocuments
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="recsearch_config">
<RECSEARCH_CONFIG>
  $xmlString
</RECSEARCH_CONFIG>
</mdex:configDocument>
</config-service:putConfigDocuments>
</config-service:configTransaction>
```

The value of the `name` attribute (`recsearch_config`) of the `configDocument` element directs the service to update the `recsearch_config` document in the server with the value of the `$xmlString` variable in the `RECSEARCH_CONFIG` node.

## Configuring value search

This topic describes how to create a graph that loads the configuration for value search.

Value search is configured by the `dimsearch_config` XML document. The default document includes an empty configuration:

```
<DIMSEARCH_CONFIG/>
```

Use the `RELRANK_STRATEGY` attribute of the `DIMSEARCH_CONFIG` element to specify a relevance ranking strategy to use on the results.



**Note:** Only specify a relevance ranking strategy in value search configuration if you have already configured that relevance ranking strategy. See [Configuring relevance ranking on page 78](#). If you specify a relevance ranking strategy that does not exist, the graph will fail with the message "Invalid Relevance ranking strategy".

For additional details about configuring value searches, see the *Oracle Endeca Server Developer's Guide*.

Use an XML-based configuration graph to configure relevance ranking. See [XML-based configuration graphs on page 84](#). Note that since the input document consists of a single node, you do not need to include a **Denormalizer** to reformat the input file as a single XML string.

### Input file

To configure value search, you need to create an input file similar to this example:

```
<DIMSEARCH_CONFIG FILTER_FOR_ANCESTORS="FALSE" RELRANK_STRATEGY="ProductRelRank" />
```

In this case, the `ProductRelRank` relevance ranking strategy must be defined before submitting this configuration or the configuration will fail.

### Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the value search configuration document. The code entered in the **Web Services Client** resembles the following example:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:putConfigDocuments
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    <mdex:configDocument name="dimsearch_config">
    <DIMSEARCH_CONFIG>
      $xmlString
    </DIMSEARCH_CONFIG>
    </mdex:configDocument>
  </config-service:putConfigDocuments>
</config-service:configTransaction>
```

The value of the `name` attribute (`dimsearch_config`) of the `configDocument` element directs the service to update the `dimsearch_config` document in the server with the value of the `$xmlString` variable in the `DIMSEARCH_CONFIG` node.

## Configuring relevance ranking

This topic explains how to build a graph to configure relevance ranking.

Relevance ranking controls the order of results returned in response to a record search. A relevance ranking strategy is defined in a RELRANK\_STRATEGY element, which contains elements specifying the individual relevance ranking modules. The following modules are available:

Use an XML-based configuration graph to configure relevance ranking. See [XML-based configuration graphs on page 84](#).

### Input document

The default `relrank_strategies` document does not define any relevance ranking strategies:

```
<RELRANK_STRATEGIES/>
```

The following example file creates a relevance ranking strategy named "ProductRelRank", which consists of the Interpreted (REL\_RANK\_INTERP) and Field (REL\_RANK\_FIELD) relevance ranking modules:

```
<RELRANK_STRATEGIES>
  <RELRANK_STRATEGY NAME="ProductRelRank">
    <REL_RANK_INTERP/>
    <REL_RANK_FIELD/>
  </REL_RANK_STRATEGY>
</RELRANK_STRATEGIES>
```

### Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the relevance ranking configuration document. The code entered in the **Web Services Client** resembles the following example:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:putConfigDocuments
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <mdex:configDocument name="relrank_strategies">
  <RELRANK_STRATEGIES>
    $xmlString
  </RELRANK_STRATEGIES>
  </mdex:configDocument>
  </config-service:putConfigDocuments>
</config-service:configTransaction>
```

The value of the `name` attribute (`relrank_strategies`) of the `configDocument` element directs the service to update the `relrank_strategies` document in the server with the value of the `$xmlString` variable in the RELRANK\_STRATEGIES node.

## Configuring stop words

This topic describes how to create a graph that loads stop words.

Stop words are words that are removed from a query before it is processed by the Dgraph.

The default `stop_words` document does not define any stop words:

```
<STOP_WORDS/>
```

Use an XML-based configuration graph to configure stop words. See [XML-based configuration graphs on page 84](#).

## Input file

The following example illustrates a typical example stop words input document:

```
<STOP_WORDS>
  <STOP_WORD>bike</STOP_WORD>
  <STOP_WORD>component</STOP_WORD>
  <STOP_WORD>an</STOP_WORD>
  <STOP_WORD>of</STOP_WORD>
  <STOP_WORD>the</STOP_WORD>
</STOP_WORDS>
```

## Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the stop words configuration document. The code entered in the **Web Services Client** resembles the following example:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:putConfigDocuments
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <mdex:configDocument name="stop_words">
  <STOP_WORDS>
    $xmlString
  </STOP_WORDS>
  </mdex:configDocument>
  </config-service:putConfigDocuments>
</config-service:configTransaction>
```

The value of the `name` attribute (`stop_words`) of the `configDocument` element directs the service to update the `stop_words` document in the server with the value of the `$xmlString` variable in the `STOP_WORDS` node.

## Configuring the thesaurus

This topic describes how to create a graph that updates the thesaurus on the server.

The thesaurus allows the system to return matches for concepts related to the words or phrases included in user queries. For example, if you have records that include "Italy", you might want also want to return those records when a user query includes "Italian".

The default thesaurus document does not define any thesaurus entries:

```
<THESAURUS/>
```

Use an XML-based configuration graph to configure the thesaurus. See [XML-based configuration graphs on page 84](#).

## Input file

The following example input file sets two thesaurus entries:

```
<THESAURUS>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>italy</THESAURUS_FORM>
```

```

    <THESAURUS_FORM>italian</THESAURUS_FORM>
  </THESAURUS_ENTRY>
</THESAURUS_ENTRY>
  <THESAURUS_FORM>france</THESAURUS_FORM>
  <THESAURUS_FORM>french</THESAURUS_FORM>
</THESAURUS_ENTRY>
</THESAURUS>

```

## Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the thesaurus configuration document. The code entered in the **Web Services Client** resembles the following example:

```

<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
  <config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
  <config-service:putConfigDocuments
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <mdex:configDocument name="thesaurus">
  <THESAURUS>
    $xmlString
  </THESAURUS>
  </mdex:configDocument>
  </config-service:putConfigDocuments>
</config-service:configTransaction>

```

The value of the `name` attribute (*thesaurus*) of the `configDocument` element directs the service to update the thesaurus document in the server with the value of the `$xmlString` variable in the `THESAURUS` node.

## Configuring precedence rules

A precedence rule suppresses refinements of an Endeca attribute until a condition is met. Suppressing these refinements simplifies navigation through the data and helps avoid information overload problems.

A precedence rule delays the display of standard or managed attributes. In other words, the attribute is hidden until a change in condition triggers its display. As a result, it is easier to navigate through the data and users avoid information overload.

For example, a set of records includes City and State attributes. The application is easier to use if the City attribute is hidden until the user has specified a State. Otherwise, multiple Cities with the same name would be presented, and the user would have difficulty selecting the correct one. For example, choosing "Portland" would return records for both Portland, Maine, and Portland, Oregon. To suppress Cities until a State is selected, create a precedence rule with State as the trigger and City as the target.

You can load precedence rules before loading the standard and managed attributes. The attributes specified in a precedence rule do not have to exist in the data domain when you configure the precedence rule. The Endeca Server does not perform any error checking to ensure that the attributes exist. For this reason, you must ensure that attribute names are spelled correctly in the precedence rule input file.

Moreover, if the trigger attribute specified in a precedence rule does not exist in the data domain, but the target attribute does exist, the precedence rule will never be triggered. This behavior effectively hides the target attribute from refinements. To correct this behavior, either remove the rule or create the trigger attribute in the data domain.

[Precedence rule schema](#)

[Format of the precedence rules input file](#)



## Building a precedence rules graph

### Precedence rule schema

Each precedence rule is represented as a single record in the data domain.

The `config-service:putPrecedenceRules` operation processes precedence rules, adding new rules and updating existing rules. Each `precedenceRule` element uses the following schema syntax:

```
<mdex:precedenceRule
  key="ruleName"
  triggerAttributeKey="triggerAttrName"
  triggerAttributeValue="mval|sval"
  targetAttributeKey="targetAttrName"
  isLeafTrigger="true|false"/>
```

The following table describes the meaning of the `precedenceRule` attributes:

Table 4.1: Meaning of precedenceRule attribute schema

precedenceRule attribute	Meaning
key	Specifies a unique identifier for the precedence rule (that is, it is the name of the rule). The identifier is a string, which does not have to follow the NCName format.
triggerAttributeKey	Specifies the name of the Endeca standard attribute or managed attribute that will trigger the precedence rule. That is, the specified attribute must be selected before the user can see the target attribute.
triggerAttributeValue	Optional. If used, specifies the attribute value (either managed value spec or standard attribute value) that must be selected before the user can see the target attribute. If not used, then any value in the trigger attribute will trigger the rule. Use of <code>triggerAttributeValue</code> in effect further refines the trigger to a specific standard or managed value.
targetAttributeKey	Specifies the name of the Endeca standard or managed attribute that appears after the trigger attribute value is selected.

precedenceRule attribute	Meaning
isLeafTrigger	<p>If the trigger is a managed attribute, <code>isLeafTrigger</code> specifies a Boolean value (that must be in lower case) that denotes the type of the trigger attribute value:</p> <ul style="list-style-type: none"> <li>• If <code>true</code>, the trigger attribute is a leaf type, which means that the precedence rule will fire only if a leaf value is selected. That is, querying any leaf managed value from the trigger managed attribute will cause the target managed value to be displayed (many triggers, one target).</li> <li>• If <code>false</code> (the default), the trigger attribute is a non-leaf type, which means that the precedence rule will fire when any value is selected. That is, if the managed value specified as the trigger or any of its descendants are in the navigation state, then the target is presented (one trigger, one target).</li> </ul> <p>Note that <code>isLeafTrigger</code> does not apply to Endeca standard attributes. You must specify it when you create a precedence rule, but whichever value you use is ignored by the Dgraph when the precedence rule is run.</p>

## Precedence rule example

The following is an example of a `config-service:putPrecedenceRules` operation that creates a precedence rule named `ProvinceRule`:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <config-service:putPrecedenceRules>
    <mdex:precedenceRule
      key="ProvinceRule"
      triggerAttributeKey="DimGeography_StateProvinceName"
      triggerAttributeValue="Queensland"
      targetAttributeKey="DimGeography_City"
      isLeafTrigger="true"/>
    </config-service:putPrecedenceRules>
  </config-service:configTransaction>
```

Note that this example does not use the optional `OuterTransactionId` element for the operation. This operation can be placed in a request structure of a **WebServiceClient** component.

## Format of the precedence rules input file

The input configuration file should contain five configuration properties and a corresponding set of value data.

The first line (the header row) of a precedence rules input file should have these header properties:

```
Key|TriggerAttribute|TriggerValue|TargetAttribute|isLeafTrigger
```

The names of the header properties are arbitrary. For example, you can use `RuleName` instead of `Key`). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).

The header properties map to the `precedenceRule` attributes as follows:

Table 4.2: Precedence rule attributes

Input Header Property	Maps to precedenceRule attribute	Description
Key	key	Name of the precedence rule.
TriggerAttribute	triggerAttributeKey	Name of the standard or managed attribute trigger.
TriggerValue	triggerAttributeValue	Standard or managed attribute value for the trigger. Optional, so the value in the input file can be blank.
TargetAttribute	targetAttributeKey	Name of the standard or managed attribute target.
isLeafTrigger	isLeafTrigger	For managed attributes, specifies if the trigger attribute is a leaf.

After the header row, the second and following rows in the input file contain configuration data for the precedence rules. The following image illustrates an example CSV configuration file for two precedence rules:

	A	B	C	D	E
1	Key	TriggerAttribute	TriggerValue	TargetAttribute	isLeafTrigger
2	AUS_Rule	DimGeography_CountryRegionName	Australia	DimGeography_StateProvinceName	FALSE
3	City_Rule	DimGeography_StateProvinceName		DimGeography_City	FALSE
4					
5					

Note that the `TriggerValue` for the second precedence rule is blank, which means that any value in the `DimGeography_StateProvinceName` attribute will trigger the rule.

## Building a precedence rules graph

To load precedence rules, use an XML-based configuration graph.

See [XML-based configuration graphs on page 84](#) for details about building a precedence rules graph.

## Web service request

Use the `putPrecedenceRules` operation of the Configuration Web Service to load precedence rules. The following code illustrates an typical example of the web services request to load precedence rules:

```
<config-service:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <config-service:putPrecedenceRules>
    <mdex:precedenceRule
      key="ProvinceRule"
      triggerAttributeKey="DimGeography_StateProvinceName"
      triggerAttributeValue="Queensland"
      targetAttributeKey="DimGeography_City"
      isLeafTrigger="true"/>
    </config-service:putPrecedenceRules>
  </config-service:configTransaction>
```

This code is added to the **Web Services Client** component in the configuration graph.

## XML-based configuration graphs

XML-based configuration graphs use an XML file as the input to the graph.

XML-based configuration graphs consist of three components:

- **Universal Data Reader**

This component reads the data into the graph. The URL field of the component specifies the location of the XML file you want to load. XML input files, like other input files, are typically stored in the project's **data-in** directory.

- **Denormalizer**

This component converts the input XML file into a single XML string. The Endeca Server's Web services require the configuration data to be input as a single string, so all carriage returns and tabs typically used to format an XML file must be removed before the file is added to the Web services request. See [Generating a single XML string on page 84](#) for details about the implementation of this component.

- **Web Services Client**

- This component submits the request, with the configuration document embedded, to the Endeca Server for processing.

Use the following edges between the components:

- Use a basic edge to link the **Universal Data Reader** to the **Denormalizer**. See [Connecting two components with an edge on page 26](#).
- Use an aggregation edge to link the **Denormalizer** to the **Web Services Client**. For details about implementing an aggregation edge, see [Creating an aggregation edge for configuration loading graphs on page 85](#).

[Generating a single XML string](#)

[Creating an aggregation edge for configuration loading graphs](#)

## Generating a single XML string

Use a **Denormalizer** component to generate a single XML string from an input XML file.

When building a graph to load configuration files, add the following CTL code to the **Denormalizer** to generate a single XML string from an input XML file:

```
integer n = 0;
string value = "";

function integer append() {
    value = value + $0.xmlString + "\n";
    n++;
    return n;
}

// This function is called once after the
// append() function was called for all records
// of a group of input records defined by the key.
// It creates a single output record for the whole group.
function integer transform() {
    $0.xmlString = value;
    value = "";
    return OK;
}
```

This code defines an integer variable for a counter, and sets the initial value to zero. It also defines an empty string variable (string value = "").

The append function cycles through the input XML and builds a single XML string by concatenating the input XML node with the current value of the string variable.

When all nodes have been concatenated to the string variable, the value of the variable is output to the \$0 port of the component.

## Creating an aggregation edge for configuration loading graphs

This task describes how to create an edge to connect a **Denormalizer** in a load configuration graph to other components in the graph.

To configure an aggregation edge:

1. Right-click on the Edge and select **New metadata>User defined**.  
Integrator ETL displays the Metadata editor with one default field.
2. In the **Record:recordName1** field:
  - (a) Change the **recordName1** default value to a descriptive name, such as **DenormEdge**.
  - (b) Leave the **Type** field as **delimited**.
  - (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. Change the **field1** name to **xmlString** and leave its **Type** as **string**.
4. Click the + (plus sign control) to add a new field. Name the field **singleAggregationKey** and set its **Type** as **integer**.
5. Click **Finish**.
6. Save the graph.

## CSV-based configuration graphs

CSV-based configuration graphs use a CSV file as the input to the graph.

CSV-based configuration graphs consist of 4 components:

- **Universal Data Reader**

This component reads the data into the graph. The URL field of the component specifies the location of the XML file you want to load. XML input files, like other input files, are typically stored in the project's **data-in** directory.

- **Transformer**

This component uses the data from the input file to create the XML file the Endeca Server uses to implement the new configuration.

- **Denormalizer**

This component converts the XML file generated by the **Transformer** into a single XML string. The Endeca Server's Web services require the configuration data to be input as a single string, so all carriage returns and tabs typically used to format an XML file must be removed before the file is added to the web services request. See [Generating a single XML string on page 84](#) for details about the implementation of this component.

- **Web Services Client**

- This component submits the request, with the configuration document embedded, to the Endeca Server for processing.

Use the following edges between the components:

- Use a basic edge to link the **Universal Data Reader** to the **Transformer**. See [Connecting two components with an edge on page 26](#).
- Use an aggregation edge to link the **Transformer** to the **Denormalizer** and the **Denormalizer** to the **Web Services Client**. For details about implementing an aggregation edge, see [Creating an aggregation edge for configuration loading graphs on page 85](#).

## Exporting and importing data domain configurations

This section describes how to export and import data domain configurations.

You may want to export a data domain configuration to be imported later for a number of reasons.

- During the development process, you may want to preserve the configuration developed at each stage as a starting point for the next stage.
- Once a production environment is in place, recommended practice is to keep a development environment synchronized with the same configuration.
- You may occasionally decide to delete a database and reload it.

Exporting a configuration exports the following data:

- The records schema; in other words, the Property Description Records (PDRs) and Dimension Description Records (DDR) that describe the attributes, their data types, behavior, and hierarchy.

- XML configuration documents, such as the record search configuration, search interfaces, and thesaurus configuration.
- The GCR, precedence rules, and similar configuration data.

When you import a configuration, the data domain is updated with all of these configurations. Note that managed attribute values (mvals) are not exported and thus cannot be imported. You must load mvals separately. See [Loading managed attribute values on page 72](#) for details.

Use the **Export Config** component to export a data domain configuration; use the **Import Config** component to import a data domain configuration.

[Building an export graph](#)

[Building an import graph](#)

[Configuring edges for export and import graphs](#)

## Building an export graph

This topic describes how to create a graph to export a data domain configuration.

An Export graph consists of two components:

- An **Export Config** component to export the configuration from the Endeca Server. When configuring this component:
  - In the **Endeca Server Host** field, specify the name or IP address of the Endeca server. The value defaults to `localhost`. You can also use the `#{ENDECA_SERVER_HOST}` global variable if the name or IP address of the host is specified in the `workspace.prm` file for the project.
  - In the **Endeca Server Port** field, specify the port on which the Endeca server listens. The value defaults to `7770`. You can also use the `#{ENDECA_SERVER_PORT}` global variable if the Endeca server port is specified in the `workspace.prm` file for the project.
  - In the **Data Domain Name** field, specify the data domain whose configuration you want to export.
  - Only check the **SSL Enabled** box if SSL is enabled on the Endeca Server.
- A writer component, typically a **Universal Data Writer**, to write the exported configuration to a file. When using the **Universal Data Writer** component, the location and name of the export file are specified in the **File URL** field on the component editor.
- The two components are joined with a specially-configured edge. See [Configuring edges for export and import graphs on page 88](#) for details about creating an export/import edge.

## Building an import graph

This topic describes how to create a graph to import a configuration to a data domain.

An import graph consists of two components:

- A reader component, typically a **Universal Data Reader**, to read the configuration file to import. When using the **Universal Data Reader**, the location and name of the import file are specified in the **File URL** field on the component editor.

- An **Import Config** component to import the new configuration into the data domain. When configuring this component:
  - In the **Endeca Server Host** field, specify the name or IP address of the Endeca Server. The value defaults to `localhost`. You can use the `#{ENDECA_SERVER_HOST}` global variable if the name or IP address of the host is specified in the `workspace.prm` file for the project.
  - In the **Endeca Server Port** field, specify the port on which the Endeca server listens. The value defaults to `7770`. You can also use the `#{ENDECA_SERVER_PORT}` global variable if the Endeca server port is specified in the `workspace.prm` file for the project.
  - In the **Data Domain Name** field, specify the data domain whose configuration you want to import.
  - Only check the **SSL Enabled** box if SSL is enabled on the Endeca Server.
- The two components are joined with a specially-configured edge. See [Configuring edges for export and import graphs on page 88](#) for details about creating an export/import edge.

## Configuring edges for export and import graphs

This topic describes how to configure the metadata of edges in export and import graphs.

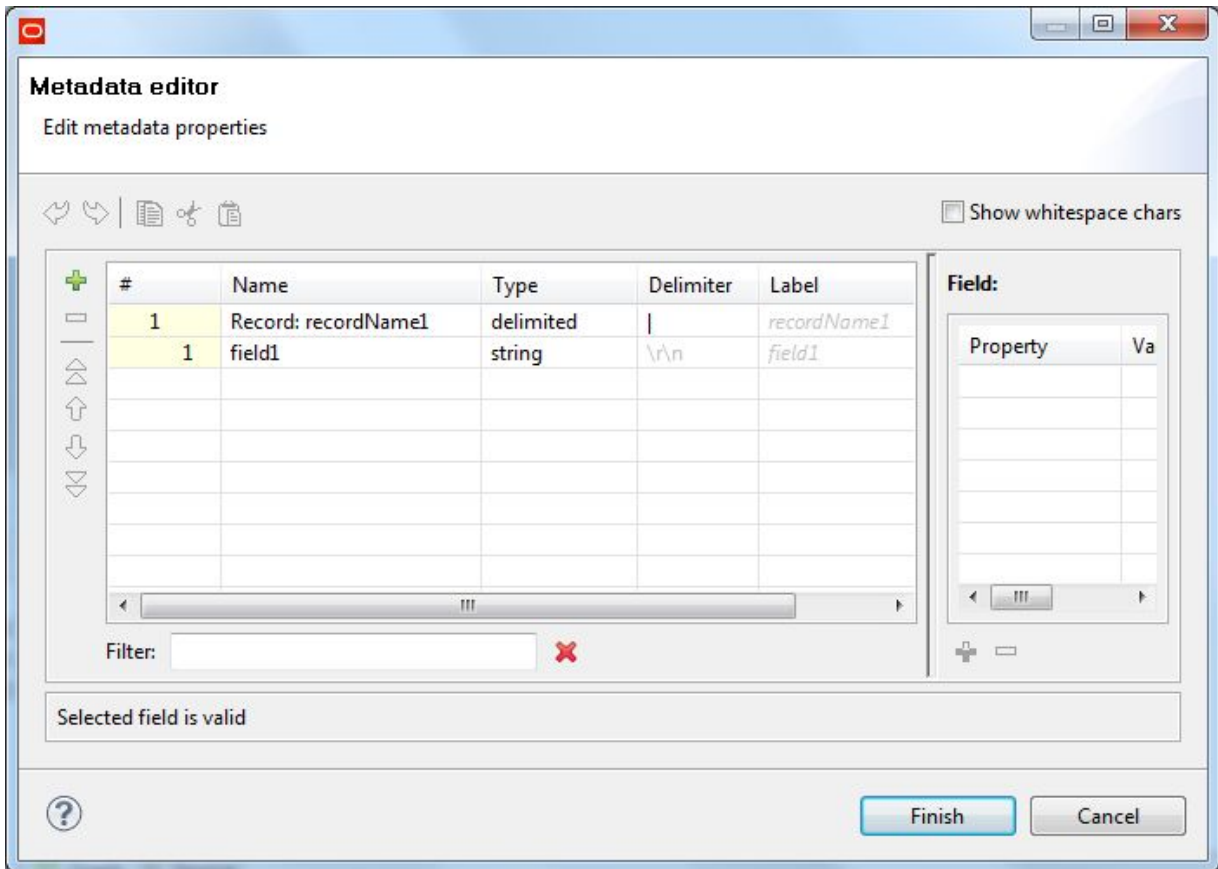
The metadata applied to edges in export and import graphs must be configured to have only one string field and no record delimiter. Therefore, the metadata of the edge must be manually modified to remove the record and field delimiters from the metadata. This modification leaves the **EOF as delimiter** property as the sole delimiter.



To configure the edge metadata definition for exporting the configuration from an Endeca data domain:

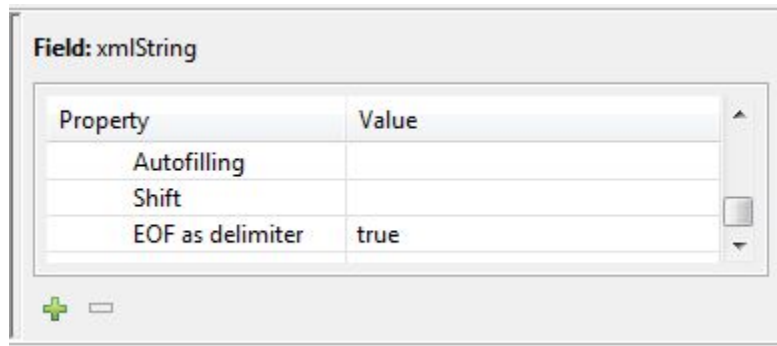
1. Right-click on the edge and select **New metadata>User defined**.

Integrator displays the Metadata editor with one default field.



2. In the **Record:recordName1** field:
  - (a) Change the **recordName1** default value to a more descriptive name (such as Export).
  - (b) Leave the **Type** field as **delimited**.
  - (c) Leave the **Delimiter** field as is for now. (You will delete it in a later step.)
3. In the Record pane, make the following changes to the **field1** property:
  - (a) Change the **field1** default name to **xmlString**.
  - (b) Leave the **Type** field set to **String**.

- (c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as illustrated in the following graphic:



4. Click **Finish**.
5. Next, you must manually remove the record and field delimiters from the metadata:
  - (a) In the Graph Editor, click the **Source** tab (which is next to the **Graph** tab).
  - (b) Find the Metadata element with the id that matches the metadata you just applied to the edge. It is most likely that the id is "0". In the Record child element, find the **fieldDelimiter** and **recordDelimiter** attributes, as shown in the following example code:

```
<Metadata id="Metadata0">
  <Record fieldDelimiter="|" name="Export" recordDelimiter="\r\n" type="delimited">
    <Field eofAsDelimiter="true" name="xmlString" type="string"/>
  </Record>
</Metadata>
```

- (c) Delete the **fieldDelimiter** and **recordDelimiter** attributes. As a result, the record should resemble the following example code:

```
<Metadata id="Metadata0">
  <Record name="Export" type="delimited">
    <Field eofAsDelimiter="true" name="xmlString" type="string"/>
  </Record>
</Metadata>
```

- (d) While still in the Source view, right-click and from the popup menu choose **Save** to save the graph.
6. Click the **Graph** icon to return to the Graph Editor.



## Chapter 5

# Managing View Definitions

---

This section describes how to export and restore view definitions.

[About view definitions](#)

[Exporting view definitions](#)

[Importing view definitions](#)

## About view definitions

Use Integrator ETL to export and restore view definitions.

Use the **Views** page in Studio to create and modify view definitions. The **Views** page provides a rich graphical user interface that simplifies the process of creating and managing views. For details, see "Defining Views of Application Data" in the *Oracle Endeca Information Discovery Studio User's Guide*.

You may occasionally need to clear, reconfigure, and reload a data domain. When you do so, you may want to export your view definitions so you restore them rather than manually recreating them.

## Exporting view definitions

This topic describes how to create a graph to export view definitions.

Exporting a view definition uses the `listEntities` operation of the Entity and Collection Configuration Web Service (sconfig) in the Endeca Server. You can build a simple graph run this operation and create the export file.

To export view definitions:

1. Create an input XML file with the `listEntities` operation. Copy the following XML code.



**Note:** The following code is formatted for presentation. In your input file, you should remove any line breaks.

The input file should be saved in the **data\_in** directory of the project whose view definitions you want to export.



**Note:** In the examples in the following steps, the input file is assumed to be named `soap.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://
/www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <listEntities xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
```

```
<outerTransactionId />
</listEntities>
</soap:Body>
</soap:Envelope>
```

2. Create a new graph in the project whose view definitions you want to export.
3. Add the following components to the graph:
  - **UniversalDataReader**
  - **HTTPConnector**
  - **UniversalDataWriter**
4. In the **UniversalDataReader**, configure the File URL property with the path and name of the input file. You can use the `${DATAIN_DIR}` global variable. For example: `${DATAIN_DIR}/soap.xml`.
5. Configure the **HTTPConnector** component with the following values:

Property	Value
URL	<code>http://\${ENDECA_SERVER_HOST}:\${ENDECA_SERVER_PORT}/endeca-server/ws/sconfig/\${DATA_DOMAIN_NAME}</code> where <ul style="list-style-type: none"> <li>• <code>\${ENDECA_SERVER_HOST}</code> is the name or IP address of the Endeca server machine. You can also use the <code>\${ENDECA_SERVER_HOST}</code> global variable for host portion of the URL.</li> <li>• <code>\${ENDECA_SERVER_PORT}</code> is the port of the Endeca server. You can also use the <code>\${ENDECA_SERVER_PORT}</code> global variable for this portion of the URL.</li> <li>• <code>\${DATA_DOMAIN_NAME}</code> is the name of the data domain whose views you want to export. You can also use the <code>\${DATA_DOMAIN_NAME}</code> global variable.</li> </ul>
Request Method	POST
Input Field	XML
Output Field	XML

6. In the **UniversalDataWriter** component, configure the File URL property with the path and name of the output file. You can use the `${DATAOUT_DIR}` global variable. For example: `${DATAOUT_DIR}/view_config.xml`.
7. Add edges to connect the **UniversalDataReader** to the **HTTPConnector**, and the **HTTPConnector** to the **UniversalDataWriter**.

8. Add metadata to one of the edges:
  - (a) Right-click on the edge and from the popup menu choose **Edit**. On the next popup menu, choose **Create Metadata**.  
Integrator ETL displays the Metadata editor.
  - (b) In the first field (*Record*), change the default **Name** (*recordName1*) to *XML*. Leave the **Type** as *delimited*. Delete the value in the **Delimiter** column. (This column should be null for this record.)
  - (c) In the following field, change the default **Name** (*field1*) to *XML*. Change the **Type** to *string*. Delete the value in the **Delimiter** column. (This column should be null for this record.)
  - (d) In the Field Properties for the XML field, set the value of the **EOF as delimiter** property to *true*.
  - (e) Save the metadata.
9. Add the metadata you just created to the other edge.
10. Save the graph.
11. Run the graph.

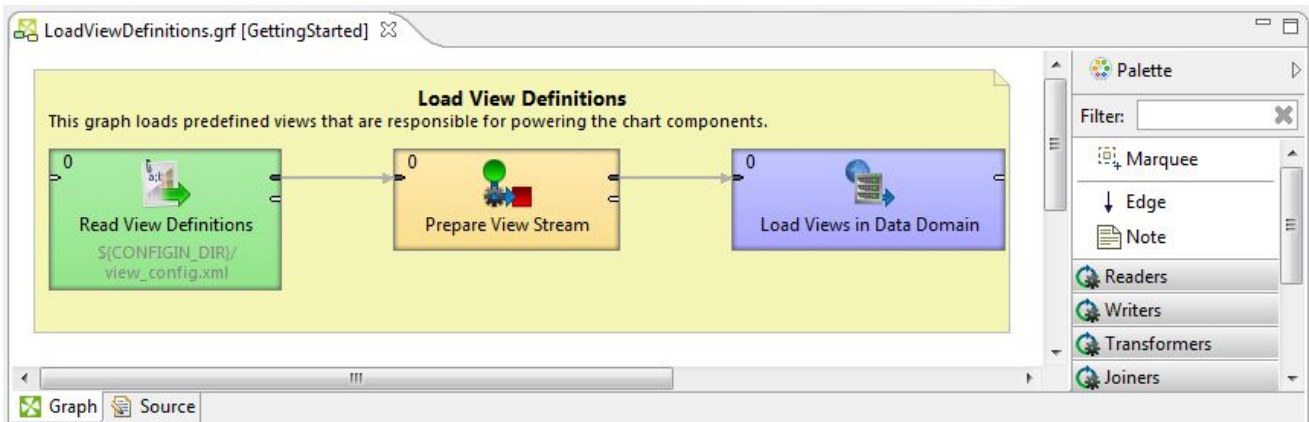
When you run the graph, an XML file with the name you specified is created in the `data-in` directory of the project. See the `view_config.xml` file in the Getting Started project's `data-in` directory for an example.

## Importing view definitions

This topic describes how to build a graph to import view definitions.

You can only import view definitions if they have already been exported.

The example graph used in this topic is from the Getting Started application. This graph is named `LoadViewDefinitions` and looks like this:



You can easily recreate the graph by using the following XML code from the `LoadViewDefinitions` graph:

```
<?xml version="1.0" encoding="UTF-8"?>
<Graph author="ej" created="Thu Jan 06 11:23:35 EDT 2016" guiVersion="4.1.0" id
="1317915999906" licenseCode="CLP1DENDEC48373255BY" licenseType="Commercial" modified
="Thu Jan 21 12:05:15 EST 2016" modifiedBy="fcal" name="LoadViewDefinitions" revision
="1.545" showComponentDetails="true">
<Global>
<Metadata id="Metadata0" previewAttachmentCharset="ISO-8859-1">
```

```

<Record fieldDelimiter="|" name="recordName1" previewAttachmentCharset="ISO-8859-1" recordDelimiter
="\\r\\n" type="delimited">
<Field name="Added" type="string"/>
<Field name="Replaced" type="string"/>
</Record>
</Metadata>
<Metadata id="Metadataall" previewAttachmentCharset="ISO-8859-1">
<Record name="viewXmlStream" previewAttachmentCharset="ISO-8859-1" recordSize="-1" type="delimited">
<Field eofAsDelimiter="true" name="xmlString" nullable="true" shift="0" size="0" type="string"/>
</Record>
</Metadata>
<GraphParameters>
<GraphParameterFile fileURL="workspace.prm"/>
</GraphParameters>
<Note alignment="1" backgroundColorB="181" backgroundColorG="245" backgroundColorR="245" enabled
="true" folded="false" height="173" id="Note0" textColorB="0" textColorG="0" textColorR
="0" textFontSize="8" title="Load View Definitions" titleColorB="0" titleColorG="0" titleColorR
="0" titleFontSize="10" width="641" x="14" y="12">
<attr name
="text"><![CDATA[This graph loads predefined views that are responsible for powering the chart
components.]]></attr>
</Note>
<Dictionary/>
</Global>
<Phase number="0">
<Node enabled="enabled" guiName="Load Views in Data Domain" guiX="472" guiY="64" id
="LOAD_VIEWS_IN_DATA_DOMAIN1" operationName="{http://www.endeca.com/endeca-server/sconfig/3}SConfig
#SConfigPort#putEntities" type="WEB_SERVICE_CLIENT" wsdlURL="http:/
/{$ENDECA_SERVER_HOST}/{$ENDECA_SERVER_PORT}/{$ENDECA_SERVER_CONTEXT}/ws/sconfig/{$DATA_DOMAIN_NAME}
?wsdl">
<attr name="namespaceBindings"><![CDATA[0=http\\://www.endeca.com/endeca-server/sconfig/3/0
XMLS=http\\://www.w3.org/2001/XMLSchema
incl=http\\://www.w3.org/2004/08/xop/include
]]></attr>
<attr name="requestStructure"><![CDATA[<ns:putEntities xmlns:ns="http://www.endeca.com/endeca-server
/sconfig/3/0">
$xmlString
</ns:putEntities>]]></attr>
<attr name="responseMapping"><![CDATA[<Mappings>
  <Mapping element="{http://www.endeca.com/endeca-server/sconfig/3/0}putEntitiesResponse">
    <Mapping element="{http://www.endeca.com/endeca-server/sconfig/3
/0}entityAdditionInformation" outPort="0"
      xmlFields="{ }numEntitiesAdded;{ }numEntitiesReplaced"
      cloverFields="Added;Replaced">
    </Mapping>
  </Mapping>
</Mappings>
]]></attr>
</Node>
<Node enabled="enabled" guiName="Prepare View Stream" guiX="250" guiY="64" id
="PREPARE_VIEW_STREAM" type="REFORMAT">
<attr name="transform"><![CDATA[//CTL2

// Transforms input record into output record.
function integer transform() {

    string xmlString
=
replace(replace(replace(replace($in.0.xmlString, 'ns3', 'ns'), 'ns2', 'ns'), 'validatedSemanticEntity', 'se
manticEntity'), 'isValid="true" ', '');

    //remove header
integer totalLength = length(xmlString);
integer viewsIndex = indexOf(xmlString, "<ns:semanticEntity");
xmlString = right(xmlString, totalLength-viewsIndex);

    //remove parsedDefinitions & dependentSources
xmlString = replace(xmlString, '<ns:parsedDefinition>.*</ns:parsedDefinition>', '');

```

```

xmlString = replace(xmlString, '<ns:dependentSources>.*</ns:dependentSources>', '');

//remove footer
totalLength = length(xmlString);
viewsIndex = indexOf(xmlString, "</ns:listEntitiesResponse");
xmlString = left(xmlString, viewsIndex);

$out.0.xmlString = xmlString;

return ALL;
}

// Called during component initialization.
// function boolean init() {}

/
/ Called during each graph run before the transform is executed. May be used to allocate and
initialize resources
// required by the transform. All resources allocated within this method should be released
// by the postExecute() method.
// function void preExecute() {}

// Called only if transform() throws an exception.
// function integer transformOnError(string errorMessage, string stackTrace) {}

/
/ Called during each graph run after the entire transform was executed. Should be used to free any
resources
// allocated within the preExecute() method.
// function void postExecute() {}

// Called to return a user-defined error message when an error occurs.
// function string getMessage() {}
]]></attr>
</Node>
<Node enabled="enabled" fileURL="{CONFIGIN_DIR}/view_config.xml" guiName
="Read View Definitions" guiX="24" guiY="64" id="READ_VIEW_DEFINITIONS" type="DATA_READER"/>
<Edge debugMode="false" fromNode="PREPARE_VIEW_STREAM:0" guiEndpoints="" guiRouter="Manhattan" id
="Edge6" inPort="Port 0 (request)" metadata="Metadataall" outPort="Port 0 (out)" toNode
="LOAD_VIEWS_IN_DATA_DOMAIN1:0"/>
<Edge debugMode="false" fromNode="READ_VIEW_DEFINITIONS:0" guiEndpoints="" guiRouter="Manhattan" id
="Edge0" inPort="Port 0 (in)" metadata="Metadataall" outPort="Port 0 (output)" toNode
="PREPARE_VIEW_STREAM:0"/>
</Phase>
</Graph>

```

To build an import view configuration graph:

1. Create a new graph in the project whose view definitions you want to import.
2. Copy the XML graph code above.
3. In your new graph, open the **Source** tab.
4. Paste the XML code you copied into the **Source** tab.
5. Save the graph.



**Note:** You may also need to create the `ViewXmlStream.fmt` metadata file to add to the edges.



## Chapter 6

# Working with Outer Transaction Graphs

---

This chapter describes how to build outer transaction graphs, which can run multiple sub-graphs in one transaction. It also provides information about committing and rolling back outer transactions.

[About transactions](#)

[Setting up outer transactions](#)

[Creating an outer transaction graph using the Transaction RunGraph component](#)

[Committing or rolling back an outer transaction](#)

[Performance impact of transactions](#)

## About transactions

Requests to load configurations or data to the Endeca Server represent transactions. An outer transaction is a set of operations performed in the Oracle Endeca data domain that are viewed as a single unit.

Integrator ETL components that load data or configurations into an Endeca data domain either make Web service requests or make requests to one of the ingest interfaces (the Data Ingest Web Service [DIWS] or the Bulk Load Interface). Each of these requests represents an individual set of operations and succeeds or fails on its own; in other words, each set of requests is a transaction. Because these transactions do not contain other transactions, they are called *inner transactions*.

If you initiate a series or set of independent inner transactions in the Endeca data domain, some may succeed and others may fail. As a result, the data set in the Endeca data domain may be only partially updated, and the data domain will be in an inconsistent state.

To ensure that the data domain ends in a consistent state, it is better for the complete set of transactions to succeed or to fail as a single unit, so the resulting data set either represents all changes or none of the changes. Moreover, you may also want to ensure that users in Studio do not access intermediate states of the data set, but only access the state of the data set before the update until the update is complete, then transition seamlessly to the updated data set.

To achieve these goals, implement a graph that runs a set of transactions within an outer transaction.

An *outer transaction* is a set of operations or transactions performed in the Endeca data domain that is processed as a single unit. If all the data updates and configuration changes made during the outer transaction complete successfully, the outer transaction as a whole is successful and can be committed. Committing the outer transaction finalizes the changes to the data domain. If any of the operations fail, the outer transaction as a whole fails and the changes can be rolled back, returning the data domain to its state before you initiated any transactions.

Use the **Transaction RunGraph** component to implement an outer transaction. The **Transaction RunGraph** component starts an outer transaction in the Endeca data domain using the Transaction Web Service. You can call other graphs as operations within the outer transaction.



## About the Transaction RunGraph component

### When to use outer transactions

## About the Transaction RunGraph component

The **Transaction RunGraph** component works as follows:

1. The component starts an outer transaction using the Transaction Web Service.
2. It runs a sections of specified sub-graphs within that transaction.
3. When all transactions in the series complete successfully, it commits the outer transaction.

You can also configure the **Transaction RunGraph** component to respond to unsuccessful transactions. Common practice is to roll back failed outer transactions.

## When to use outer transactions

Use outer transactions when you want a set of transactions to succeed or fail as a unit.

You may want to use outer transactions in the following circumstances:

- You want to make a set of changes to the data domain as a unit, to ensure the consistency of the data domain.
- You want to make a set of changes to the data domain, but you want Studio users to continue using the original data set until all changes are complete. Once the changes are complete, you want Studio users to use the updated data set.

Outer transactions are very useful when you want to update the data set in the Endeca data domain. Consider using outer transaction to implement all data updates.

Another common use is running an initial load of the data domain, or a baseline update.

## Setting up outer transactions

To set up outer transactions, define the outer transaction global variable in the project file, and add outer transaction nodes to Web services requests in sub-graphs.

### Adding the outer transaction global variable to the project

In the `workspace.prm` file for your project, add the global variable `OUTER_TRANSACTION_ID`. The variable should be empty (`OUTER_TRANSACTION_ID=`). When you run the outer transaction graph, the **Transaction RunGraph** component sets the value of this variable to *transaction* and passes it to each sub-graph contained in the transaction.

### Adding the outer transaction global variable to web service requests

If you use a **Web Service Client** component to run Endeca Web services that modify the Endeca data domain, you must include an `<OuterTransactionId>` element. The value of this element is the ID of the outer transaction. Standard practice is to use the `OUTER_TRANSACTION_ID` global variable:

`#{OUTER_TRANSACTION_ID}`. Using the global variable ensures that the graph can run both independently and as part of an outer transaction without modifying either the graph components or `workspace.prm`.

The following code illustrates an example Web service call with the `OUTER_TRANSACTION_ID` global variable:

```
<config-service:OuterTransactionId>#{OUTER_TRANSACTION_ID}
</config-service:OuterTransactionId>
```

## Creating an outer transaction graph using the Transaction RunGraph component

This example illustrates a simple implementation of an outer transaction graph.

This example outer transaction graph runs two sub-graphs. The first sub-graph loads a standard attribute schema into an Endeca data domain, the other loads a managed attribute schema into the data domain.

[Format of the steps input file](#)

[Creating an outer transaction graph](#)

### Format of the steps input file

The input file for the **Transaction RunGraph** component specifies the graphs to run within the outer transaction.

The following graphic illustrates a sample input file, `AttributeSteps.csv`:

	A	B	C
1	Path	Arguments	
2	./graph/LoadAttributeSchema.grf		
3	./graph/LoadTaxonomySchema.grf		
4			

The first line is the header row, which defines the names of the properties:

```
Path,Argument
```

The actual names of the properties are arbitrary; you can use different names from those used in this illustration. The properties are delimited (for example, by the comma in the example CSV file). The second and following rows in the input file contain the input values:

- The first column in each row (`Path` in this example) lists the path to a graph to be run by the **Transaction RunGraph** component. The outer transaction runs these sub-graphs in the order they are listed.
- The second column in each row (`Arguments` in this example) specifies any graph command-line arguments for the graph listed in the first column. No arguments are specified in this example input file.

Standard practice is to store the input file in the project's `data-in` directory.

## Creating an outer transaction graph

This topic describes how to create a graph to run outer transactions.

Before creating an outer transaction graph, create the step input file. See [Format of the steps input file on page 98](#).

To create an outer transaction graph:

1. Create a new graph in the project where you want to run graph in an outer transaction.
  2. Add the following components to the graph: **Universal Data Reader** (from the Readers section) and **Transaction RunGraph** (from the Discovery section).
  3. Edit the **Universal Data Reader**:
    - (a) In the **File URL** field, specify the steps input field in the `data-in` directory.
    - (b) Set the **Number of skipped records per source** to 1.
    - (c) Optionally, specify a **Component name**.
  4. Add an edge connecting the **Universal Data Reader** to the **Transaction RunGraph** component. Create a new metadata file for the edge using the **Extract from flat file** option and specifying the steps input file as the **File URL**.
  5. Edit the **Transaction RunGraph** component.
    - (a) Specify the **Endeca Server Host** and **Endeca Server Port**.
    - (b) In the **Upon failure** field, specify the action you want to component to take when a transaction fails. Options include:
      - **Rollback**

Roll the data store back to the state it was in before the outer transaction started, then commit the transaction.
      - **Commit**

Commit those changes that have been made successfully before the failure occurred, then commit the outer transaction.
      - **Do nothing**

Stop processing changes, but do nothing more. The outer transaction is left open and running, which means you need to stop the outer transaction manually.
- For more information, see [Committing or rolling back an outer transaction on page 99](#).
6. Save the graph.

## Committing or rolling back an outer transaction

You can build graphs that commit or roll back an outer transaction that failed to commit successfully.

This procedure assumes that you have followed the recommended practice described earlier. Specifically, this procedure assumes you have specified the `OUTER_TRANSACTION_ID` global variable as empty in the project's `workspace.prm` file and have used that global variable in the components in your project.

In some instances, a graph that starts an outer transaction may fail to commit the transaction. An uncommitted transaction may occur, for example, when you are creating a new graph and troubleshooting its sub-graphs. If any of the sub-graphs fail, the entire graph running an outer transaction may fail also.

Since only one outer transaction can be in progress at a time, if a graph running an outer transaction fails, you cannot run any other graphs that start outer transactions until the outer transaction that is in progress is committed. In such cases, you can commit an outer transaction manually.

Typically, you may need to close an already running outer transaction after you receive a transaction-related error, when trying to run one of your graphs. If you have followed recommended practice, the outer transaction will be running with an ID of "transaction".

This topic describes how to create graphs that either commit or roll back a running transaction using operations of the Transaction Web Service:

- A Commit Transaction graph uses the `commitOuterTransaction` operation to end a transaction. If an outer transaction with the specified ID (usually "transaction") is in progress and if the operation succeeds, the Endeca data domain commits the changes to the index made within this outer transaction, and starts processing unqualified queries and updates against this version of the index.
- The Rollback Transaction graph uses the `rollbackOuterTransaction` operation to roll back an outer transaction. If a running outer transaction fails, use this operation to roll back to the previously-committed version of the index and stop the transaction.

Before running a commit or rollback transaction graph, confirm that the data domain instance is running and that the Transaction Web Service is available by issuing a URL command from your browser, similar to the following example. Be sure to use the correct port number of your Endeca server (7770 by default) and the name of the Endeca data domain ("bikes" in the following example).

```
http://localhost:7770/ws/transaction/bikes?wsdl
```

To create a commit transaction or rollback transaction graph:

1. Create an empty graph and add a **WebServiceClient** component.
2. Edit the **Web Service Client**.
  - (a) In the **WSDL URL** field, specify the URL of the Transaction Web Service as illustrated above. Remember to use the port and data store of your implementation.
  - (b) In the **Operation name** field, choose either `commitOuterTransaction` or `rollbackOuterTransaction`.
  - (c) Access the **Edit request structure** dialog and in the **Generate request** field, enter one of the following code blocks (Note: The following code assumes you are following recommended practice.):

For a Commit Transaction graph:

```
<ns:request xmlns:ns="http://www.endeca.com/MDEX/transaction/1/0">
  <ns:commitOuterTransaction>
    <ns:OuterTransactionId>transaction</ns:OuterTransactionId>
  </ns:commitOuterTransaction>
</ns:request>
```

For a Rollback Transaction graph

```
<ns:request xmlns:ns="http://www.endeca.com/MDEX/transaction/1/0">
  <ns:rollbackOuterTransaction>
    <ns:OuterTransactionId>transaction</ns:OuterTransactionId>
  </ns:rollbackOuterTransaction>
```

```
</ns:request>
```

3. Save the graph.

## Performance impact of transactions

Running an outer transaction does not affect performance of the Oracle Endeca Server.

However, when an outer transaction is in progress (especially if it is running update operations on a large amount of data), it increases the disk usage, resulting in higher disk high-water mark values (Linux).



## Chapter 7

# Loading Data from Special Repositories

---

Integrator ETL includes customized features to load data from an Integrator Acquisition System record store and from Oracle Business Intelligence Server.

[Loading data from an IAS Record Store](#)

[Loading Data from Oracle Business Intelligence Server](#)

## Loading data from an IAS Record Store

Use the **Record Store Reader** component to read records from an Integrator Acquisition System (IAS) record store instance into an Integrator ETL graph.

The Integrator Acquisition System provides the ability to crawl a variety of data sources to acquire data for Oracle Endeca Information Discovery applications. Data sources include file systems, content management systems, Web servers, and custom data sources. IAS converts the content of the documents and files it crawls into Endeca records and stores the records in a Record Store instance. IAS can crawl data sources multiple times, updating the data domain with new records for the new content and data as well as updating existing data.

The graph can then use an integration component (such as the **Merge Records** or **Bulk Load** component) to load the records into an Endeca data domain. The graph can also include additional components, such as a **Text Enrichment** component or a **Text Tagger** component, to manipulate or enrich the data prior to loading it.

The **Record Store Reader** component provides two options for reading data from the IAS record store:

- Set the value of the **Read Type** field to `Full Extract` to read the latest generation of records currently stored in the Record Store. This configuration is best used for initial loads or a baseline updates.
- Set the value of the **Read Type** field to `Incremental` to read only records that have been added or modified since the last committed generation in the Record Store that was read by the client specified in the **IAS Client ID** field. This configuration is best used for incremental updates to a data domain.

### Prerequisites

Before using the **Record Store Reader**, you must:

- Install the Integrator Acquisition System (IAS) Version 3.2.x.
- Configure and run an IAS crawl to populate the IAS Record Store instance with Endeca records.

For full information on IAS, including how to configure and run a crawl, see the *Developer's Guide*.

[Configuring the Record Store Reader](#)

[Generating edge metadata with the Record Store Wizard](#)

## Configuring the Record Store Reader

The **Record Store Reader** reads Endeca records from an IAS Record Store into a graph.

To configure the Record Store Reader:

1. In **IAS Service Host**, specify the name or IP address of the machine where the Integrator Acquisition System is installed.
2. In **IAS Service Port**, specify the port on which the IAS service is listening. The default value is 8401.
3. If IAS is installed on a WebLogic Server container, in **IAS Service Context Root**, specify the context root of the IAS service web application. The default IAS context root is `ias-service`.



**Note:** If you install IAS to a Jetty container, you can ignore this field.

4. In **IAS Record Store Instance**, specify the name of the Record Store whose records you want to read into the graph.
5. In **IAS Client ID**, enter a name that identifies the Integrator ETL client to the Record Store. The default value is `IntegratorETL`. If multiple graphs access the same Record Store, you should specify a different value in this field for each graph.

The Record Store uses this client ID to keep track of which generations of records in the Record Store have been read by this client.

6. In **IAS Read Type**, choose the type of read you want to perform:
  - **Full Extract** - This option reads the last full generation of records in the Record Store.
  - **Incremental** - This option reads only records that were added or modified since the last time the record store was read by a **Record Store Reader** with a client ID matching the value specified in the **IAS Client ID** field.
7. Check **SSL Enabled** only if you have enabled SSL on the Endeca Server.
8. In **Multi-assign delimiter**, specify a character to separate values in multi-assign fields. The default value is the Unicode DELETE character [U007F].
9. In **Read Batch Size**, specify the number of records in each batch fetched from the IAS Record Store. The default value is 100 records.
10. In **Socket Timeout**, specify the maximum period of inactivity between fetching two consecutive data packets before the fetch operation times out. The default is 300000 milliseconds (5 minutes).

## Generating edge metadata with the Record Store Wizard

You can use the Record Store Wizard to generate an external metadata file by querying the Record Store instance for its properties.

Before you can use the Record Store Wizard:

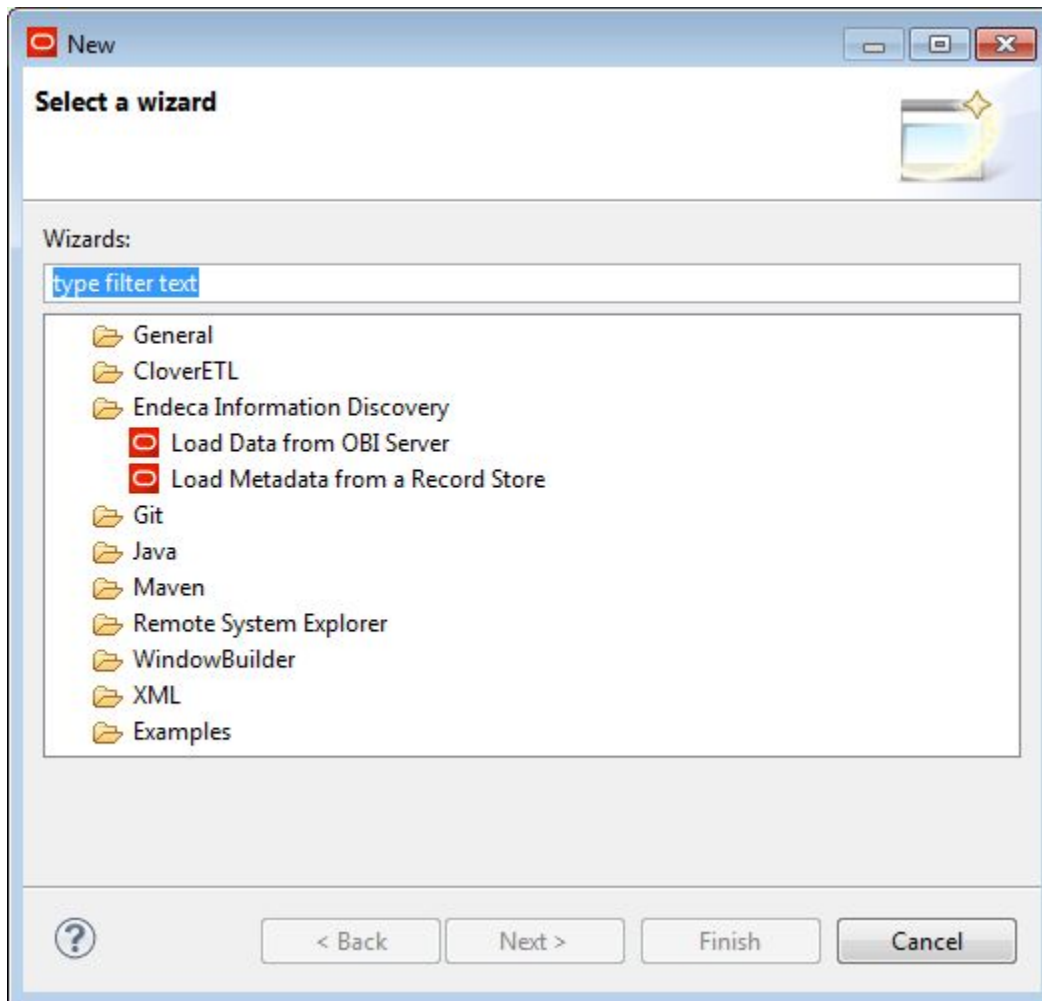
- The IAS Record Store instance for your crawl must have at least one committed generation of records. That is, you must have run a full (baseline) crawl with the output sent to the Record Store.
- The IAS must be running so that the Record Store Wizard can connect to it.
- You must have created an Integrator ETL project and graph for the **Record Store Reader** component: you store the metadata file in this project.

- If you have multi-assign values in the Record Store instance, you must configure the attribute schema for multi-assignment (using the `mdex-property_IsSingleAssign` property). For details, see [Loading the standard attribute schema on page 67](#).

To generate edge metadata with the Record Store Wizard:

1. From your Record Store Reader project, select **File>New>Other**.

Integrator ETL displays the Select a Wizard menu:





2. In the Select a Wizard menu, select **Load Metadata from a Record Store** and then click **Next**. Integrator ETL displays the Record Store Wizard.

**New Metadata from Record Store Wizard**

This wizard creates a new metadata file based on data in a Record Store.

Project:  

File Name:

IAS Service Host:

IAS Service Port:

IAS Service Context Root:

Record Store Instance:

SSL Enabled:  false

? < Back Next > Finish Cancel

3. In the Record Store Wizard, enter these values:
  - (a) In the **Project** field, click **Browse** and select your project from the Folder Selection dialog.
  - (b) In the **File Name** field, enter the pathname (project folder and file name) for the metadata file you want to create. You can use the default name: `iasmetadata.fmt`.
  - (c) In the **IAS Service Host** field, enter the name of the machine on which the IAS Service is running. This name should be the same as the name in the **IAS Service Host** configuration property of the **Record Store Reader** component.
  - (d) In the **IAS Service Port** field, enter the port of the IAS Service. This port should be the same as the one in the **IAS Service Port** configuration property of the **Record Store Reader** component.
  - (e) The value of the **IAS Service Context Root** field defaults to `ias-server`. If you installed IAS into WebLogic Server container and you changed the context root of your IAS installation, change the value of this field to the context root you specified. If you installed IAS to a Jetty container you can ignore this field.

- (f) In the **Record Store Instance** field, enter the name of the Record Store instance that you created. This name should be the same as the one in the **Record Store Instance** configuration property of the **Record Store Reader** component.
- (g) Only toggle the **SSL Enabled** field to true if the IAS Service is SSL enabled.

When you enter data in all fields on the dialog, the Record Store Wizard should resemble the following example:

**New Metadata from Record Store Wizard**  
This wizard creates a new metadata file based on data in a Record Store.

Project:

File Name:

IAS Service Host:

IAS Service Port:

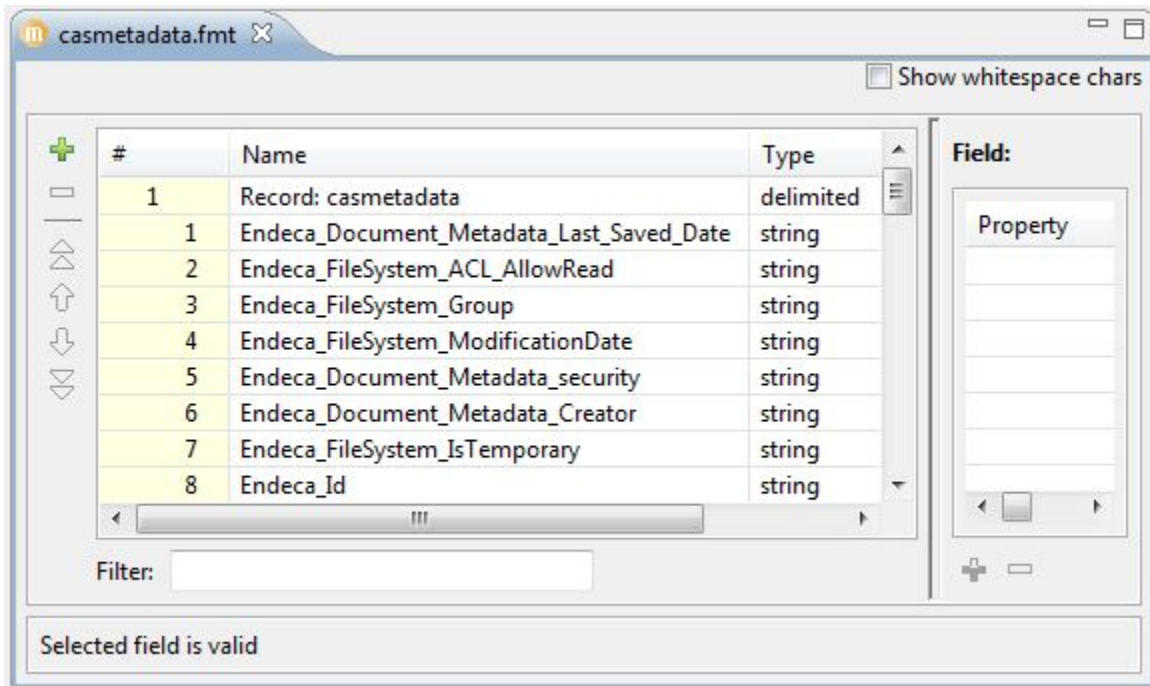
IAS Service Context Root:

Record Store Instance:

SSL Enabled:  true

4. Click the **Finish** button.

The wizard retrieves the record properties from the Record Store instance and displays them in the Integrator ETL Metadata editor, as shown in this truncated example:



The external metadata file is stored in the folder you specified in the **File Name** field.

Assign this metadata to the edge that joins the **Record Store Reader** component to the next component in the graph.

## Loading Data from Oracle Business Intelligence Server

Use the **Load Data from OBI Server** wizard to create a new project that connects to an Oracle Business Intelligence Server (OBI Server) and retrieves data from it.

[Using the Load Data from OBI Server wizard](#)

[Loading Oracle Business Intelligence Server data to Endeca Server](#)

[Configuring OBI Server graphs for Integrator ETL Server](#)

### Using the Load Data from OBI Server wizard

This topic describes how to run the OBI Server wizard to retrieve data from an Oracle Business Intelligence Server (OBI Server).

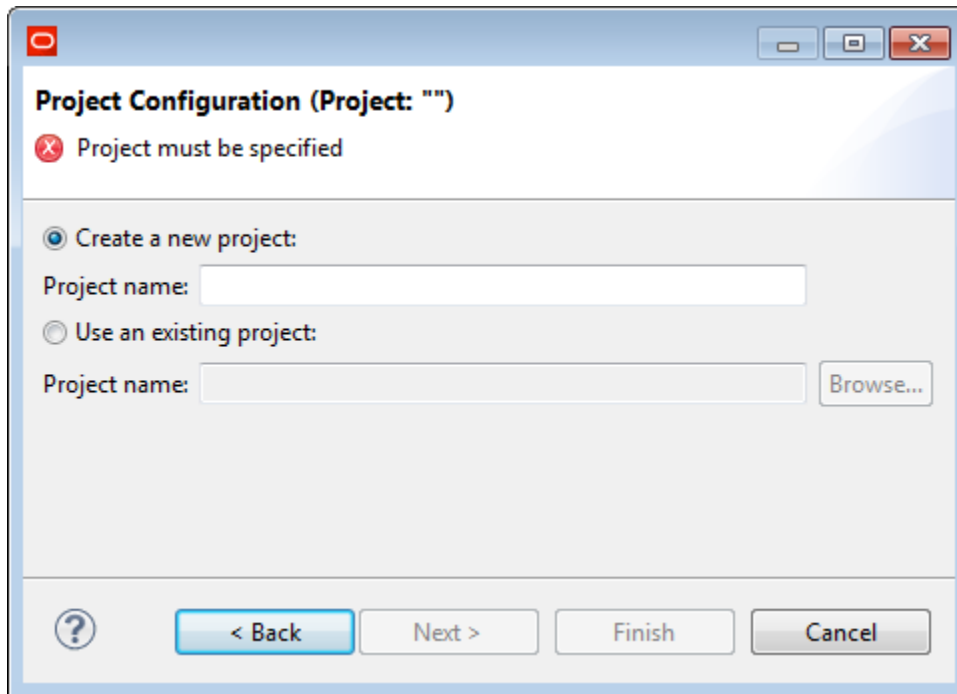
As a pre-requisite, the **Load Data from OBI Server** wizard requires a running OBI Server and a running Endeca Server.

In addition, if you intend to use an encrypted password for the OBI Server connection, make sure that you have set up a Master Password in Integrator ETL, as explained in [Securing graph parameters on page 14](#). Encrypted passwords are used only in secure graph parameters, as that is the only place where Integrator ETL de-encrypts them when running a graph. As a result, instead of storing the OBI password directly in BServer.cfg, the OBI Server Connector Wizard 3.2 stores the password in workspace.prm as **BI\_SERVER\_PASSWORD**.

To create a project to load data from an OBI Server:

1. In the Menu bar, choose **File > New > Project**.  
Integrator ETL displays the **New Project** dialog.
2. Expand the **Endeca Information Discovery** node and select **Load Data from OBI Server**. Click **Next**.

Integrator ETL displays the **Project Configuration dialog** of the Load Data from OBI Server wizard.



3. Select the **Create a new project** radio button. Enter a name for the project. Click **Next**.

Integrator ETL displays the **Endeca Data Domain Configuration** dialog.

4. On this dialog:
  - (a) Enter the **Endeca Server host**. You can enter either the host name or the IP address of the Endeca Server.
  - (b) Enter the **Endeca server** port.
  - (c) Enter the **Data domain name**. If the data domain you specify does not exist, it will be created when you run the Baseline graph in the project.
  - (d) The **Data domain language** defaults to English. If you want to load data from a different language, select it from the drop list. Options include all languages supported by Endeca Server.
  - (e) Enter the **Collection key**.



**Note:** The **Collection key** can only include alphanumeric characters and underscores. The name must begin with either a letter or an underscore. The name cannot begin with a number.

For more information about Collections, see the *Oracle Endeca Server Developer's Guide*.

(f) Click **Next**.

Integrator ETL displays the **OBI Server Configuration** dialog.

The screenshot shows a Windows-style dialog box titled "Load Data from OBI Server". The main title is "OBI Server Configuration (Project: 'LoadOBIDData')". Below the title is the instruction "Configure the connection to the OBI Server." The dialog contains several input fields: "OBI Server host:", "OBI Server port:", "Username:" (for extracting data), and "Password:" (for extracting data). There is a checkbox labeled "Encrypt password" next to the password field. Below these is another checkbox labeled "Use different account to extract RDP metadata". Underneath is a section for "Account with privileges to export RDP metadata" with its own "Username:" and "Password:" fields. A "Connect to OBI Server" button is positioned to the right of the second password field. At the bottom of the dialog, there is a help icon (question mark) on the left and four buttons: "< Back", "Next >", "Finish", and "Cancel".

5. Enter the **OBI Server host** (name or IP address) and **OBI Server port** of the OBI Server from which you want to retrieve data. Enter the **Username** and **Password** of the user you want to use to log in to the OBI Server. The user you specify must have privileges to extract data.

By default, the same user is used to extract all data. If you want to use a different user to extract RDP metadata, check the **Use different account to extract RDP metadata** box. Under "Account with privileges to export RDP metadata", enter the **Username** and **Password** of the user you want to use to extract RDP metadata. The user you specify must have privileges to extract metadata.

6. Click **Connect to OBI Server**.

Integrator ETL attempts to connect to the OBI Server you specified using the connection and authentication data you entered. When you establish a connection, Integrator ETL enables the **Next** button.



**Note:** You must establish a connection before you can continue in the wizard.

7. If you want to encrypt the passwords:
    - (a) Enter `#{BI_SERVER_PASSWORD}` in the **Password** field.
    - (b) Check the **Encrypt password** box.
    - (c) Click **Next**.
  8. If you do not want to encrypt the passwords, click **Next**.
- Integrator ETL displays the **OBI Model** dialog.

**Load Data from OBI Server**

**OBI Model (Project: "LoadOBIData")**  
Select the OBI application and tables to import.

**Subject areas**  
Activities

**Fact tables**

- Service Request Facts
- Financial Account Facts
- Insurance Policy Facts
- Account Facts
- Insurance Claim Facts
- Activity Facts

**Dimension tables**

- Actual End Date
- Actual Start Date
- Employee Organization
- Geography
- Opportunity Close Date
- Partner Employee Organization
- Profile
- Activity
- Asset

Select All Unselect All

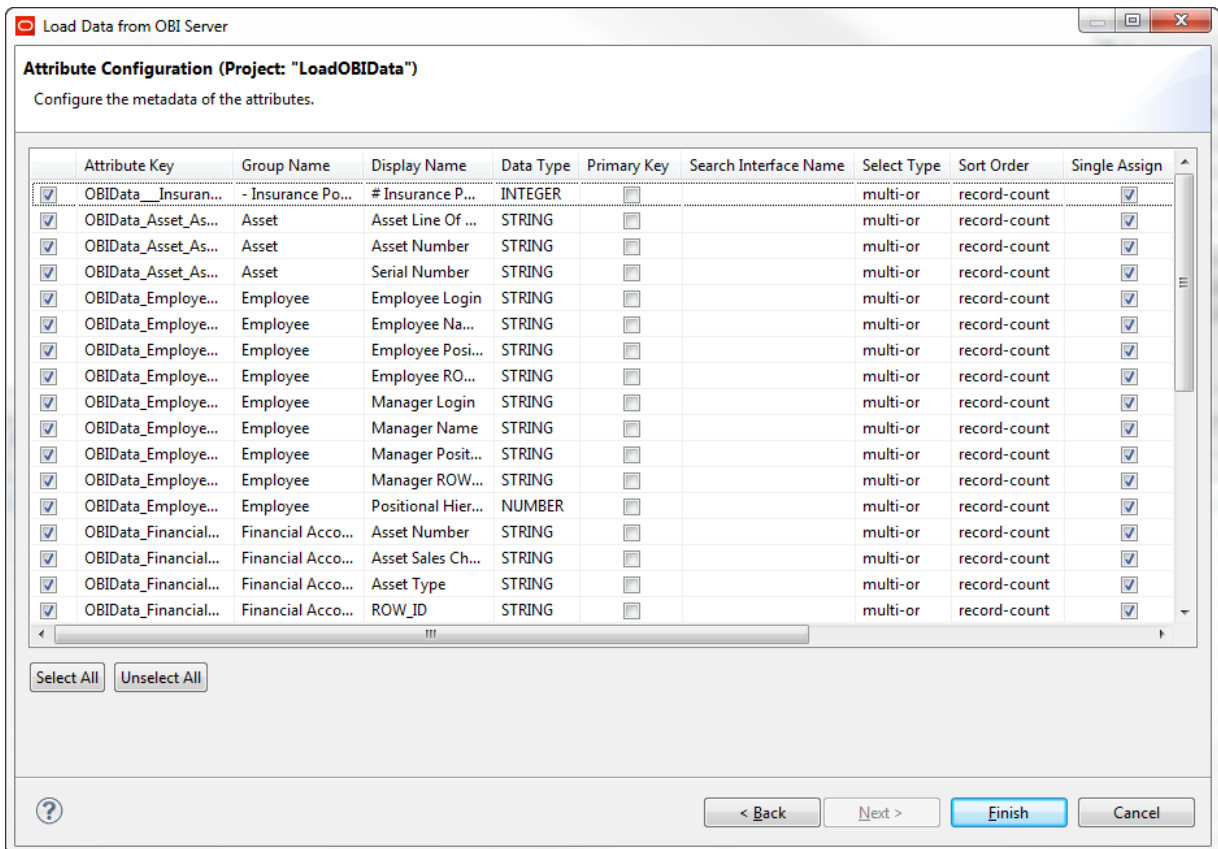
? < Back Next > Finish Cancel

- Click on the **Subject areas** drop list, and choose the OBI subject area from which you want to retrieve data. You can also type a filter in this field to filter the list of available Subject areas. When you start typing, the list of subject areas that match the filter you enter is displayed in a popup dialog. Select the subject area you want to use from the list of matches.

The **Fact tables** and **Dimension tables** for the subject area you selected are listed.

- Check the boxes next to the tables from which you want to retrieve data. To check all boxes in the Dimension table list, click **Select All**. You can click on a table listed in the **Fact tables** field to filter the list of dimension tables. Only the dimension tables associated with the fact table you selected will be listed.
- Click **Next**.

Integrator ETL displays the **Attribute Configuration** dialog populated with guesses about the values for each column in the table.






The following table describes the columns on the **Attribute Configuration** dialog.

Table 7.1: Attribute Configuration Properties

Column Name	Description	Editable
Attribute Key	<p>Name of the standard attribute created from this column.</p> <p>The <b>Collection key</b> specified on the Endeca Data Domain Configuration dialog is appended to the beginning of the value of each <b>Attribute Key</b>.</p> <p>Maps to the <code>mdex-property_Key</code> PDR property.</p>	No
Group Name	<p>Groups to which the attribute belongs</p> <p>Maps to the <code>system-property_GroupMembership</code> PDR property.</p>	<p>Yes</p> <p>Click in the field and modify the text.</p>
Display Name	<p>Name of the attribute displayed in user interfaces.</p> <p>Maps to the <code>mdex-property_DisplayName</code> PDR property.</p>	<p>Yes</p> <p>Click in the field and modify the text.</p>
Data Type	<p>The mdex data type of the standard attribute.</p> <p>Maps to the <code>mdex-property_Type</code> PDR property.</p>	No

Column Name	Description	Editable
Primary Key	<p>Checkbox specifying whether an attribute should be used as the primary key (or uniquePropertyKey) of the collection to which the data will be loaded.</p> <p>If you select multiple attributes, they will be concatenated to create the uniquePropertyKey for ingest into Endeca Server.</p> <p>If you specify a single attribute, that attribute must be unique, and must have a value (it cannot be null or empty). If you specify a combination of attributes, the combination of values must be unique, and cannot result in a null or empty value.</p> <p>If you do not specify an attribute, a primary key value will be created automatically using a simple incrementing value.</p> <p>The Primary Key is always named RecSpec_&lt;collection_key&gt;, where collection_key is the value you specified in the <b>Collection key</b> field on the Endeca Data Domain Configuration dialog.</p>	Yes

Column Name	Description	Editable
Search Interface	The search interface assigned to the standard attribute. The search interface controls the search behavior of the standard attribute.	<p>Yes if the data type of the attribute is <code>mdex:string</code>. Otherwise, no.</p> <p>Click in the field and enter the names of the search interfaces you want to use for the standard attribute. You can enter multiple search attributes in this field, separated by commas.</p> <p> <b>Note:</b> Define the Search Interfaces in the data domain configuration before loading data from OBI Server.</p>
Select Type	<p>Defines the multi-select behavior of the standard attribute.</p> <p>Maps to the <code>system-navigation_Select</code> PDR property.</p>	<p>Yes</p> <p>Click in the field and choose the value from the drop list. Available values include:</p> <ul style="list-style-type: none"> <li>• <code>single</code> Users can select only one refinement from this attribute.</li> <li>• <code>multi-and</code> Users can select multiple refinements from the attribute. Records are only returned if the value of the assignment matches all selected refinements.</li> <li>• <code>multi-or</code> Users can select multiple refinements from the attribute. Records are returned if they match any of the selected refinements.</li> </ul> <p>Defaults to <code>multi-or</code>.</p>

Column Name	Description	Editable
Sort Order	<p>Specifies the order in which to display refinements in navigation.</p> <p>Maps to the <code>system-navigation_Sorting</code> PDR property.</p>	<p>Yes</p> <p>Click in the field and choose the value from the drop list. Available values include:</p> <ul style="list-style-type: none"> <li>• <code>lexical</code> Sorts refinements in alphabetical or numerical order.</li> <li>• <code>record-count</code> Sorts by number of records, in descending order.</li> </ul> <p>Defaults to <code>record-count</code>.</p>
Single Assign	<p>Specifies whether the standard attribute is single-assign (checked, or true) or multi-assign (unchecked or false).</p> <p>Maps to the <code>mdex-property_IsSingleAssign</code> PDR property.</p>	Yes
Value Searchable	<p>If checked (or true), value search is enabled for the standard attribute. If unchecked (or false), value search is disabled for the standard attribute.</p> <p>This option should only be checked if the value of the <b>Data Type</b> is <code>STRING</code>.</p> <p>Maps to the <code>mdex-property_IsPropertyValueSearchable</code> PDR property.</p>	Yes
Text Searchable	<p>If checked (or true), text search is enabled for the standard attribute. If unchecked (or false), text search is disabled for the standard attribute.</p> <p>Maps to the <code>mdex-property_IsTextSearchable</code> PDR property.</p>	Yes

Column Name	Description	Editable
Show Record Counts	If checked (or true), record counts are shown for refinements. If unchecked (or false), record counts are not shown.  Maps to the <code>system-navigation_ShowRecordCounts</code> PDR property.	Yes

The following table describes mappings from OBI Server to Integrator ETL to Endeca Server.

Table 7.2: Data Type Mappings

OBI type name	OBI data type	Integrator ETL data type name	mdex data type name
BIGINT	-5	Long	mdex:long
BINARY	-2	N/A	N/A
CHAR() for bit data	-7	N/A	N/A
CHAR	1	String	mdex:string
DATE	9	Date	mdex:dateTime
DECIMAL	3	Number	mdex:double
DOUBLE	8	Number	mdex:double
FLOAT	6	Number	mdex:double
INTEGER	4	Integer	mdex:int
LONGVARBINARY	-4	N/A	N/A
LONGVARCHAR	-1	String	mdex:string
NUMERIC	2	Number	mdex:double
REAL	7	Number	mdex:double
SMALLINT	5	Integer	mdex:int
TIME	10	Date	mdex:dateTime
TIMESTAMP	11	Date	mdex:dateTime

OBI type name	OBI data type	Integrator ETL data type name	mdex data type name
TINYINT	-6	Integer	mdex:int
VARBINARY	-3	N/A	N/A
VARCHAR	12	String	mdex:string

12. Click **Finish**.

Integrator ETL runs the wizard processing.

The wizard creates a new project with the following artifacts, listed by directory:

- config-in
  - BaselineSteps.csv
  - AttributeGroups\_<collection\_key>.csv
  - AttributeMetadata\_<collection\_key>.csv
  - AttributeSearchability\_<collection\_key>.csv
  - OBI metadata file named <host>-<port>-<objectID>.xml, where:
    - <host> is the value you entered for the **OBI Server host** on the OBI Server configuration dialog
    - <port> is the value you entered for the **OBI Server port** on the OBI Server configuration dialog
    - <objectID> specifies the metadata object ID. Valid values are 2035 (specifies logical table metadata) or 2004 (represents logical complex join metadata).
- conn
  - A connection configuration file named BIServer.cfg
  - bijdbc.jar
- data-in
  - A query statement file: QueryStatement\_<collection\_key>.sql
- graph
  - Baseline.grf
  - InitDataDomain.grf
  - LoadConfiguration\_Collection\_<collection\_key>.grf
  - LoadData\_Collection\_<collection\_key>.grf
  - SetDataDomainLanguage.grf
- meta
  - DataRecord\_<collection\_key>.fmt
  - DataRecordRecSpec\_<collection\_key>.fmt

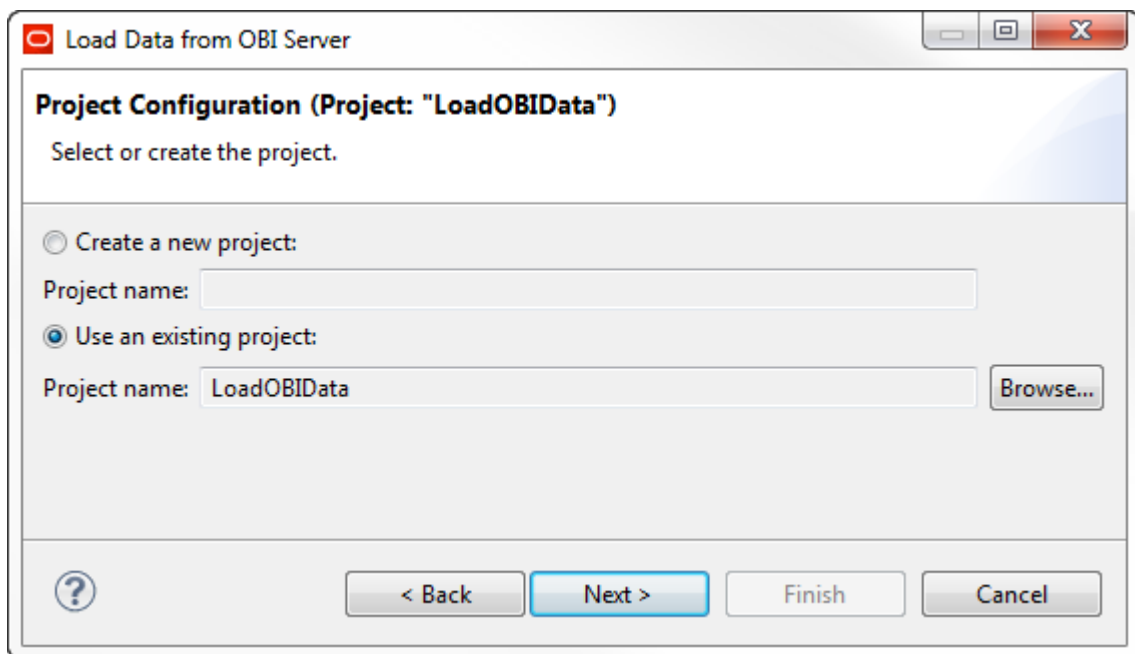
The Endeca data domain configurations specified on the **Endeca Data Domain Configuration** dialog are stored in the project's `workspace.prm` file.

## Adding collections to a Load data from OBI project

You can add multiple collections to a Load data from OBI project.

To add a collection to a Load data from OBI project:

1. In the Menu bar, choose **File > New > Project**.  
Integrator ETL displays the **New Project** dialog.
2. Expand the **Information Discovery Node** and select **Load Data from OBI Server**. Click **Next**.  
Integrator ETL displays the **Project Configuration dialog** of the Load Data from OBI Server wizard.



3. Choose a load data from OBI project:
  - (a) Select the **Use an existing project** radio button.
  - (b) Click the **Browse** button  
Integrator ETL displays the **Select a Project** dialog.
  - (c) Select the Load data from OBI project to which you want to add the collection and click **OK**.  
Integrator ETL enters the project you selected in the **Project Name** field.
  - (d) Click **Next**.  
Integrator ETL displays the **Endeca Data Domain Configuration** dialog, populated with the Endeca Server and data domain specified for the Load data for OBI project you selected.
4. Enter a new **Collection key**.

5. Click **Next**.

Integrator ETL displays the **OBI Server Configuration** dialog, populated with the OBI Server data for the Load data for OBI project you selected.

6. Click **Connect to OBI Server**.



**Note:** You must connect to the OBI server you can continue with the wizard.

Integrator ETL offers to download OBI metadata again. Click **No** to use the OBI metadata you already downloaded. Click **Yes** to download OBI metadata again.



**Note:** Do not change the **User name**. Only one user can be used to load OBI data in a single project. If you want to use different users to load OBI data to different collections in a data domain, you must create a different project for each user.



7. Click **Next**.

Integrator ETL displays the **OBI Model** dialog.

**Load Data from OBI Server**

**OBI Model (Project: "LoadOBIData")**  
Select the OBI application and tables to import.

**Subject areas**  
Activities

**Fact tables**

- Service Request Facts
- Financial Account Facts
- Insurance Policy Facts
- Account Facts
- Insurance Claim Facts
- Activity Facts

**Dimension tables**

- Actual End Date
- Actual Start Date
- Employee Organization
- Geography
- Opportunity Close Date
- Partner Employee Organization
- Profile
- Activity
- Asset

Select All Unselect All

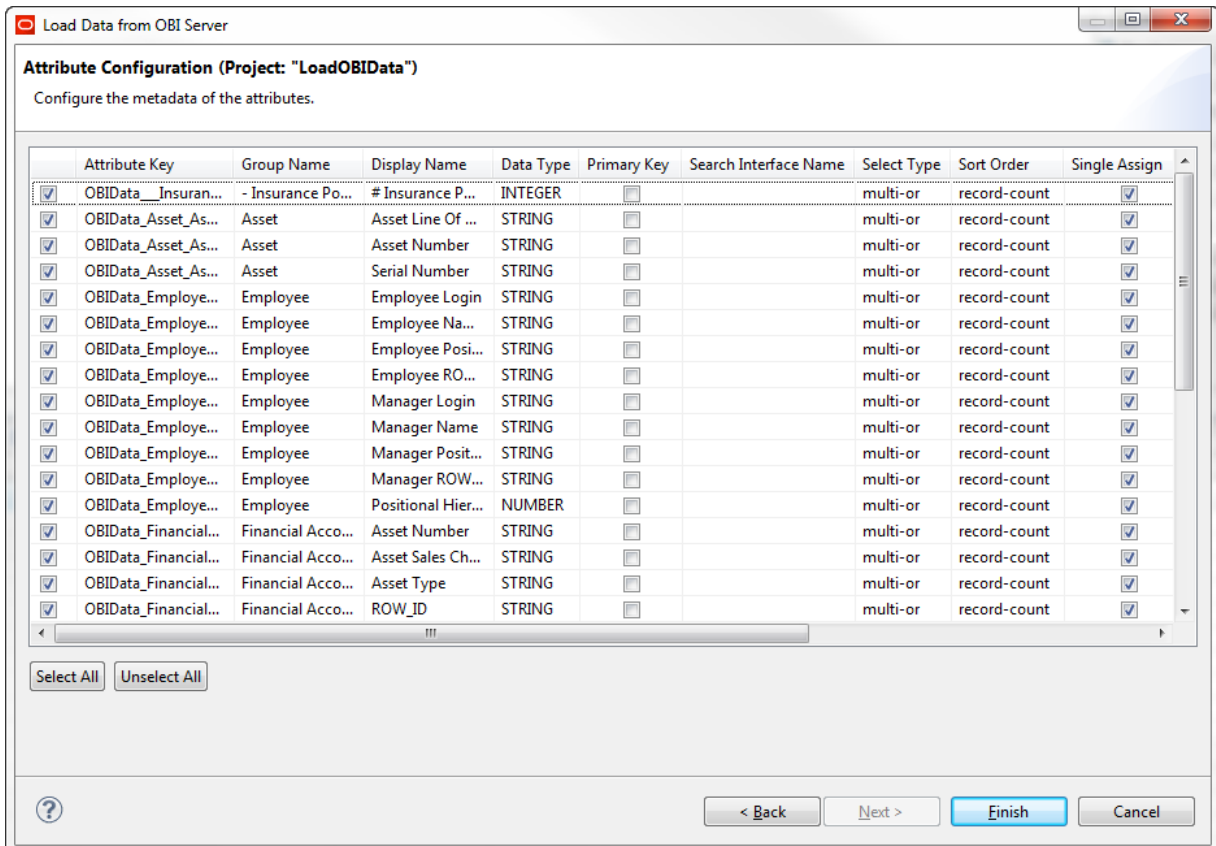
? < Back Next > Finish Cancel

8. Click on the **Subject areas** drop list, and choose the OBI subject area from which you want to retrieve data. You can also type a filter in this field to filter the list of available Subject areas. When you start typing, the list of subject areas that match the filter you enter is displayed in a popup dialog. Select the subject area you want to use from the list of matches.

The **Fact tables** and **Dimension tables** for the subject area you selected are listed.

9. Check the boxes next to the tables from which you want to retrieve data. To check all boxes in the Dimension table list, click **Select All**. You can click on a table listed in the **Fact tables** field to filter the list of dimension tables. Only the dimension tables associated with the fact table you selected will be listed.
10. Click **Next**.

Integrator ETL displays the **Attribute Configuration** dialog populated with guesses about the values for each column in the table.



11. Click **Finish**.

Integrator ETL runs the wizard processing.

The wizard adds the following files to the project:

- config-in
  - AttributeGroups\_<collection\_key>.csv
  - AttributeMetadata\_<collection\_key>.csv
  - AttributeSearchability\_<collection\_key>.csv
- data-in
  - A query statement file: QueryStatement\_<collection\_key>.sql

- graph
  - `LoadConfiguration_Collection_<collection_key>.grf`
  - `LoadData_Collection_<collection_key>.grf`
- meta
  - `DataRecord_<collection_key>.fmt`
  - `DataRecordRecSpec_<collection_key>.fmt`

## Loading Oracle Business Intelligence Server data to Endeca Server

To load Oracle Business Intelligence data to Endeca Server, run the graph `Baseline.grf`.

The **Load Data from OBI Server** wizard creates graphs to generate default configurations for the data domain. If you want different configurations for the data domain, modify the existing graphs (particularly the `LoadConfiguration_<collection_key>.grf` graph) or add new graphs for load your configurations. If you add graphs, be sure to modify the input file for the `Baseline.grf` graph.

## Configuring communication security for OBI graphs

Load data from OBI graphs are configured for secure communication by default.

The following properties are added to the `workspace.prm` file for every Load OBI data project:

- `SSL_ENABLED`

Specifies whether SSL communication is enabled with the OBI Server. Valid values are

  - `true`

SSL communication is enabled on the OBI Server
  - `false`

SSL communication is not enabled on the OBI Server

Defaults to `true`.
- `HTTP_PROTOCOL`

Specifies which protocol to use to communication with the OBI Server. Valid values are

  - `https`

Use secure HTTP (HTTPS) to communicate with the OBI Server.
  - `http`

Use unsecure HTTP (HTTP) to communication with the OBI Server.

Defaults to `https`.

The default configuration of these properties is

```
SSL_ENABLED=true
HTTP_PROTOCOL=https
```

This configuration supports secure communication with the OBI Server.

To configure unsecure communication with the OBI Server, change the value of the `SSL_ENABLED` property to `false` and change the value of the `HTTP_PROTOCOL` property to `http`:

```
SSL_ENABLED=false
HTTP_PROTOCOL=http
```

## Configuring OBI graphs for encrypted passwords

If you encrypted the OBI Server password, must configure the password encryption key to run the graphs.

To configure OBI graphs for encrypted passwords:

1. To run the load as an outer transaction, in `Baseline.grf`, modify the **RunGraph** component (named Run Steps by default).
  - (a) Uncheck **The Same JVM** box (in other words, set it to false).
  - (b) In the **Command line arguments** property, add the argument `-pass <encryption_key>`.
  - (c) If you use SSL, in the **Alternate JVM command line**, specify the paths to the keystore and trust store, and the passwords you specified for them. Append "-cp" to the alternate JVM command line.
 

For example: `java -Djavax.net.ssl.keyStore=C:\Oracle\Endeca\Discovery\3.2.0\IntegratorETL\SSL\endecaServerClientCert.ks -Djavax.net.ssl.keyStorePassword=endeca123 -Djavax.net.ssl.trustStore=C:\Oracle\Endeca\Discovery\3.2.0\IntegratorETL\SSL\endecaServerTrustStore.ks -Djavax.net.ssl.trustStorePassword=endeca123 -cp`
  - (d) Save your changes.
2. You may also want to configure the `LoadData_<collection_key>.grf` so you can run it individually:
  - (a) In the Navigator pane, select the graph.
  - (b) In the Menu bar, choose **Run > Run Configuration**.
 

Integrator ETL displays the Run Configuration wizard.
  - (c) In the **Password** field, enter the encryption key.

## Configuring OBI Server graphs for Integrator ETL Server

Graphs that connect to OBI Server require modification to run on Integrator ETL Server.

If you want to run a graph that connects to OBI Server on Integrator ETL Server, you must define an absolute path to the sandbox root. Note that defining an absolute path to the sandbox root is recommended practice when creating a sandbox. If you do not specify the absolute path, the graph cannot find the OBI JDBC driver (`bijdbc.jar`) and will fail.

If you encrypt the OBI Server password, you must also configure your graphs to use the encrypted password:

- In the Sandboxes tab, select the `Baseline.grf` graph, go to the **File editor** tab and remove `sameInstance="false"`.
- Add the property `password` to the `workspace.prm` file. The value of this property is the encryption key. All graphs in the project will use the encryption key.



**Note:** You can also define the property in individual graphs. In the Sandboxes tab, select the `LoadData_<collection_key>` graph. Go to the Config properties tab, and create the property . The value of this property is the encryption key.



## Chapter 8

# Enhancing Text

---

Integrator ETL provides two options for enhancing text before loading it to an Endeca Server data domain: text enrichment and text tagging. You can also detect the language of input text fields.

*Choosing Text Enrichment or Text Tagging*

*Using Text Tagger components*

*Using Text Enrichment*

*Detecting text language*

## Choosing Text Enrichment or Text Tagging

Use these guidelines to choose between text enrichment and text tagging.

- Use the **Text Tagger White List** component when you want to match on specific terms; for example, you want to find the word "leak", but not any variations on the word.
- Use the **Text Tagger Regex** component when you want to match variations of known terms; for example, you want to match on "leaks", "leakage", or "leaking", as well as on "leak".
- Use the **Text Enrichment** component when you want to match on combinations of known terms. This functionality is provided by the categorization using query topics. For example, you want to match on "leak", "leaking", or "leakage" near the word "oil".

For more information about categorization and the query topics, see <https://www.lexalytics.com/technology/categorization>.

- Use the **Text Enrichment** component when you don't know what themes or terms might be found in the text you want to evaluate.

## Using Text Tagger components

Use Text Tagger components to enhance your data with tags.

Use these text tagger components when you know the content that you want to match on to apply a tag.

- Use the **Text Tagger White List** component when you know the specific content (words or phrases) you want to match, and you have specific content with which you want to tag the record.
- Use the **Text Tagger Regex** component when you know general patterns you want to match in content, and apply as tags.

You can use these components independently of each other (neither Text Tagger component depends on the other), or you can use both types in the same graph.

*Overwriting and appending target values*

*Text Tagger input and output metadata*

*Using the Text Tagger Whitelist component*

*Using the Text Tagger Regex component*

## Overwriting and appending target values

The **Overwrite Target Field** configuration property of the components determines how tags are written to the output field.

If you specify `False` for this field (which is the default), the tags added by the Text Tagger are appended to any value that may already exist in the field as it was input to the Text Tagger.

If you specify `True` for this field, the tags added by the Text Tagger overwrite any value that may exist in the field as input to the Text Tagger. If not match is found for an existing record, a null value overwrites any existing value.

In both cases, the character specified in the **Multi-assign delimiter** field is used to separate values when multiple tag values are added to the field.

## Text Tagger input and output metadata

Output metadata fields for the Text Tagger components need not be in the same order as input metadata fields so long as the name of the field in the output metadata is the same as the name of the field on the input metadata.

In addition, the output metadata from a Text Tagger component must contain all the fields contained in the input metadata.

## Using the Text Tagger Whitelist component

The **Text Tagger Whitelist** component adds tags based on a white list of terms to match.

The **Text Tagger Whitelist** component takes an input file that lists terms to match in an unstructured text property, and the associated terms to output as tags to the record. When the component finds a match in the specified field, it writes the specified tag values to the output field. You can only specify one field and one input file per instance of the **Text Tagger Whitelist** component. If you want to tag multiple fields, or if you apply multiple white lists to the same field, you must use additional instances of the component.

The **Text Tagger Whitelist** component does not impose any restrictions on the source of the data. You can use a database, a delimited file, or rich text files such as PDF or HTML files. Nor does the component impose any restriction on the reader component used to read the data in. You can use the **Universal Data Writer**, the **Record Store Reader**, the Database Reader, or any other reader appropriate to load the data from the source. The only restriction the **Text Tagger Whitelist** component imposes is that the component can parse the value of the field you specify as the input. If the value of the field cannot be parsed, the behavior is undefined (in other words, it is unpredictable).

## Text Tagger Whitelist input tags-rules

The input for the **Text Tagger Whitelist** component uses a fixed metadata schema and a specific ordering.

The **Text Tagger Whitelist** component requires the metadata of the input tags-rules to include two properties in the specified order:

- **SearchTerm**

The terms the component searches for in the input text field.

- **TagValue**

The value to add to the tag output field when a match is found for the associated SearchTerm.

The data type of both properties is string. The names of the properties are significant and must be spelled and capitalized as listed above.

The first row of the input file is the header row. This row must use the metadata schema, as illustrated in the following pipe-delimited example:

```
SearchTerm|TagValue
```

The following text illustrates a simple example input file:

```
SearchTerm|TagValue
United States|American topic
France|French topic
Japan|Japanese topic
```

In this example, if the string “United States” is found in the content of the source property, the tag “American topic” is added to the output target field. If the string “France” is found, the tag “French topic” is added to the output target field. If the string “Japan” is found, the tag “Japanese topic” is added to the output field.

You can use any supported input source as the input for the tags-rules. For example, you may choose to use a delimited file, such as a .csv file, and read the input into the graph using a **Universal Data Reader**. Or you can store your tag-rules in a database and read them using a **Database Reader**.

## Adding the Text Tagger Whitelist component to a graph

This topic describes the requirements for adding the **Text Tagger Whitelist** component to a graph.

The **Text Tagger Whitelist** component requires two inputs:

- The source data to search for matches to tag

This data can be read in by any appropriate reader component. You can also include any appropriate processing prior to inputting the data to the **Text Tagger Whitelist** component.

- The tag rules input

This data can be read in by any appropriate reader (for example, a **Universal Data Reader** for a .csv file, or a **Database Reader** for database input). No additional processing is usually performed on this input.

## Configuring the Text Tagger Whitelist component

When you add the **Text Tagger Whitelist** component to a graph, configure the following fields:

- In the **Source Field Name** field, specify the name of the property in the input records that you want to search for matches to the SearchTerm values.



- In the **Target Field Name** field, specify the name of the output field to which you want to write the tags when a match is found for a SearchTerm value.
- In the **Overwrite Target Field**
  - Check the box (configure the field to `True`) if you want to overwrite any existing value in the target field with tags written by the **Text Tagger Whitelist** component.
  - Leave the box unchecked (configure the field to `False`) if you want to append tags written by the **Text Tagger Whitelist** component to any existing value in the target field.

For additional information, see [Overwriting and appending target values on page 127](#).

- If you want searches to match the case of the values in the SearchTerm, check the **Case Sensitive Matches** box (in other words, set the field to `True`). If you want to match the values regardless of case (case-insensitive matches), leave the box unchecked (in other words, set the field to `False`).
- In the **Multi-assign Delimiter** field, specify the character to use to separate tags if you write multiple tags to the output field specified in the Target Field property. See also [Multi-assign delimiter on page 207](#).
- In the **Search Term Maximum Characters Length** field, specify the maximum number of characters for values in the SearchTerm. For further details, see [Search Term Lengths on page 129](#).
- The **Number of Threads** defaults to 1.

When processing large whitelists (more than 1000 Search Terms), or very large text fields, or both, you may notice that processing takes a long time. In that situation, you can improve performance by increasing the number of threads used in white list text tagging processing. A good starting point is to specify a number of threads that matches the number of processing cores available. Actual performance is affected by a number of factors, including the number of cores available and other processing taking place on the machine, so you may need to adjust the number of threads to achieve the desired results.

### Search term lengths

The **Search Term Maximum Characters Length** field of the **Text Tagger Whitelist** component specifies the maximum length, in characters, of values in the SearchTerm property.

The value of this field must be larger than the number of characters in the search term tag-rules input. For example, if the longest search term is 50 characters, the value of this field must be 51 or higher.

The reason for this requirement is the search behavior of the **Text Tagger Whitelist** component. As the component traverses the source record, it grabs text in chunks based on the number of characters specified in the Search Term Maximum Characters. For example, using the setting just mentioned (51), the component grabs 51 characters at a time. If the search term is longer than the number of characters specified in this field, the term will never be found, because it will be too long to match the text the component is currently assessing.

### Text Tagger Whitelist edges

The **Text Tagger Whitelist** component can use basic edges unless the source or target components require a different edge.

See [Connecting two components with an edge on page 26](#) for details.

Be sure to define the output field in the edge from the **Text Tagger Whitelist** component to the next component to ensure that the data for this field is passed on for further processing.

## Using the Text Tagger Regex component

The **Text Tagger Regex** component uses regular expressions (regex) both to search for matches and to render output.

The component takes two inputs:

- The **Search Pattern** is a set of one or more regular expressions that is applied to the specified input property to search for matches.
- The **Render Pattern** is the replacement string written out to the target field when a regular expression results in a match in the input data.

The patterns are defined in an input file.

You can only specify one field and one input file per instance of the **Text Tagger Regex** component. If you want to tag multiple fields, you must use additional instances of the component.

The component implements Oracle's `java.util.regex` package to parse and match the pattern of the regular expression. Therefore, the supported regular-expression constructs are the same as those in the documentation page for the `java.util.regex.Pattern` class at this URL:

<http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>

Valid constructs you can use include:

- Escaped characters, such as `\t` for the tab character.
- Character classes (simple, negation, range, intersection, subtraction). For example, `[^abc]` means match any character except a, b, or c, while `[a-zA-Z]` means match any upper- or lower-case letter.
- Predefined character classes, such as `\d` for a digit or `\s` for a whitespace character.
- POSIX character classes (US-ASCII only), such as `\p{Alpha}` for an alphabetic character, `\p{Alnum}` for an alphanumeric character, and `\p{Punct}` for punctuation.
- Boundary matchers, such as `^` for the beginning of a line, `$` for the end of a line, and `\b` for a word boundary.
- Logical operators, such as `X|Y` for either X or Y.

For a full list of valid constructs, see the `Pattern` class documentation page at the URL listed above.

The following code illustrates example of a useful regular expression that uses the POSIX `\p{Alnum}` construct:

```
^\p{Alnum}[\p{Alnum}\.\-' ]+$
```

This **Search Pattern** input regular expression matches only terms that have at least two characters, start with an alphanumeric character, and contain only alphanumeric, period, dash, apostrophe, and space characters. (The apostrophe ensures that terms such as `O'Malley` match).

In addition, you can define capture groups or match groups. You can include the results of these capture groups in the Render Pattern using the syntax `$1`, where 1 is the index of the capture group you want to include. Note that `$0` refers to the entire matched expression. For example, if you defined a regular expression that included two capture groups, you could define the following Render Pattern:

```
$0 includes $1 and $2
```

For additional details, see <http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#cg>.

## Text Tagger Regex input patterns file

The input for the **Text Tagger Regex** component uses a fixed metadata schema and a specific ordering.

The **Text Tagger Regex** component requires the metadata of the input patterns file to include two properties in the specified order:

- **SearchPattern**

A set of one or more regular expressions that is applied to the specified input property to search for matches.

- **RenderPattern**

A replacement string written out to the target field when a regular expression results in a match in the input data.

The data type of both properties is string. The names of the properties are significant and must be spelled and capitalized as listed above.

The first row of the input file is the header row. This row must use the metadata schema, as illustrated in the following pipe-delimited example:

```
SearchTerm|RenderPattern
```

The following text illustrates a simple example input file:

```
SearchTerm|RenderPattern
\b(\w*)(J\w+) Smith\b| $2 Found
\b(J\w*)\b|First Name: $1
^(.*) (.*)$|$2 and $1
```

You can use any supported input source as the input for the tags-rules. For example, you may choose to use a delimited file, such as a .csv file, and read the input into the graph using a **Universal Data Reader**. Or you can store your regular expressions in a database and read them using a **Database Reader**.

## Adding Text Tagger Regex to a graph

The **Text Tagger Regex** component requires two inputs: the source data to search for matches and the input patterns to search and render.

When adding the **Text Tagger Regex** to a graph, configure the following fields:

- In the **Source Field Name** field, specify the name of the property in the input records that you want to search for matches to the regular expression specified in the **Search Pattern** input.
- In the **Target Field Name** field, specify the name of the output field to which you want to write the output generated by the regular expression in the **Render Pattern** input when a match is found.
- In the **Overwrite Target Field**
  - Check the box (configure the field to `True`) if you want to overwrite any existing value in the target field with tags written by the **Text Tagger Whitelist** component.
  - Leave the box unchecked (configure the field to `False`) if you want to append tags written by the **Text Tagger Whitelist** component to any existing value in the target field.

For additional information, see [Overwriting and appending target values on page 127](#).

- In the **Multi-assign Delimiter** field, specify the character to use to separate tags if you write multiple tags to the output field specified in the Target Field property. See also [Multi-assign delimiter on page 207](#).

## Text Tagger Regex edges

The **Text Tagger Regex** component uses a basic input edge unless the source component requires a different edge.

Be sure to define the output field in the edge from the **Text Tagger Regex** component to the next component to ensure that the data for this field is passed on for further processing.

## Using Text Enrichment

The **Text Enrichment** component provides the ability to extract and assess free-form text data.

Extracted information includes:

- Entities (such as people, places, organizations)
- Quotations
- Themes

The **Text Enrichment** component uses the Saliency Engine from Lexalytics. Depending on your license, the Saliency Engine may also provide the ability to assess the sentiment of the input text. Sentiment can be evaluated for the whole input (or document), the sentiment towards specific entities, or the sentiment towards specific themes.

## Supported Text Enrichment features

The Saliency Engine supports a wide variety of text extraction features, but only a limited set of these features are supported by the Endeca **Text Enrichment** component. The following table lists the text extraction features supported by the Endeca **Text Enrichment** component.

Table 8.1: Supported Text Enrichment features

Text Enrichment feature	Resulting information in the output record
Sentiment Analysis	An overall sentiment score for the current document, for specific entities, or for specific themes. This functionality is available by special license.  This feature can be enabled and disabled.

Text Enrichment feature	Resulting information in the output record
Named Entities	<p>A list of named entities in the current document. You can specify which types of entities to extract. Supported entity types include:</p> <ul style="list-style-type: none"> <li>• Company (i.e., businesses)</li> <li>• Person</li> <li>• Place (i.e., geographical locations)</li> <li>• Product</li> <li>• Sports</li> <li>• Title</li> <li>• List (for user-defined entities)</li> </ul> <p>The output record includes one column per type. Each column can contain multiple values.</p> <p>If Sentiment Analysis is enabled, the entities are added to different groups based on their sentiment scores. You must specify the ranges for the entity sentiment scores. The output record includes one column per range and each column can contain multiple values.</p> <p>This feature can be enabled or disabled.</p>
Themes	<p>A list of themes in the document. All meta-themes are added to the output record in a field you specify.</p> <p>For any theme that is not a meta-theme, if the theme score is higher than a user-specified threshold, then:</p> <ul style="list-style-type: none"> <li>• If Sentiment Analysis is enabled, the theme is added to a group based on its sentiment score. You must specify the ranges for the sentiment scores. The output record includes one column per range and each column can contain multiple values.</li> <li>• Regardless of whether Sentiment Analysis is enabled or disabled, the theme is added to another (i.e., not meta theme) user-specified field.</li> </ul> <p>This feature can be enabled or disabled.</p>
Quotations	<p>A list of quotes in the document, with an attribution to the speaker. You can specify the maximum length of quotes and the name of the field/property in the output record.</p> <p>This feature can be enabled or disabled,</p>
Document Summary	<p>A shortened version of the input content that best represents the whole content in a limited number of words.</p> <p>This feature is always enabled. It cannot be disabled.</p>

## Lexalytics information sources

The Lexalytics Support Web site provides two sources of information on the Saliency Engine:

- The Documentation Wiki: <http://dev.lexalytics.com/wiki/pmwiki.php>
- The Developer Blog: <http://dev.lexalytics.com/blog>

Although both sources are aimed at a developer audience, they can provide useful information for Integrator ETL users who are implementing the **Text Enrichment** feature.

[Text Enrichment prerequisites](#)

[Text Enrichment properties file](#)

[Query topics definition file](#)

[Adding the Text Enrichment component to a graph](#)

[Text Enrichment component edges](#)

[Processing text formatted in all caps](#)

[Normalizing themes](#)

[Adding foreign language processing to Text Enrichment](#)

[Adding Social Media processing to Text Enrichment](#)

## Text Enrichment prerequisites

The **Text Enrichment** component requires the Saliency Engine and a properties file, in addition to the input source text to process. If you want to use the query topics feature, you also need a query topics properties file; if you want to use normalized themes, you need a `normalization.dat` file.

## Saliency Engine

The **Text Enrichment** component requires installation of the Saliency Engine on the same machine as Integrator ETL. For details on installing the Saliency Engine, see the *Oracle Endeca Text Enrichment Installation Guide*.

When installing the Saliency Engine, write down the path to the Saliency Engine `data` directory. You must specify this path when configuring the **Text Enrichment** component.

## Source Input

The source input is the text to be processed by the Saliency Engine. You can use any supported input source, including files, database columns, and IAS record store data.

Input text must end sentences appropriately with appropriate punctuation (usually a period, but question marks or exclamation points if appropriate), and must be separated by spaces. If sentences do not end correctly and are not spaced correctly, themes will not be extracted correctly.

If the input text is formatted in all upper case, use the `setFlattenAllUpperCase` property. For details, see [Processing text formatted in all caps on page 148](#).

## Text Enrichment properties file

The Text Enrichment Properties file defines the configuration of the Salience Engine for the **Text Enrichment** component instance. All instances of the component can use the same properties file, or you can use different properties files to support different instances of the component.

The Text Enrichment properties file:

- specifies whether supported text extraction features are enabled
- specifies the input fields to process
- defines scoring thresholds for assessments
- defines the field names for assessment output data

The spelling and case of the configuration properties must match the spelling and case listed in the following sections.

The following sections and tables describe the configuration properties. Some of the setting values are **user-variable** (in other words, the user can customize the name or setting), while others must use the specific values described in the table.

### Global Sentiment Analysis property

The following property provides overall control of sentiment analysis:

```
te.sentiment.analysis.enabled
```

If this property is set to `true`, all levels of sentiment analysis (document, entity, and theme) are enabled. Each level can then be enabled and disabled individually.

If you want to use this feature, you must purchase a license that includes sentiment analysis.

Table 8.2: Document Sentiment Analysis properties

Document Sentiment property	Meaning
<code>te.document-sentiment.enabled</code>	If set to <code>true</code> (the default), Sentiment Analysis is enabled for documents and an overall sentiment score is computed for the current document. If set to <code>false</code> , Document Sentiment Analysis is disabled.
<code>te.document-sentiment.field</code>	Sets the target field name in the output records in which the sentiment score is written. The field name is user-variable, and <code>DocumentSentiment</code> is the default.
<code>te.document-sentiment.use-chains</code>	If set to <code>true</code> (the default), document sentiment scoring will use lexical chains in the computation of the sentiment score. The default is <code>true</code> .

Table 8.3: Named Entity processing properties

Named Entity property	Meaning
<code>te.entity.enabled</code>	If set to <code>true</code> (the default), Named Entity Extraction is enabled. If set to <code>false</code> , Named Entity Extraction is disabled.
<code>te.entity.types</code>	<p>Sets the types of named entities to extract. Supported types are:</p> <ul style="list-style-type: none"> <li>• Company</li> <li>• Person</li> <li>• Place</li> <li>• Product</li> <li>• Sports</li> <li>• Title</li> <li>• List (must be used if you have user-defined entities)</li> </ul> <p>The default types are <code>Person</code>, <code>Company</code>, and <code>Product</code>.</p> <p>Each configured entity type is written to a target field whose name is made up of the entity type (such as <code>Person</code>) prefixed by the <code>te.entity-sentiment.field</code> value.</p>
<code>te.entity.field-prefix</code>	<p>Sets the prefix name that is used to determine the final field names for the named entities. The field name is user-variable, and <code>Entities</code> is the default. For example, if you set this value to <b>Entities</b> and you configure <code>Person</code> and <code>Company</code> entity types to be extracted, then <b>EntitiesPerson</b> and <b>EntitiesCompany</b> will be the two target field names in the output records.</p> <p>If you have user-defined entities, then all the user-defined entities are put in the <b>EntitiesList</b> target field.</p>
<code>te.entity-sentiment.enabled</code>	If set to <code>true</code> (the default), Sentiment Analysis is enabled for entities and a sentiment score is computed for the entities. If set to <code>false</code> , Entity Sentiment Analysis is disabled.
<code>te.entity-sentiment.cut1.label</code>	<p>Sets the value for this <code>cut1</code> label. This will be the column name for the negative entities to be extracted. The field name is user-variable, and <code>EntitiesNegative</code> is the default.</p> <p>Negative entities are entities with a score less than the negative threshold.</p>
<code>te.entity-sentiment.cut1.value</code>	<p>Sets the <code>EntitiesNegative</code> threshold. The value is user-variable (for example, a value of <code>-0.1</code> can be used). Entity-sentiment scores that are less than this value are written to the <code>EntitiesNegative</code> record field.</p>



Named Entity property	Meaning
<code>te.entity-sentiment.cut2.label</code>	Sets the value for this <code>cut2</code> label. This will be the column name for the neutral type of entity sentiments to be extracted. Neutral entities are entities with a sentiment score between the negative and positive thresholds. The field name is user-variable, and <code>EntitiesNeutral</code> is the default.
<code>te.entity-sentiment.cut2.value</code>	Sets the <code>EntitiesPositive</code> threshold. The value is user-variable (for example, a value of <code>0.1</code> can be used). Entity-sentiment scores that are greater than this value are written to the <code>EntitiesPositive</code> record field.
<code>te.entity-sentiment.cut3.label</code>	Sets the value for this <code>cut3</code> label. This is the column name for the positive type of entity sentiments to be extracted. Positive entities are entities with a score greater than the positive threshold. The field name is user-variable, and <code>EntitiesPositive</code> is the default.

Table 8.4: Theme processing properties

Theme property	Meaning
<code>te.theme.enabled</code>	If set to <code>true</code> (the default), Theme Extraction is enabled. If set to <code>false</code> , Theme Extraction is disabled.
<code>te.theme-type.enabled</code>	<p>Specifies whether output themes will be standard or normalized.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>• <code>standard</code></li> <li>• <code>normalized</code></li> </ul> <p>If you specify <code>normalized</code> as the value for this property, you must define a <code>normalization.dat</code> file in the directory <code>%LEXALYTICS_HOME%/data/themes</code>. For more information, see <a href="#">Normalizing themes on page 148</a>. If the file <code>normalization.dat</code> does not exist, standard themes will be output.</p> <p>If the text enrichment properties file does not include this property, standard themes are output. If the value of the property is specified incorrectly, the graph will fail.</p>
<code>te.theme.field</code>	Sets the target field name in the output records in which kept theme names are written. The field name is user-variable, and <code>Themes</code> is the default. Kept themes are those themes whose score is higher than the <code>te.theme.score.threshold</code> setting and have made the <code>te.theme.keep-max</code> cut-off list.

Theme property	Meaning
<code>te.theme.score.threshold</code>	Sets a score threshold for keeping themes. That is, only keep themes with a score greater than this threshold. The value is user-variable, and <code>1.0</code> is the default.
<code>te.theme.keep-max</code>	Sets a threshold for keeping the best themes. That is, of those themes that are above the <code>te.theme.score.threshold</code> setting, only keep the themes with the best scores. The value is user-variable, and the default is <code>100</code> .
<code>te.theme-sentiment.enabled</code>	If set to <code>true</code> (the default), Sentiment Analysis is enabled for themes and a sentiment score is computed for the themes. If set to <code>false</code> , Theme Sentiment Analysis is disabled.
<code>te.theme-sentiment.cut1.label</code>	Sets the value for this <code>cut1</code> label. The field name is user-variable, and <code>ThemesNegative</code> is the default. Negative themes are themes with a score less than the negative threshold.
<code>te.theme-sentiment.cut1.value</code>	Sets the <code>ThemesNegative</code> threshold. The value is user-variable (for example, a value of <code>-0.1</code> can be used).
<code>te.theme-sentiment.cut2.label</code>	Sets the value for the <code>cut2</code> label. The field name is user-variable, and <code>ThemesNeutral</code> is the default. Neutral themes are themes with a sentiment score between the negative and positive thresholds.
<code>te.theme-sentiment.cut2.value</code>	Sets the <code>ThemesPositive</code> threshold. The value is user-variable (for example, a value of <code>0.1</code> can be used). <code>ThemesPositive</code> are themes with a score greater than the positive threshold.
<code>te.theme-sentiment.cut3.label</code>	Sets the value for this <code>cut3</code> label. The field name is user-variable, and <code>ThemesPositive</code> is the default. Positive themes are themes with a score greater than the positive threshold.
<code>te.meta-theme.field</code>	Sets the target field name in the output records in which the meta-themes are written. The field name is user-variable, and <code>ThemesMeta</code> is the default. Meta-themes are a list of themes in the document.
<code>te.meta-theme.frequency.threshold</code>	Sets a score threshold for keeping meta-themes. That is, only keep meta-themes with a score greater than this threshold. The value is user-variable, and <code>1.0</code> is the default.

Table 8.5: Quotation processing properties

Quotation property	Meaning
<code>te.quotation.enabled</code>	If set to <code>true</code> (the default), Quoted Context Extraction is enabled. If set to <code>false</code> , Quoted Context Extraction is disabled.
<code>te.quotation.field</code>	Sets the target field name in the output records in which quoted content is written. The field name is user-variable, and the default is <code>Quotes</code> .
<code>te.quotation.max-length</code>	Sets the maximum length (in characters) of a quotation. The default length is 200. Note that if the quotation in the source field is longer than this setting, the source quotation is not written to the target field.

Table 8.6: Social Media property

Social Media property	Meaning
<code>te.short-content.enabled</code>	If set to <code>true</code> , the processing of Social Media (for example, Twitter data) is enabled. If set to <code>false</code> (the default), Social Media processing is disabled. Note that Social Media (short content) processing is only applicable to default data (which means English). If you are using a language data other than English, make sure to set this property to <code>false</code> .

Table 8.7: Document Summary properties

Document Summary property	Meaning
<code>te.summary.field</code>	Sets the column name in the output file in which the summarization of the input content is written. The field name is user-variable, and the default is <code>Summary</code> .
<code>te.summary.length</code>	Sets the document summary length in sentences. The default length is 3 sentences.

Table 8.8: Basic custom properties

Basic Custom properties	Meaning
<code>te.salience.userdataDirectory</code>	Takes an absolute path to a directory that contains a user-created data dictionary.

Basic Custom properties	Meaning
<code>te.sentiment.setSentimentDictionary</code>	Takes an absolute path to a user-created dictionary that will be used as the sentiment dictionary for the Saliency Engine (that is, this dictionary overrides the default Saliency sentiment dictionary).
<code>te.sentiment.addSentimentDictionary</code>	Takes an absolute path to a user-created dictionary that will be used in addition to the current sentiment Analysis dictionary: <ul style="list-style-type: none"> <li>If <code>te.sentiment.setSentimentDictionary</code> has been used, then the additional dictionary is added to the first user-created sentiment dictionary.</li> <li>If <code>te.sentiment.setSentimentDictionary</code> has not been used, then the additional dictionary is added to the default Saliency sentiment dictionary.</li> </ul>

Table 8.9: Query topic processing properties

Query Topics property	Meaning
<code>te.query-topics.enabled</code>	If set to <code>true</code> (the default), query topic processing is enabled. If set to <code>false</code> , query topic processing is disabled.
<code>te.query-topics.field</code>	Sets the target field name in the output records to which specified query topics are written. The field name is user-variable, and <code>QueryTopics</code> is the default.
<code>Saliency.Options.QueryTopics.setQueryTopicList</code>	Specifies the location and name of the file used to define the topics and queries you want to use to tag output from this instance of the Text Enrichment component.
<code>te.query-topics-sentiment.enabled</code>	If set to <code>true</code> (the default), Sentiment Analysis is enabled for query topics and a sentiment score is computed for the topics. If set to <code>false</code> , Query Topic Sentiment Analysis is disabled.
<code>te.query-topic-sentiment.cut1.label</code>	Sets the value for this <code>cut1</code> label. The field name is user-variable, and <code>QueryTopicsNegative</code> is the default. Negative query topics are query topics with a score less than the negative threshold.
<code>te.query-topic-sentiment.cut1.value</code>	Sets the <code>QueryTopicsNegative</code> threshold. The value is user-variable (for example, a value of <code>-0.1</code> can be used).
<code>te.query-topic-sentiment.cut2.label</code>	Sets the value for the <code>cut2</code> label. The field name is user-variable, and <code>QueryTopicsNeutral</code> is the default. Neutral query topics are query topics with a sentiment score between the negative and positive thresholds.

Query Topics property	Meaning
<code>te.query-topic-sentiment.cut2.value</code>	Sets the <code>QueryTopicsPositive</code> threshold. The value is user-variable (for example, a value of 0.1 can be used). Positive query topics are query topics with a score greater than the positive threshold.
<code>te.query-topic-sentiment.cut3.label</code>	Sets the value for this <code>cut3</code> label. The field name is user-variable, and <code>QueryTopicsPositive</code> is the default. Positive query topics are themes with a score greater than the positive threshold.

## Advanced Custom options

In addition to the `te.*` configuration properties listed above, you can set other options provided by the Saliency API. You can use custom options from the following classes:

```
Saliency.Options.Base.xxx
Saliency.Options.Collections.xxx
Saliency.Options.Concepts.xxx
Saliency.Options.Entities.xxx
Saliency.Options.QueryTopics.xxx
Saliency.Options.Sentiment.xxx
```

where `xxx` is the name of the specific method you want to configure, such as `Saliency.Options.Base.setFailLongSentence`.

Information on these classes is available in the Lexalytics Saliency 5.1 Javadoc:

<http://dev.lexalytics.com/doc/java-se5.1/>

These API extension points are not parsed by the **Text Extraction** component. The values are passed directly to the Saliency Engine as is.

## Sentiment activator interaction

Four configuration activation properties control Sentiment Analysis:

- `te.sentiment-analysis.enabled`  
Enables or disables Sentiment Analysis on a global basis.
- `te.document-sentiment.enabled`  
Enables or disables Document Sentiment Analysis.
- `te.entity-sentiment.enabled`  
Enables or disables Entity Sentiment Analysis.
- `te.theme-sentiment.enabled`  
Enables or disables Theme Sentiment Analysis.

If `te.sentiment-analysis.enabled` is set to `false`, Sentiment Analysis is disabled globally. The document, entity, and theme sentiment activators are all treated as `false`, regardless of the specific setting of the individual activators. No sentiment analysis of any type is performed.

If `te.sentiment-analysis.enabled` is set to `true`, you can enable and disabled document, entity, and theme sentiment analysis in any combination. For example, if you are not interested in entity sentiment analysis, you can disable it but enable document and theme sentiment analysis.

## Customizing the theme, entity, and query topic sentiment cuts

If you are using Sentiment Analysis for themes, entities, and query topics, you can customize the number of cuts. The "Named Entity property", "Theme property" and "Query topic" tables above assume that you are using three cuts for positive, negative, and neutral scores, but you can use more or fewer cuts.

For example, named-entities are added to different user-configured fields based on their sentiment scores. You can configure the various output fields by specifying range-thresholds and field-names as follows (names in bold-face are user-supplied names):

```
te.entity-sentiment.cut1.label = fieldName1
te.entity-sentiment.cut1.value = sentimentScore1
te.entity-sentiment.cut2.label = fieldName2
te.entity-sentiment.cut2.value = sentimentScore2
te.entity-sentiment.cut3.label = fieldName3
te.entity-sentiment.cut3.value = sentimentScore3
...
te.entity-sentiment.cut-1.label = fieldNameN-1
te.entity-sentiment.cut-1.value = sentimentScoreN-1
te.entity-sentiment.cutN.label = fieldNameN
```

This field schema can be represented graphically by this illustration:



The above configuration specifies **N** different fields into which the named-entities will be mapped based on their sentiment-scores. Any entity whose sentiment-score is between `MIN_FLOAT` and **sentimentScore1** will be placed in **fieldName1**. Then, any entity whose sentiment-score is between **sentimentScore1** and **sentimentScore2** will be placed in **fieldName2**, and so on. Finally, any entity whose sentiment score is between **sentimentScoreN-1** and `MAX_FLOAT` will be placed in **fieldNameN**.

The label can be any string that is allowed to be a field-name (e.g., `EntitiesBucket1`). The value can be any floating-point number.



**Note:** There are no default values for the above-mentioned properties in the Text Enrichment component. Therefore, a property will not be used unless you add it to the properties file, with a named label and a floating-point value.

The following is an example configuration:

```
te.entity-sentiment.cut1.label = EntitiesNegative
te.entity-sentiment.cut1.value = -0.1
te.entity-sentiment.cut2.label = EntitiesNeutral
te.entity-sentiment.cut2.value = 0.1
te.entity-sentiment.cut3.label = EntitiesPositive
```

Configure theme sentiment and query topic sentiment the same way. The only difference is the name of the fields used in the configuration.

## Sample Text Enrichment properties file

```
# Enable Sentiment Analysis on global basis
te.sentiment-analysis.enabled = true

# Enable Document Sentiment
te.document-sentiment.enabled = true
te.document-sentiment.field = DocumentSentiment

# Enable Entity extraction
te.entity.enabled = true
# Entity types to allow and their prefix
te.entity.types = Person, Company, Product, Place
te.entity.field-prefix = Entities
# Entity sentiment goes -0.1 < s < 0.1
te.entity-sentiment.enabled = true
te.entity-sentiment.cut1.label = EntitiesNegative
te.entity-sentiment.cut1.value = -0.1
te.entity-sentiment.cut2.label = EntitiesNeutral
te.entity-sentiment.cut2.value = 0.1
te.entity-sentiment.cut3.label = EntitiesPositive

# Enable Theme extraction
te.theme.enabled = true
te.theme.field = Themes
# Only keep themes with score greater than the threshold
te.theme.score.threshold = 0.0
# Of those that are above the threshold, only keep the best 50
te.theme.keep-max = 50

# Theme sentiment goes -0.1 &lt; s &lt; 0.1
te.theme-sentiment.cut1.label = ThemesNegative
te.theme-sentiment.cut1.value = -0.1
te.theme-sentiment.cut2.label = ThemesNeutral
te.theme-sentiment.cut2.value = 0.1
te.theme-sentiment.cut3.label = ThemesPositive
# Set meta-theme field and only keep those above 0.1
te.meta-theme.field = ThemesMeta
te.meta-theme.frequency.threshold = 0.1

# Enable Quotation extraction
te.quotation.enabled = true
te.quotation.field = Quotes
# Max length of a quotation, in characters
te.quotation.max-length = 400

#Enable query topic processing
te.query-topics.enabled = true
te.query-topics.field = QueryTopics
#Set the location of the query topics definition file
Salience.Options.QueryTopics.setQueryTopicList = /localdisk/djones/lexalytics/salience-6.0/custom
/QueryDefinedTopics.dat
te.query-topics-sentiment.enabled = true
te.query-topics-sentiment.cut1.label = QueryTopicsNegative
te.query-topics-sentiment.cut1.value = -0.1
te.query-topics-sentiment.cut2.label = QueryTopicsEntitiesNeutral
te.query-topics-sentiment.cut2.value = 0.1
te.query-topics-sentiment.cut3.label = QueryTopicsPositive

#Enable Twitter processing
te.short-content.enabled = true

# Summary is always enabled
te.summary.field = Summary
# Document summary length in sentences
te.summary.length = 2

# Set location of my user directory
```

```
te.saliency.userdataDirectory=/localdisk/djones/lexalytics/saliency-6.0/data/user
# Add my sentiment dictionary to the Saliency default
te.sentiment.addSentimentDictionary=/localdisk/djones/lexalytics/saliency-6.0/custom/custom.hsd
```

## Query topics definition file

The query topics definition file defines the topics you want to use to tag your text, and the queries used to evaluate whether to add the topic to an Endeca record.

The query topics definition file is a simple text file consisting of one row for each topic you want to use. The format of each row is:

```
Topic<tab>queries
```

where

- `Topic` is the topic tag you want to add to the Endeca record.
- `queries` is the set of queries applied to the input text to determine whether the topic will be added to the output.

For example:

```
Business      "federal reserve" OR earnings OR partnership OR merger OR CEO OR CFO OR "chief
executive"
```

For details about the query grammar, see <http://dev.lexalytics.com/wiki/pmwiki.php?n=DataDir.QuerySyntax>.

## Sample query topics definition file

```
Business      "federal reserve" OR earnings OR partnership OR merger OR CEO OR CFO OR "chief executive"
Popular       Culture      celebrity OR paparazzi OR hollywood OR famous
Environment   environment OR ecology OR "climate change" OR biodiversity OR EPA
Health        health OR medicine OR healthcare OR disease OR immunization OR drugs
Sports        baseball OR football OR basketball OR golf OR hockey OR soccer
Politics      election OR ((senate OR congress OR representative ) AND (republican OR democrat)) OR
westminster  OR parliament
Religion      archbishop OR bishop OR pope OR "roman catholic" OR protestant OR catholic OR islam OR
hindu
Science       scientific OR "technological breakthrough" OR scientist OR researcher OR
"government lab"
Technology    internet OR "silicon valley" OR facebook OR twitter OR ipad OR iphone OR (droid NEAR
google) OR biotech OR "venture fund" OR techcrunch
```

## Adding the Text Enrichment component to a graph

Before adding a **Text Enrichment** component to a graph, be sure you have created a Text Enrichment properties file. Also be sure you know the location of the Saliency `data` directory.

When you add the **Text Enrichment** component to a graph, you must configure the following required properties:

- **Configuration file**

The absolute path to the Text Enrichment properties file (see [Text Enrichment properties files on page 135](#)) you want to use for this instance of the component. Note that you can implement different properties files to support different component configurations.

- **Input field**



The name of the field in the input source that contains the text you want to enrich. You can only enrich one field per instance of the component. If you want to enrich multiple fields in your records, you must use multiple instances of the **Text Enrichment** component in the graph.

- **Salience data path**

The absolute path to the Salience data directory.



**Note:** When you click in the **Value** field for any property that requires a path, Integrator ETL displays a ... browse button that you can use to browse the file system to find the location whose path you want to use.

You can also enter the following optional properties:

- **Error-handling key field**

Name of the field that stores the record key you want to use in error output. When processing of a record fails, Integrator ETL logs the failure and uses the value of the field specified in this key to identify the record. If you do not specify a value for this property, the record spec (primary key) field is used. This property is especially useful if you generate a key by concatenating the value of multiple fields; you can specify another easily-identifiable field that stores a unique value. Oracle recommends specifying a value for this property. Use the record spec (primary key) if no other option is available.

- **Text threshold (percent)**

The minimum percentage of alphanumeric characters that input field can contain to be processed. If you do not specify a value for this property, Integrator ETL uses a default of 80 (in other words, the text is only submitted to the Salience Engine for enrichment only if at least 80 percent of the characters in the field are alphanumeric).

- **Number of threads**

The number of processing threads the component should use. If no value is specified for this property, Integrator ETL uses a default of 1 (in other words, the Text Enrichment component uses only one thread for processing).

- **Multi-assign delimiter**

The character used to separate multiple values in a data fields. The default is the Unicode DELETE character (U007F).

## Text Enrichment component edges

The **Text Enrichment** component only requires a basic edge for input. The output edge, however, must include all the fields from the input, plus all fields added by the **Text Enrichment** component.

The specific fields on the output edge depend on the configuration of the Salience Engine defined in the Text Enrichment properties file for the component instance. Thus, the edge metadata includes some combination of the following properties:

- Document summary
- Quotations
- Entities, which may include:
  - Persons

- Companies
- Places
- Products
- Sports
- Titles
- User-defined entities
- Document Sentiment
- Themes
- Meta Themes (list of themes in the document)
- Theme cuts  
One field per cut. For example, if you have three cuts, you need three Theme cut fields; if you have five Theme cuts, you need five cut fields.
- Entity cuts  
One field per cut, as in the Theme cuts example.
- Query Topics
- Query topic cuts  
One field per cut. For example, if you have three cuts, you need three Query Topic cut fields; if you have five cuts, you need five cut fields.

If using the example Text Enrichment properties file, you must create the following properties for the edge metadata:

- DocumentSentiment
- SalesOrderNumber
- EntitiesPerson
- EntitiesProduct
- EntitiesCompany
- ThemesMeta
- ThemesNegative
- ThemesNeutral
- ThemesPositive
- EntitiesNegative
- EntitiesNeutral
- EntitiesPositive
- SurveyResponses
- Summary
- Quotes

- QueryTopics
- QueryTopicsNegative
- QueryTopicsNeutral
- QueryTopicsPositive

For details on the Text Enrichment properties file, see [Text Enrichment properties file on page 135](#).

Note that field type validation is performed for the Text Enrichment component. It will validate the field type and report an error if a wrong field type is detected.

## Creating metadata for an example Text Enrichment edge

This example assumes you have already created an edge to join the **Text Enrichment** component to the next component in the graph.

The following example illustrates the creation of edge metadata based on the example file provided in [Text Enrichment properties file on page 135](#). In this example, we will assume that the expected input is a .csv file (hence, the delimiter is a comma). We will name the record metadata "EnrichedText".

To create edge metadata for the example:

1. Right-click on the edge and from the popup menu choose **New metadata>User defined**.  
Integrator ETL displays the Metadata editor.
2. In the **Record:recordName1** field:
  - (a) Change the default name (**recordName1**) to **EnrichedText**.
  - (b) The value of the **Type** field defaults to `delimited`. Leave this value.
  - (c) In the **Delimiter** field, choose the comma character.
3. In **field1**:
  - (a) Change the **Name** to `DocumentSentiment`.
  - (b) Leave the **Type** as `string`.
4. To add a new field:
  - (a) Click the + (plus sign)  
Integrator ETL adds a new field to the list of fields. The new field is named **field $n$** , where  $n$  is the next number in the list of fields.
  - (b) Change the **Name** to `SalesOrderNumber`. In this case, leave the **Type** as `string`.
5. Repeat Step 4 for each field you need to add to the metadata. Note that **Type** of sentiment fields should be `decimal`.
6. Click **OK** to save your changes.

## Processing text formatted in all caps

To ensure correct processing of text formatted in all caps, use the `setFlattenAllUpperCase` property.

If the input text is formatted using all upper case, text enrichment processing may be inconsistent. To ensure consistent processing of this text, add the `setFlattenAllUpperCase` property to your text enrichment properties file:

```
Salience.Options.Base.setFlattenAllUpperCase = true
```

## Normalizing themes

You can normalize a number of discovered themes into a single reported theme.

For example, suppose theme processing discovers iPhones, iPads, Androids, and Blackberries. Rather than reporting separate themes for each device, you would prefer to output one theme: "smart device". Normalizing themes provides this capability.

You must create a text file named `normalization.dat` and store it in the directory `%LEXALYTICS_HOME%/data/themes`. The format of this file is

```
<stemmed_theme>[tab]<normalized_theme>
```

For example, to implement the normalization case for smart devices described earlier, your `normalization.dat` file would include the following data:

```
iPhone[tab]smart device
iPad[tab]smartdevice
Android[tab]smart device
Blackberry[tab]smart device
```

In the text enrichment properties file, add the property `te.theme-type.enabled` with the value `normalized`. If you do not specify this property, stemmed themes will be output. If you specify the property incorrectly, the graph will fail.



**Note:** Any theme specified in the `normalization.dat` file will be normalized. Thus, you may see normalization occurring even in graphs where you have not specified `te.theme-type.enabled=normalized`. In this case, delete `normalization.dat` and use standard themes.

## Adding foreign language processing to Text Enrichment

The Salience Engine supports text enrichment in Chinese, French, German, Portuguese, and Spanish.

To add foreign language processing to Text Enrichment:

1. Download the Oracle Endeca Text Enrichment module from the Oracle Software Delivery Cloud.
2. For details on unzipping the language packages and installing the data libraries, see the *Oracle Endeca Text Enrichment Installation Guide*.
3. When configuring an instance of the **Text Enrichment** component that you want to process a supported foreign language, in the **Salience data path** property, specify the path to the foreign language directory for the language you want to use in processing.

For example, if you want to add Text Enrichment in German, specify the path to the `ge-data` directory, including the `ge-data` directory itself, such as `C:/Program Files (x86)/Lexalytics/ge_data`.

## Processing multiple languages

When processing text from multiple languages, add a separate instance of the Text Enrichment component for each language.

Processing for each language requires a unique data library. One instance of the Text Enrichment component cannot process multiple libraries. Therefore, you must add a unique instance of the Text Enrichment component for each language stream.

If each language input comes from a different input source, the stream from each source can be piped directly to its own instance of the Text Enrichment component. If a single input includes multiple languages, use Language Detection to determine the language of each input record, and filter each language stream to a separate instance of the Text Enrichment component.

## Adding Social Media processing to Text Enrichment

The Salience Engine supports enrichment of content derived from Social Media platforms, such as Twitter.

Twitter processing provides an alternative to the standard English Text Enrichment processing. This processing should only be used with content derived from Twitter feeds. Content from other sources should use the standard English processing.



**Note:** Social Media processing is only available in English. It is not available for other languages supported by Text Enrichment.

To add Social Media processing to Text Enrichment:

1. Make sure that the Salience Engine has been installed. The Social Media feature is part of the Salience Engine itself.
2. Set the new `te.short-content.enabled` property to `true` in the Text Enrichment properties file.

## Detecting text language

Integrator ETL provides the ability to automatically detect the language of input text.

Use the Language Detector to evaluate text fields in input records to determine the language of the record.

The Language Detector uses Oracle Language Technology (OLT) to evaluate the input text and determine the language. The Language Detector can detect all languages supported by the Endeca Server. For details, see "Supported languages" in the *Oracle Endeca Server Developer's Guide*.

The component passes the content of the specified text field for each record to OLT for evaluation. You must install OLT to use the Language Detector. For details about installing OLT for the Language Detector, see "Installing Oracle Language Technology (OLT)" in the *Oracle Endeca Information Discovery Integrator ETL Installation Guide*. OLT evaluates and determines a set of scores for the text. The supported language with the highest score is reported as the language of the text. The component outputs the language code for this language in a field whose name you specify in the component configuration.

If the input text of the specified field does not match a language supported by the component, the component outputs "unknown". If the value of the specified field is null, or consists only of white spaces or non-alphabetic characters, the component outputs "non language characters".

Note that only one field can be evaluated per instance of the Language Detector component. If you want to evaluate multiple fields, you must add one instance of the Language Detector component for each field whose language you want to evaluate.

### *Language Detector edges*

## Language Detector edges

The **Language Detector** only requires a basic edge for input. The output edge, however, must include the output language field.

The name you specify for the output language field on the output edge must match the name you specify for the **Output language field** property in the component configuration.



## Chapter 9

# Component Reference

---

This chapter is a reference for the Oracle Endeca Information Discovery components available in the Integrator ETL Palette.

*Add KVPs component*

*Bulk Add/Replace Records component*

*Delete Records component*

*Export Config component*

*Import Config component*

*Language Detector component*

*Merge Managed Values component*

*Merge Records component*

*Modify Records component*

*Record Store Reader*

*Reset Data Domain component*

*Text Enrichment component*

*Text Tagger Regex component*

*Text Tagger Whitelist component*

*Transaction RunGraph component*

*Visual properties of components*

*Common configuration properties of components*

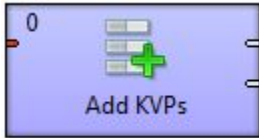
*Multi-assign delimiter*

*Batch size adjustments by components*

*Valid characters for ingest*

## Add KVPs component

The **Add KVPs** component updates records in the Endeca data domain by adding new key-value pairs to them.



Use the **Add KVPs** component to update records by adding new key-value pair (KVP) assignments to those records. If no Endeca record matches an input record, a new Endeca record is created. The component uses the Data Ingest Web Service (DIWS) to update records.

- You can only load new key-value pairs (KVPs) to a record with this component. You cannot delete or replace existing KVPs.
- You can only load standard attribute values. Adding managed attribute values is not supported.
- You cannot assign multiple values to a single property in a single input record. To assign multiple values to a property, you must add separate rows in the input file. If you attempt to assign multiple values in the same row, they are treated as a single value.
- If an assignment specifies a standard attribute (property) that does not exist in the data domain, the new standard attribute is created by DIWS with system default values for the Property Description Record (see [Standard attribute default values on page 38](#)). You can, however, specify a data type for the new standard attribute in the mdexType column of the input file.

The primary use case for this component is to load source data that is stored in a key-value pair format rather than a rectangular data model.



**Note:** String data submitted for ingest must consist of valid XML characters. For details see [Valid characters for ingest on page 208](#).

### Input metadata

The first row of the data source input file is the record header row and must use this schema:

```
specKey|specValue|kvpKey|kvpValue|mdexType
```

The following table provides details of these properties:

Schema property	Meaning
specKey	The name of the primary key (record spec) of the record to which the key-value pair will be added.
specValue	The value of the record's primary key.



Schema property	Meaning
kvpKey	The name (key) of the Endeca standard attribute to be added to the record. If the standard attribute does not exist in the data domain, it is automatically created with system default values. For the default values, see <a href="#">Standard attribute default values on page 38</a> .
kvpValue	The value of the standard attribute to be added to the record.
mdexType	Specifies the mdex type (such as mdex:int or mdex:dateTime) for the kvpKey standard attribute. This parameter is intended for use when you want to create a new standard attribute and want to specify its property type. If a new PDR for the standard attribute is created and mdexType is not specified, then the type of the new standard attribute defaults to mdex:string. If the standard attribute already exists, you can specify an empty value for mdexType. For a list of valid data types, see <a href="#">Supported data types on page 36</a> .

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Add KVPs** component.

Table 9.1: Add KVPs component properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use localhost.	MyEndecaServer 255.255.255.0
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001

Name	Description	Valid Values	Example
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	/endeca-server
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	quickstart
Spec Attribute	Specifies the primary key (record spec) for the records on which the operation will be performed.	Name of the primary key.  If the primary key does not exist in the data domain, the property is created automatically with system default values.	FactSales_OrderNumber
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True)  Unchecked (False)	
Batch Size	Specifies the batch size (in bytes) for the ingest operation. A batch consists of one or more complete records.  See also <a href="#">Batch size adjustments by connectors on page 208</a> .	A positive integer equal to or greater than 1 defines the batch size. If the batch size is too small to fit the last record in the batch, the size is reset to accommodate that record. The batch size then returns to the specified batch size.  Specifying zero (0) or a negative integer turns off batching. When batching is turned off, records are submitted to the data domain one at a time.	1000000  0

Name	Description	Valid Values	Example
Maximum number of failed batches	Sets the maximum number of batches that can fail before the ingest operation is ended.	Either a 0 (allows no failed batches) or a positive integer.	15

## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.
- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.2: Port 0 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Number of Records affected	Long	Total number of records successfully ingested	99999
Time Taken in Seconds	Numeric	Total time to process the batch, in seconds	127

Table 9.3: Port 1 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Fault Message	String	Error message returned by the Endeca Server	

## Bulk Add/Replace Records component

The **Bulk Add/Replace Records** component adds new records or replaces existing records in an Endeca data domain.



The **Bulk Add/Replace Records** component uses the Endeca Server's Bulk Load Interface. (Other components use the Data Ingest Web Service [DIWS]). The Bulk Load Interface defines the basic characteristics of this component:

- The component can load data source records only.  
Thus, you cannot use this component to load
  - Property Description Records (PDRs)
  - Dimension Description Records (DDRs)
  - Managed attribute values (MVals)
  - Global Configuration Record (GCR)
  - Data domain configuration documents
- Existing records in the Endeca data domain are replaced, not updated. In other words, the load operation is a replace operation not an append operation.
- A primary-key attribute (also called a record spec) is required for each record to be added or replaced.
- If an assignment (key-value pair) specifies a standard attribute (property) that does not exist in the Endeca data domain, a new standard attribute is automatically created with system default values for the PDR (see [Standard attribute default values on page 38](#)).
- The component does not send records in batches. It employs a single streaming connection to the data domain.



**Note:** String data submitted for ingest must consist of valid XML characters. For details see [Valid characters for ingest on page 208](#).

### Post ingest behavior

The default behavior of the Bulk Load Interface is to force a merge to a single generation at the end of every bulk-load ingest operation. This behavior is intended to maximize query performance at the end of a single, large, homogenous data update that would occur during a regularly scheduled update window.

The **Post Ingest Query Optimization** property of the **Bulk Add/Replace Records** allows you to control when the post-ingest merge occurs:

- If this property is set to `true` (the default), the merge is forced immediately after ingest.
- If the property is set to `false`, a merge is not forced at the end of an update, but instead relies on the regular background merge process to keep the generations in order over time. This behavior is more suitable for parallel heterogeneous data updates where low overall update latency is paramount.

The **Post Ingest Dictionary Update** property controls when the Aspell spelling dictionary is updated:

- If this property is set to `true` (the default), a dictionary update is forced immediately after the ingest.
- If this property is set to `false`, dictionary update is disabled. However, you can later manually update the dictionary using the `updateSpellingDictionaries` operation of the Data Ingest Web Service. For details, see the *Oracle Endeca Server Data Loading Guide*.

## Input metadata schema

The input metadata schema for the **Bulk Add/Replace Records** component is not fixed. Each metadata field represents a property on a data domain record.

The metadata type of the Integrator ETL field (as shown in the Edit Metadata dialog on the edge connecting to the connector) translates to the `mdex` property type. For example, the Integrator ETL `integer` data type translates to the `mdex:int` data type. Note that you must override this behavior to support Integrator ETL non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`). For details, see [Creating mdexType Custom properties on page 39](#).

Multi-value input fields can be defined using either the multi-assign delimiter or a list data type. See [Processing multi-value data on page 49](#) for additional details.

## Use cases

Use the **Bulk Add/Replace Records** component to load data in bulk when it is acceptable to delay the visibility of the updates and for query performances to stop while data is loaded.

Use this component for the following cases:

- Full index initial load of records when no schema has been loaded. In this scenario, the Endeca data domain has no user data records and also has no user-created schema (in other words, no existing PDRs). In this case, all new properties (including the primary-key properties) are created with system default values.
- Full index initial load of records, after you have loaded the record schema.
- Adding more new records to the Endeca data domain any time after the initial loading of records. As in the initial load case, new standard attributes that do not exist in the data domain are automatically created with default system values.
- Replacing existing records in the Endeca data domain any time after the initial loading of records. In this case, all the key/value pairs of the existing record are replaced with the key-value pairs of the input file.

## Configuration properties




**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Bulk Add/Replace Records** component.

Table 9.4: Bulk Add/Replace Records properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>

Name	Description	Valid Values	Example
Spec Attribute	Specifies the primary key (record spec) for the records that will be uploaded.	<p>Name of the primary key attribute.</p> <p>If the primary key does not exist in the data domain, the property is created automatically with system default values.</p> <p> <b>Note:</b> When specifying the Spec Attribute in the <b>Bulk Add/Replace Records</b> component, you should not prepend the collection key. When you check the <b>Prefix Attributes With Collection Key</b> box, the collection key will be prepended automatically. If you prepend the collection key when specifying the Spec Attribute, it will be prepended twice.</p>	FactSales_OrderNumber

Name	Description	Valid Values	Example
Collection key	<p>Required. Specifies the key of the collection to which the data should be loaded.</p> <ul style="list-style-type: none"> <li>• If the Collection does not already exist, it will be created.</li> <li>• If the Collection already exists, any modifications will be added to the specified Collection. In other words, new records will be added to the specified collection, and records will be updated in the specified collection.</li> </ul> <p>Note that if you specify an existing collection, the value in the <b>Spec attribute</b> field must match the spec attribute of the existing collection.</p>	Alphanumeric characters, but must begin with either a letter or an underscore.	New Collection
Prefix Attributes With Collection Key	<p>Specifies whether to prepend attribute names with the name of the collection when uploading data to the data domain. The underscore character is used as a separator between the collection key and the attribute name: "collectionkey_attributename".</p>	<p>Checked (True; the collection key will be prepended to attribute names when uploading data to the data domain.)</p> <p>Unchecked (false; the attribute name is not modified when uploading data to the data domain; default.)</p>	



Name	Description	Valid Values	Example
Post Ingest Query Optimization	Specifies whether to merge records immediately after the ingest operation is complete or to use the standard background merge process.	Checked (True; default) Unchecked (False)	
Post Ingest Dictionary Update	Specifies whether to update the spelling dictionary automatically immediately after the ingest operation is complete. If the dictionary is not updated when the ingest operation is complete, you must issue the update command manually using web services.	Checked (True; default) Unchecked (False)	
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	
Stop after this many errors	Specifies the maximum number of ingest errors allowed in a single load operation. If this number of errors occurs, the ingest operation is terminated.	Either 0 (no errors are allowed) or a positive integer.	0 15

Name	Description	Valid Values	Example
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.  See also <a href="#">Multi-assign delimiter on page 207</a> .	A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.	
Timeout (ms)	Specifies the timeout of operations of the component.  If timeouts occur when running graphs, operations on the Endeca Server may be taking too long. Change the value of this parameter to allow more time for operations on the Endeca Server.  The default configures a one minute timeout.	Integers	60000

## Data domain status after a failed ingest operation

When a bulk load ingest operation is terminated because of an error, records that were ingested before the error should be included in the data domain. Although the data domain may accept queries on the ingested records, you should consider the data domain to be in an inconsistent state. To restore a consistent state, review the logs to determine the problems that caused the bulk load operation to fail, correct these problems, then reload the data.

## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.

- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.5: Port 0 metadata

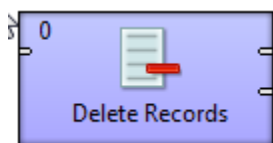
Port Field Name	Data Type	Description	Example
Records added	Long	Number of records added to the data domain	984341
Records Queued	Long	Number of records queued for processing but not processed	1568
Records Rejected	Long	Number of records submitted that were not added to the data domain	24836
State	String	Data domain status string returned by the Bulk Load API	

Table 9.6: Port 1 metadata

Port Field Name	Data Type	Description	Example
Fault Message	String	Error message returned by the Endeca Server	

## Delete Records component

The **Delete Records** component deletes specified Endeca records.



The **Delete Records** component uses the Data Ingest Web Service (DIWS) to delete complete records from the Endeca data domain. The component can delete multiple records in a single transaction. Two options are available to select the records to delete:

- You can input a data array that defines the assignments to use to select the records to delete.
- You can enter an Endeca Query Language (EQL) record specifier (`WHERE` clause) to select the records to delete.

These options are mutually exclusive. If you choose to use an EQL record specifier to select the records to delete, you cannot also use an input array.

## Using an input array to select records to delete

An input array is only necessary if you choose not to use an EQL query. You can also use an input array to support an EQL query. For details, see "Using EQL" below.

The metadata schema of the input array for the **Delete Records** component is an arbitrary array of property names and values that specify the records to delete. The first row of the input array is the record header row and defines the names of the properties to use to select the records to delete.

Attributes are joined using an AND operator. Thus, the fewer properties included in the input array, the more general the selection and the more records will be deleted. Conversely, the more properties included in the input array, the more specific the selection and the fewer records will be deleted.

For the examples below, assume the following schema has been defined in the data domain:

```
DimGeography_GeographyKey|DimGeography_City|DimGeography_StateProvinceCode|DimGeography_StateProvince
Name|DimGeography_CountryRegionCode|DimGeography_CountryRegionName|DimGeography_PostalCode
```

- If the following array is input, Integrator ETL deletes all records where the value of the `DimGeography_StateProvinceName` property is Alabama:

```
DimGeography_StateProvinceName
Alabama
```

- If the following array is input, Integrator ETL deletes records where the value of the `DimGeography_City` property is `Newton` and the value of the `DimGeography_StateProvinceName` is `British Columbia`

```
DimGeography_City|DimGeography_StateProvinceName
Newton|British Columbia
```

- If the following array is input, Integrator ETL deletes records where the value of the `DimGeography_City` property is `Newton` and the value of the `DimGeography_StateProvinceName` is `British Columbia` and the value of the `DimGeography_PostalCode` is `V2M1P1`

```
DimGeography_City|DimGeography_StateProvinceName|DimGeography_PostalCode
Newton|British Columbia|V2M1P1
```

Multi-value attributes can be specified using either a multi-assign delimiter or a list data type. See [Processing multi-value data on page 49](#) for additional details.

When using a multi-value attribute to select records, the matching behavior is defined by the value of the **Multi-assign specifier attribute behavior** property.

- If the value of the **Multi-assign specifier attribute behavior** property is `Matches exactly`, then Endeca records will only be selected if the value of the attribute on the record exactly matches the input value of the attribute. In other words, the value of the attribute on the Endeca record must include all values for the attribute on the input record, and only those values. If the value of the attribute on the Endeca record includes additional values beyond those on the input record, the Endeca record will not be selected.
- If the value of the **Multi-assign specifier attribute behavior** property is `Includes`, then Endeca records will be selected if the value of the attribute on the record includes one or more of the values on the input record. All values need not match, and the Endeca record can include additional values beyond those included on the input record.

## Using EQL to select records to delete

You can use an Endeca Query Language (EQL) record specifier (the `WHERE` clause of an EQL query) to select the records to delete. For example: `"DimGeography_City"='Newton' AND "DimGeography_StateProvinceName"='British Columbia'` selects records where the value of the

DimGeography\_City property is Newton and the value of the DimGeography\_StateProvinceName is British Columbia.



**Note:** Standard practice in EQL is to use double quotation marks around attribute names and single quotation marks around attribute values.

You can also use an input array to support an EQL record specifier. Use variables to specify the input properties. For example, if you input the following array:

```
City|StateProvince
Newton|British Columbia
Townsville|Queensland
Longmont|Colorado
```

You could enter the following EQL: "DimGeography\_City"=\$input.City and "DimGeography\_StateProvinceName"=\$input.StateProvince.

When selecting records based on the value of multi-assign properties, you use a different syntax depending on whether you want to select values based on the complete set values of the attribute or you want to select values based on the individual members of the set of values of the attribute.

If you want to select records based on the complete set of values of an attribute, use the syntax "attribute"={value1,'value2',value3,...'valuen'}. For example, if Color is a multi-assign attribute, and you want to select records where the value of the attribute is the set "red,green,blue", you would use the syntax "Color"={ 'red' , 'green' , 'blue' }. Note that the order of members of the set is not significant. In other words, the record specifier "Color"={ 'green' , 'blue' , 'red' } selects the same set of records as the record specifier "Color"={ 'red' , 'green' , 'blue' ).

The value of the **Multi-assign specifier attribute behavior** property determines the behavior when records include more or fewer values than the input set. If the value of this property is *Includes*, records are selected if they include any member of the input set and do not need to match the values in the input set exactly. If the value of the **Multi-assign specifier attribute behavior** is *Matches exactly*, records are only selected if the value of the attribute includes all members of the input set. Records that contain more or fewer values will not be selected. In other words, when using the record specifier "Color"={ 'red' , 'green' , 'blue' }, a record where the value of Color is "blue,red" would not be selected; a record where the value of Color is "red,green,blue,yellow" would also not be selected.

If you want to use an input array that includes multi-value fields, wrap the variable in curly braces: "Color"={\$input.Color}.

To select records based on individual members of the set the values of a multi-assign property, use the explicit qualification syntax. For example, if Color is a multi-assign attribute, the record specifier `SOME "value" IN "Color" SATISFIES( "value" = 'red' )` selects records where one of the members of the set of values of the Color attribute is 'red'.

For full details about Endeca Query Language, see the *Oracle Endeca Server EQL Guide*.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Delete Records** component.

Table 9.7: Delete Records component properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>
Record Set Specifier Attributes	Specifies the input attributes to use to select the records on which to operate	Attributes available from the input metadata	
Multi-assign specifier attribute behavior	Specifies how multi-value input attributes will be matched when selecting records.	<ul style="list-style-type: none"> <li>Matches exactly Records are selected only if the value of the attribute in the record includes all input values for the attribute.</li> <li>Includes Records are selected if the value of the attribute on the record matches any of the input values for the attribute.</li> </ul>	

Name	Description	Valid Values	Example
Truncate collection	Specifies whether to delete records from the Collection specified in the <b>Collection key</b> property.	Checked (True) Unchecked (false)	
Collection Key	Key of the Collection from which to delete records.	Valid Collection keys. If the specified Collection does not exist, the graph fails and returns an error.	NewCollection
Use EQL Record Set Specifier	Specifies whether to use an EQL expression to select the records on which to operate.  Standard EQL practice is to use double quotation marks around attribute names and single quotation marks around attribute values.	Unchecked (false) Checked (true)	"DimGeography_City" = 'Newton' AND "DimGeography_StateProvinceName" = 'British Columbia'
EQL Record Set Specifier	The EQL expression to use to select the records on which to operate.	A valid EQL expression	
Multi-assign specifier attribute behavior	Specifies how multi-value input attributes will be matched when selecting records.	<ul style="list-style-type: none"> <li>• Matches exactly Records are selected only if the value of the attribute in the record includes all input values for the attribute.</li> <li>• Includes Records are selected if the value of the attribute on the record matches any of the input values for the attribute.</li> </ul>	

Name	Description	Valid Values	Example
SSL Enabled	<p>Enables or disables SSL for the component.</p> <p>SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.</p>	<p>Checked (True)</p> <p>Unchecked (False)</p>	
Batch Size	<p>Specifies the batch size (in bytes) for the ingest operation. A batch consists of one or more complete records.</p> <p>See also <a href="#">Batch size adjustments by connectors on page 208</a>.</p>	<p>A positive integer equal to or greater than 1 defines the batch size. If the batch size is too small to fit the last record in the batch, the size is reset to accommodate that record. The batch size then returns to the specified batch size.</p> <p>Specifying zero (0) or a negative integer turns off batching. When batching is turned off, records are submitted to the data domain one at a time.</p>	<p>1000000</p> <p>0</p>
Multi-assign delimiter	<p>Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.</p> <p>See also <a href="#">Multi-assign delimiter on page 207</a>.</p>	<p>A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.</p>	
Maximum number of failed batches	<p>Sets the maximum number of batches that can fail before the ingest operation is ended.</p>	<p>Either a 0 (allows no failed batches) or a positive integer.</p>	<p>15</p>



## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.
- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.8: Port 0 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Number of Records Deleted	Long	Number of records deleted from the data domain as a whole	42683
Number of Records Affected	Long	Number of records from which key/value pairs (assignments) have been removed, while retaining the rest of the record	19834
Time Taken in Seconds	Numeric	Total time to process the batch, in seconds	127

Table 9.9: Port 1 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Fault Message	String	Error message returned by the Endeca Server	

## Export Config component

The **Export Config** component exports the schema and configuration stored in an Endeca data domain.



The **Export Config** component uses Configuration Web Service operations.

- The component exports the configuration and schema through an output port. You can connect this port to a Writer component to write the exported configuration and schema into a file. This file can be imported later to apply the configuration to a data domain.
- The component does not export your view definitions. To export view definitions, use the `listEntities` operation of the Entity Configuration Service. For details, see the *Oracle Endeca Sever Developer's Guide*.

### Use cases

The **Export Config** and **Import Config** connectors are intended to be used in the following cases:

- You want to save and re-apply modifications to a data domain configuration. These modifications may have been implemented using Integrator ETL or using Studio (for example changing attribute groups or attribute group names). Once you have made a change, you generally want to preserve it and re-apply it if you need to clear or update the data domain. Use these connectors to export a configuration and to re-apply the configuration later.
- You want to run a baseline updates to refresh and reload data. The typical scenario consists of the following graphs:
  1. A graph using the **Export Config** connector to export the configuration and schema of the data domain.
  2. A graph using the **Reset Data Domain** to remove all records and re-provision the Endeca data domain.
  3. A graph using the **Import Config** to import and restore the configuration to the data domain.
  4. A graph to reload the data records.
  5. A graph using the **Transaction RunGraph** component. This graph starts a transaction that runs the other graphs.

### Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Export Config** component.

Table 9.10: Export Config properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified. The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>
SSL Enabled	Enables or disables SSL for the component. SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	

## Import Config component

The **Import Config** imports a schema and configuration into an Endeca data domain.



The **Import Config** component uses Configuration Web Service operations.

- This component imports a schema and configuration that was previously exported to a file. Use a **UniversalDataReader** component to read the file that stores the previously exported configuration and

schema. Connect the output port of the **UniversalDataReader** to the input port on **Import Config** component.

- Only basic XML validation takes place. The **Import Config** component uses Configuration Web Service operations, so the imported configuration file must conform to the requirements of the Configuration Web Service WSDL document and must contain only valid XML to describe the configuration and schema.

## Use cases

The **Export Config** and **Import Config** connectors are intended to be used in the following cases:

- You want to save and re-apply modifications to a data domain configuration. These modifications may have been implemented using Integrator ETL or using Studio (for example changing attribute groups or attribute group names). Once you have made a change, you generally want to preserve it and re-apply it if you need to clear or update the data domain. Use these connectors to export a configuration and to re-apply the configuration later.
- You want to run a baseline updates to refresh and reload data. The typical scenario consists of the following graphs:
  1. A graph using the **Export Config** connector to export the configuration and schema of the data domain.
  2. A graph using the **Reset Data Domain** to remove all records and re-provision the Endeca data domain.
  3. A graph using the **Import Config** to import and restore the configuration to the data domain.
  4. A graph to reload the data records.
  5. A graph using the **Transaction RunGraph** component. This graph starts a transaction that runs the other graphs.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Import Config** component.

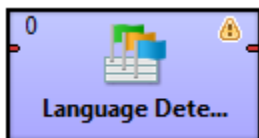
Table 9.11: Import Config properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use localhost.	MyEndecaServer 255.255.255.0

Name	Description	Valid Values	Example
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	/endeca-server
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	quickstart
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True)  Unchecked (False)	

## Language Detector component

Use the **Language Detector** component to automatically detect the language of long text strings.



The **Language Detector** component uses Oracle Language Technology (OLT) to evaluate the input strings and determine the language of the string. The component can detect the following languages supported by the Endeca Server. For details about supported languages, see "Supported languages" in the *Oracle Endeca Server Developer's Guide*.

When the string is evaluated, it is scored based on the linguistic information in the text. The language that receives the highest score is determined to be the language of the text, and the name of that language is output by the component. If the text cannot be determined to be a language supported by OLT, the component outputs "unknown language". If the input string consists of non-alphabetic characters, the component outputs "non-language characters". If an error occurs during string processing, the component outputs "error".

## Metadata schema

The metadata schema for the **Text Enrichment** component is not fixed.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Language Detector** component.

Table 9.12: Language Detector properties

Name	Description	Valid Values	Example
Input field	Name of the source field in the input source record whose text you want to evaluate to determine the language	Field names	survey_responses
Output language field	The name of the field to which you want to output the name of the detected language. The output metadata must include this field.	String	Language
Record key	The key of the record. Used to log errors. Can be null.		
Threads	Number of threads to use for language processing. Defaults to 8. A variety of factors affect processing performance. You may need to adjust the value of this property to achieve the optimal results for your system.	Integers	8

## Merge Managed Values component

The **Merge Managed Values** component loads managed attribute values (a taxonomy) into the Endeca data domain, or updates existing managed attribute values.



Use the **Merge Managed Values** component to maintain Endeca managed attribute values (the units of a taxonomy) in the Endeca data domain using the Configuration Web Service .

- The component maintains only managed values (mvals). It does not maintain standard values (svals).
- All the managed values must belong to only one managed attribute.
- If the managed attribute does not exist in the Endeca data domain, the managed attribute is created with system default values for the Dimension Description Record (DDR) as well as the Property Description Record (PDR), if it also does not exist. For a list of the default values, see [Default values for new attributes on page 38](#).
- The component also provides the option of creating synonyms for managed values.

### Input metadata

The input metadata schema of the **Merge Managed Values** component is fixed and uses a specific ordering. The first row of the data source input file is the record header row and must use this schema:

```
spec|displayname|parent|synonym|rank
```

where:

- *spec* is a unique string identifier for the managed value. This is the managed value spec.
- *displayname* is the name of the managed value.
- *parent* is the parent ID for this managed value, If this is a root managed value, use a forward slash (/) as the ID. If this is a child managed value, specify the unique ID of the parent managed value.
- *synonym* optionally defines the name of a synonym. Synonyms can be added to both root and child managed values. You can add multiple synonyms to a single managed value, with the synonyms separated by a delimiter that you specify in the configuration dialog.
- *rank* is the rank order of the managed attribute value in relation to other managed attribute values. Negative values are allowed. Ties in rank value are broken arbitrarily.

### Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Merge Managed Values** component.

Table 9.13: Merge Managed Values component properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>
Managed Attribute Name	Specifies the name of the managed attribute to which to add managed values.	The name of a managed attribute. The name must use the NCName format. In other words, the name should not include a namespace.  If the specified managed attribute does not exist in the Endeca data domain, the managed attribute is automatically created with system default values.	Category



Name	Description	Valid Values	Example
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	
Synonym Delimiter	Defines the delimiter used to specify multiple synonyms for the associated MVal.	A single character that can be used as a delimiter. The default is the Unicode DELETE characters (\U007F).	

## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.
- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.14: Port 0 metadata

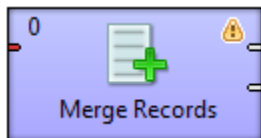
Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Number of Managed Attributes Added	Long	Number of new managed attributes added to the data domain	1
Number of Managed Values Added	Long	Number of values added to both new and existing managed attributes	19834
Time Taken in Seconds	Numeric	Total time to process the batch, in seconds	127

Table 9.15: Port 1 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Fault Message	String	Error message returned by the Endeca Server	

## Merge Records component

Use the **Merge Records** component to add complete new records to a data domain, or to add new assignments to existing records.



The **Merge Records** component uses the Data Ingest Web Service (DIWS) to add complete new records to an Endeca data domain, or to add new assignments to existing records in an Endeca data domain. Note that new assignments are appended to existing records, they do not overwrite existing records. If an input record matches an existing record, and any of the input properties match a property that is specified as single-assign, an error occurs and the record is not updated.

You can choose only to add new records rather than to update existing records. If any existing record matches on the spec attribute of the input record, an error occurs. The input record is not added and the existing record is not modified.

The **Merge Records** component operates on one Endeca record per input record.



**Note:** String data submitted for ingest must consist of valid XML characters. For details see [Valid characters for ingest on page 208](#).

### Input metadata

The input metadata of the **Merge Records** component is an arbitrary array of property names and values that define the data to be added to the Endeca data domain. If a record exists that matches the property value defined in the spec attribute (primary key), the new values are added to the existing record. If no record exists that matches the property value defined in the spec attribute, a new record is added to the Endeca data domain.

The following code illustrates an example input array:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-6738|White|500|375
CA-7457|Black|500|375
CB-2903|Silver|1000|750
```

```
CN-6137|Silver|1000|750
CR-7833|Gold|1000|750
FH-2981|Silver|500|375
```

Suppose that `ProductKey` is the spec attribute for the data domain. Also suppose that a record already exists with the following values: `ProductKey=CA-5965,Color=Silver`. The value `Black` will be added to the `Color` assignment on this record, resulting in the assignment `Color=Silver,Black`

The following assignments will also be added to the record:

- `SafetyStockLevel=500`
- `ReorderPoint=375`

Suppose that no records exist with values of `CR-7833` and `FH-2981` for `ProductKey`. New records will be created using the input values for the assignments on the records.

Multiple values can be input for an assignment using either a multi-assign delimiter or using a list data type. See [Processing multi-value data on page 49](#) for additional details.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Merge Records** component.

Table 9.16: Merge Records component properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified. The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>

Name	Description	Valid Values	Example
Spec Attribute	Specifies the primary key (record spec) for the records on which the operation will be performed.	Name of the primary key. If the primary key does not exist in the data domain, the property is created automatically with system default values.	FactSales_OrderNumber
Collection key	<p>Required. Specifies the key of the collection to which the data should be loaded.</p> <ul style="list-style-type: none"> <li>• If the Collection does not already exist, it will be created.</li> <li>• If the Collection already exists, any modifications will be added to the specified Collection. In other words, new records will be added to the specified collection, and records will be updated in the specified collection.</li> </ul> <p>Note that if you specify an existing collection, the value in the <b>Spec attribute</b> field must match the spec attribute of the existing collection.</p>	Alphanumeric characters, but must begin with either a letter or an underscore.	New Collection
SSL Enabled	<p>Enables or disables SSL for the component.</p> <p>SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.</p>	Checked (True) Unchecked (False)	

Name	Description	Valid Values	Example
Batch Size	<p>Specifies the batch size (in bytes) for the ingest operation. A batch consists of one or more complete records.</p> <p>See also <a href="#">Batch size adjustments by connectors on page 208</a>.</p>	<p>A positive integer equal to or greater than 1 defines the batch size. If the batch size is too small to fit the last record in the batch, the size is reset to accommodate that record. The batch size then returns to the specified batch size.</p> <p>Specifying zero (0) or a negative integer turns off batching. When batching is turned off, records are submitted to the data domain one at a time.</p>	<p>1000000</p> <p>0</p>
Multi-assign delimiter	<p>Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.</p> <p>See also <a href="#">Multi-assign delimiter on page 207</a>.</p>	<p>A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.</p>	
Maximum number of failed batches	<p>Sets the maximum number of batches that can fail before the ingest operation is ended.</p>	<p>Either a 0 (allows no failed batches) or a positive integer.</p>	<p>15</p>
Disable updates	<p>Disables updates to existing records. If this option is checked, and an input record matches an existing record, the operation fails, which causes the entire batch to fail.</p>	<p>Checked (True; if an input record matches an existing record, the update fails, which causes the entire batch to fail.)</p> <p>Unchecked (false; if an input record matches an existing record, the existing record is updated with the input data.)</p>	

## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.
- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.17: Port 0 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Number of Records Deleted	Long	Number of records deleted from the data domain as a whole	42683
Number of Records Affected	Long	Number of records from which key/value pairs (assignments) have been removed, while retaining the rest of the record	19834
Time Taken in Seconds	Numeric	Total time to process the batch, in seconds	127

Table 9.18: Port 1 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Fault Message	String	Error message returned by the Endeca Server	

## Modify Records component

Use the **Modify Records** component to modify assignments on records in an Endeca data domain. You can add, modify, or delete specific assignments.

The **Modify Records** component uses the Data Ingest Web Service (DIWS) to modify assignments on records in an Endeca data domain. The component can add new assignments to records, or can modify or delete existing assignments on records. You can modify multiple records in one transaction.

Two options are available to select the records to modify:

- You can include the attributes and values used to select the records in the input data array. These attributes are called Record Set Specifier Attributes.
- You can enter an Endeca Query Language (EQL) record set specifier (*WHERE* clause) to select the records to modify.

These options are mutually exclusive. If you choose to use an EQL record specifier to select the records to modify, you cannot also use an input array to select the records. An input array is still necessary to input the data to add, overwrite, or delete from assignments in your Endeca records.

If you are adding or modifying records, the input is still necessary to input the data to add or overwrite on the record.



**Note:** String data submitted for ingest must consist of valid XML characters. For details see [Valid characters for ingest on page 208](#).

### Using an input array

The input array for the **Modify Records** component is an arbitrary array of property names and values that specify the records to modify and the values to add to the records or delete from them. Note that null values are added to the specified records; null values overwrite all existing values.

The examples below use the following input array:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-6738|White|500|375
CA-7457|Black|500|375
CB-2903|Silver|1000|750
CN-6137|Silver|1000|750
CR-7833|Gold|1000|750
FH-2981|Silver|500|375
```

Assume `ProductKey` is the spec attribute. This value would be entered in the **Record Set Specifier Attributes** property to select the records to modify.

If the value of the **Operation** property is `Add Assignments`, and the records do not include values for the `Color`, `SafetyStockLevel`, and `ReorderPoint` properties, then the input values for these properties are added to the records specified by the `ProductKey` properties. If the records already include values for these properties, and the properties support multi-assign, the input values are appended to the existing values for these properties. If the properties do not support multi-assign, the operation fails; as a result, the entire batch fails.

If the value of the **Operation** property is `Replace Assignments`, the values in the input array replace any existing values of the specified properties in the records specified by the `ProductKey`. If the value of any input property is null, the null value overwrites all existing values in the property. For example, suppose the input array included the following record:

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
```

```
CA-6738 | |
```

Then the values of the `Color`, `SafetyStockLevel`, and `ReorderPoint` properties are overwritten with null values. (In other words, this operation is effectively a wildcard delete operation for these properties.)

If the value of the **Operation** property is `Delete Assignments` and the following array is input (`ProductKey` is the spec attribute):

```
ProductKey|Color|SafetyStockLevel|ReorderPoint
CA-5965|Black|500|375
CA-7457|Black| |
```

then:

- For the record where `ProductKey=CA-5965`, the input values are deleted from the specified properties.
- For the record where `ProductKey=CA-7457`, the assignment `Black` is deleted from the `Color` property.

Multi-value attributes can be specified using either a multi-assign delimiter or a list data type. See [Processing multi-value data on page 49](#) for additional details.

When using a multi-value attribute to select records, the matching behavior is defined by the value of the **Multi-assign specifier attribute behavior** property.

- If the value of the **Multi-assign specifier attribute behavior** property is `Matches exactly`, then Endeca records will only be selected if the value of the attribute on the record exactly matches the input value of the attribute. In other words, the value of the attribute on the Endeca record must include all values for the attribute on the input record, and only those values. If the value of the attribute on the Endeca record includes additional values beyond those on the input record, the Endeca record will not be selected.
- If the value of the **Multi-assign specifier attribute behavior** property is `Includes`, then Endeca records will be selected if the value of the attribute on the record includes one or more of the values on the input record. All values need not match, and the Endeca record can include additional values beyond those included on the input record.

## Using EQL to select the records to modify

You can use an EQL record specifier (the `WHERE` clause of an EQL query) to select the records to modify. Use variables to specify the input properties to use to select. For example, if you input the following array:

```
ProductCategory|ProductSubcategory|SafetyStockLevel|ReorderPoint
Components|Hardware|250|185
Components|Wheels|25|18
Clothing|Hats|40|35
Clothing|Shirts|25|20
```

you could enter the following EQL: `"EnglishProductCategoryName"=$input.ProductCategory` and `"EnglishProductSubcategoryName"=$input.ProductSubcategory`

When selecting records based on the value of multi-assign properties, you use a different syntax depending on whether you want to select values based on the complete set values of the attribute or you want to select values based on the individual members of the set of values of the attribute. If you want to select records based on the complete set of attribute values, use the syntax `"attribute"='value'`, and ensure that the input value is a multi-value set.

For example, if `Color` is a multi-assign attribute, and you want to update the assignments. You might define the following input array:

```
AssignedColors|NewColor
red,gree,blue|black
yellow,pink,orange|brown
```



and use the following EQL: `"Color"={ $input.AssignedColors}`.

Note that the order of members of the set is not significant. In other words, the input value `green,blue,red` selects the same set of records as the input value `red,green,blue`.

The value of the **Multi-assign specifier attribute behavior** property determines the behavior when records include more or fewer values than the input set. If the value of this property is `Includes`, records are selected if they include any member of the input set and do not need to match the values in the input set exactly. If the value of the **Multi-assign specifier attribute behavior** is `Matches exactly`, records are only selected if the value of the attribute includes all members of the input set. Records that contain more or fewer values will not be selected. In other words, when using the record specifier `"Color"={'red','green','blue'}`, a record where the value of `Color` is `"blue,red"` would not be selected; a record where the value of `Color` is `"red,green,blue,yellow"` would also not be selected.

To select records based on the value of members of the set of values of multi-assign properties, use the explicit qualification syntax. For example, if `Color` is a multi-assign attribute, and you define the following input array:

```
AssignedColors|NewColor
red|red
brown|brown
```

the record specifier `EVERY "value" IN "Color" SATISFIES("value"<>$input.AssignedColors)` selects records where the values of the `Color` attribute do not include the values in the `AssignedColors` input field.

For full details about Endeca Query Language, see the *Oracle Endeca Server EQL Guide*.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Modify Records** component.

Table 9.19: Modify Records component properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>

Name	Description	Valid Values	Example
Data Domain Name	<p>Name of the data domain that will be modified.</p> <p>The data domain should be running when the graph containing the connector is run.</p>	Valid data domain names	quickstart
Operation	<p>Specifies the operation to perform. Options include:</p> <ul style="list-style-type: none"> <li>• Add Assignments</li> <li>• Replace Assignments</li> <li>• Delete Assignments</li> </ul>		
Record Set Specifier Attributes	Specifies the input attributes to use to select the records on which to operate	Attributes available from the input metadata	
Multi-assign specifier attribute behavior	Specifies how multi-value input attributes will be matched when selecting records.	<ul style="list-style-type: none"> <li>• Matches exactly Records are selected only if the value of the attribute in the record includes all input values for the attribute.</li> <li>• Includes Records are selected if the value of the attribute on the record matches any of the input values for the attribute.</li> </ul>	

Name	Description	Valid Values	Example
Use EQL Record Set Specifier	<p>Specifies whether to use an EQL expression to select the records on which to operate.</p> <p>Standard EQL practice is to use double quotation marks around attribute names and single quotation marks around attribute values.</p>	Unchecked (false) Checked (true)	<pre>"DimGeography_City" ='Newton' AND "DimGeography_State ProvinceName"='Brit ish Columbia'</pre>
Multi-assign specifier attribute behavior	<p>Specifies how multi-value input attributes will be matched when selecting records.</p>	<ul style="list-style-type: none"> <li>• Matches exactly Records are selected only if the value of the attribute in the record includes all input values for the attribute.</li> <li>• Includes Records are selected if the value of the attribute on the record matches any of the input values for the attribute.</li> </ul>	
EQL Record Set Specifier	<p>The EQL expression to use to select the records on which to operate.</p>	A valid EQL expression	
SSL Enabled	<p>Enables or disables SSL for the component.</p> <p>SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.</p>	Checked (True) Unchecked (False)	

Name	Description	Valid Values	Example
Batch Size	<p>Specifies the batch size (in bytes) for the ingest operation. A batch consists of one or more complete records.</p> <p>See also <a href="#">Batch size adjustments by connectors on page 208</a>.</p>	<p>A positive integer equal to or greater than 1 defines the batch size. If the batch size is too small to fit the last record in the batch, the size is reset to accommodate that record. The batch size then returns to the specified batch size.</p> <p>Specifying zero (0) or a negative integer turns off batching. When batching is turned off, records are submitted to the data domain one at a time.</p>	<p>1000000</p> <p>0</p>
Multi-assign delimiter	<p>Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.</p> <p>See also <a href="#">Multi-assign delimiter on page 207</a>.</p>	<p>A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.</p>	
Maximum number of failed batches	<p>Sets the maximum number of batches that can fail before the ingest operation is ended.</p>	<p>Either a 0 (allows no failed batches) or a positive integer.</p>	<p>15</p>

## Output Ports

Each Information Discovery component that modifies record data in the data domain (adding or removing records or key/value pairs) has two output ports:

- **Port 0** returns status information describing batches of records that were successfully ingested.

- **Port 1** returns error information describing batches of records that the data domain failed to ingest. Each output record to the port corresponds to a failed batch, not to individual records.

Table 9.20: Port 0 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Number of Records Deleted	Long	Number of records deleted from the data domain as a whole	42683
Number of Records Affected	Long	Number of records from which key/value pairs (assignments) have been removed, while retaining the rest of the record	19834
Time Taken in Seconds	Numeric	Total time to process the batch, in seconds	127

Table 9.21: Port 1 metadata

Port Field Name	Data Type	Description	Example
Start Row	Long	ID of starting row of the batch	00001
End Row	Long	ID of ending row of the batch	99999
Fault Message	String	Error message returned by the Endeca Server	

## Record Store Reader

The **Record Store** Reader reads records from a Integrator ETL Acquisition System (IAS) Record Store instance.



The **Record Store Reader** component reads Endeca records stored in an Endeca Record Store. These records are derived by IAS from crawls of file systems, content management systems, Web servers, and custom data sources. The extracted records can be written to an output file or loaded into an Endeca data domain via an Information Discovery component.

## Input metadata

The input metadata for the **Record Store Reader** component is not fixed. Use the Record Store Metadata Wizard to extract the metadata from the Record Store instance.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Record Store Reader** component.

Table 9.22: Record Store Reader properties

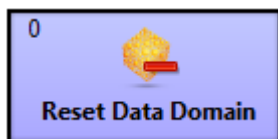
Name	Description	Valid Values	Example
IAS Service Host	The name or IP address of the machine on which the IAS Services is running.	Valid host names Valid IP addresses	localhost (default) 255.255.255.0
IAS Service Port	The port on which the IAS Service listens.	Valid port numbers	8401 (default)
IAS Service Context Root	The context root of the IAS Service web application when IAS is installed to a WebLogic Server container.  The default IAS context root is <code>ias-service</code> .  If you install IAS to a Jetty container, you can ignore this field.	String	<code>ias-service</code>
IAS Record Store Instance	The name of the unique Record Store instance from which to read records.	Record Store names	<code>productdata</code>

Name	Description	Valid Values	Example
IAS Client ID	<p>An arbitrary name that identifies the Integrator ETL client (or a specific instance of the Record Store Reader) to the Record Store.</p> <p>The Record Store uses the value of the IAS Client ID to keep track of which generations of records have been read by this client. If multiple clients (multiple instances of the Record Store Reader) access the same Record Store, each instance should have a unique name.</p>	Alphanumeric characters	IntegratorETL (default)
IAS Read Type	Specifies the type of read operation to use to read records from the Record Store.	<ul style="list-style-type: none"> <li>• <b>Full Extract</b> Retrieves all records from the most recently committed generation in the Record Store (also known as "baseline records").  This is the default option.</li> <li>• <b>Incremental</b> Retrieves only records that were added or changed since the last committed generation (also known as "delta records").</li> </ul>	
SSL Enabled	<p>Enables or disables SSL for the component.</p> <p>SSL should only be enabled when the IAS instance to which you are connecting has SSL enabled.</p>	<p>Checked (True)</p> <p>Unchecked (False)</p>	

Name	Description	Valid Values	Example
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.  See also <a href="#">Multi-assign delimiter on page 207</a> .	A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.	
Read Batch Size	The number of records fetched from the Record Store in a batch.	Integers greater than 0	100 (default)
Socket Timeout	The maximum time, in milliseconds, to wait between two consecutive data packets.	Integers greater than or equal to 0 A value of 0 is treated as an infinite timeout.	300000 (default; five minutes)
Client Timeout	The maximum time to wait for a connection to be established with the IAS Service.	Integers greater than or equal to 0 A value of 0 is treated as an infinite timeout.	300000 (default; five minutes)

## Reset Data Domain component

The **Reset Data Domain** component resets the Endeca data domain to the empty state.



The **Reset Data Domain** component removes all of the records (including the schema and view definitions) from the Endeca data domain, re-provisions the Endeca data domain, and updates the spelling dictionary. This component uses operations from the Data Ingest Web Service. These operations delete all records (including schema records) and configurations, and provision the Endeca data domain. Then, the component uses the `updateSpellingDictionaries` operation of the Data Ingest Web Service to update the spelling dictionary.



## Use cases

Use this component when you want to clear the data domain. For example:

- During development, you may want to reset the data domain to remove incorrect data domain configurations.
- In production, you may want to refresh the data domain through a baseline update.

## Configuration properties

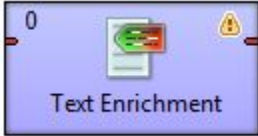
The following table describes the configuration properties available for the **Reset Data Domain** component.

Table 9.23: Reset Data Domain properties

Name	Description	Valid Values	Example
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use <code>localhost</code> .	<code>MyEndecaServer</code> <code>255.255.255.0</code>
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7001, but it can be changed to another port.	7001
Endeca Server Context Root	Identifies the WebLogic application root context of the Endeca Server	Valid root context names in WebLogic	<code>/endeca-server</code>
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	<code>quickstart</code>
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	

## Text Enrichment component

The **Text Enrichment** component can extract, summarize, and assess input text.



The **Text Enrichment** component uses the Saliency Engine from Lexalytics to extract entities (people, places, organizations, themes, and quotes) from source files. The extracted entities can be written to an output file or loaded into an Endeca data domain.

The Saliency Engine also provides the ability to assess the sentiment of input text.

### Metadata schema

The metadata schema for the **Text Enrichment** component is not fixed.

### Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Text Enrichment** component.

Table 9.24: Text Enrichment properties

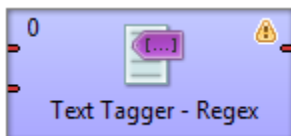
Name	Description	Valid Values	Example
Configuration file	Absolute path to the Text Enrichment properties file.  Recommended practice is to store the configuration file in the project directory.	Valid file path  You can use <code>\${PROJECT}</code> or a similar global variable to specify the path.	<code>\${PROJECT}/TextEnrichments.properties</code>
Input field	Name of the source field in the input source record that you want to enrich (extract entities and assess sentiment)	Field names	<code>survey_responses</code>

Name	Description	Valid Values	Example
Salience data path	Absolute path to the Lexalytics data directory	Valid file path	C:/Program Files (x86)/Lexalytics/data  /usr/endeca/salience/data
Error handling key field	Specifies a field to store error-handling output. You must specify a value for this field. If you do not have a specific error field, you can specify the primary key field name. (The primary key field must exist in the input metadata.)	Alphanumeric characters	salience_errors
Text threshold (percent)	The minimum percentage of alphanumeric characters that the input field must contain for the field to be processed. If no threshold is specified, the system default is 80.	Positive integers	80
Number of threads	The number of threads the component should consume. If no thread count is specified, the component uses one thread.	Positive integers	4
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.  See also <a href="#">Multi-assign delimiter on page 207</a> .	A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.	

Name	Description	Valid Values	Example
Saliency Warning Log Level	<p>Specifies the logging level of messages from the Saliency engine that will be reported in the Integrator ETL log.</p> <p>Defaults to <code>WARN</code></p> <p>Note that the value of this property does not override the logging level of Integrator ETL as a whole. If the logging level of Integrator ETL is configured to a less-detailed logging level than the Saliency logging level, the detailed Saliency logging output will not be included in the Integrator ETL logging output.</p>	<ul style="list-style-type: none"> <li>• OFF</li> <li>• FATAL</li> <li>• ERROR</li> <li>• WARN</li> <li>• INFO</li> <li>• DEBUG</li> <li>• TRACE</li> <li>• ALL</li> </ul>	

## Text Tagger Regex component

The **Text Tagger Regex** component uses regular expressions (regexes) to match text in a specified text field on the incoming records and then tags the output records with a computed value.



The **Text Tagger Regex** component takes a list of search pattern/render pattern pairs and searches for the input patterns in the field specified in the **Source Field Name** property in the input record. If the regular expression results in a match in a record, an output generated by the render pattern (from the search pattern/render pattern input) is added to the field on the record specified in the **Target Field Name** property.

### Metadata schema

The metadata schema for the **Text Tagger Regex** component is not fixed.

### Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

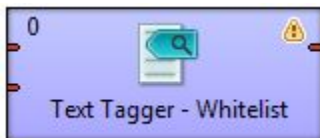
The following table describes the configuration properties available for the **Text Tagger Regex** component.

Table 9.25: Text Tagger Regex properties

Name	Description	Valid Values	Example
Source Field Name	Name of the text field in the input records that will be searched for matches.	Properties in the input record.	
Target Field Name	Name of the field in the output records into which the tags will be written.	Valid property names.	
Overwrite Target Field	Specifies whether to overwrite any existing input value in the specified target field with the new tag output.	Checked (True) Unchecked (False)	
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.  See also <a href="#">Multi-assign delimiter on page 207</a> .	A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (\U007F). You do not have to use this field if your data does not include multi-assign properties.	

## Text Tagger Whitelist component

The **Text Tagger Whitelist** component uses tags-rule input to define the terms to match in a specified text field on incoming records and the value to write out to the target field.



The **Text Tagger Whitelist** component takes a list of search-term/tag-value pairs and searches for the terms in the field specified in the **Source Field Name** property in the input record. If a term matches in a record, the tag value (from the tags-rule input) is added to the field on the record specified in the `Target Field Name` property.

## Metadata schema

The metadata schema for the **Text Tagger Whitelist** component is not fixed.

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Text Tagger Whitelist** component.

Table 9.26: Text Tagger Whitelist properties

Name	Description	Valid Values	Example
Source Field Name	Name of the text field in the input records that will be searched for matches.	Properties in the input record.	
Target Field Name	Name of the field in the output records into which the tags will be written.	Valid property names.	
Overwrite Target Field	Specifies whether to overwrite any existing input value in the specified target field with the new tag output.	Checked (True) Unchecked (False)	
Case Sensitive Matches	Specifies whether a match requires that the case of characters in the string match the value in the tags-rule field.	<ul style="list-style-type: none"> <li>• True Case must match the input string for a valid match.</li> <li>• False A match is valid regardless of whether case matches.</li> </ul>	

Name	Description	Valid Values	Example
Multi-assign delimiter	<p>Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.</p> <p>See also <a href="#">Multi-assign delimiter on page 207</a>.</p>	A single character that is the multi-assign delimiter. The default is the Unicode DELETE character (U007F). You do not have to use this field if your data does not include multi-assign properties.	
Search Term Maximum Character length	Specifies the maximum length, in characters, of terms in the tags-rule input.	Positive integers	51

Name	Description	Valid Values	Example
Number of threads	<p>Specifies the number of processing threads the component should use when processing text.</p> <p>When processing large whitelists (more than 1000 Search Terms), or very large text fields, or both, you may notice that processing takes a long time. In that situation, you can improve performance by increasing the number of threads used in white list text tagging processing. A good starting point is to specify a number of threads that matches the number of processing cores available. Actual performance is affected by a number of factors, including the number of cores available and other processing taking place on the machine, so you may need to adjust the number of threads to achieve the desired results.</p>	<p>Positive integers</p> <p>Defaults to 1.</p>	1

## Transaction RunGraph component

The **Transaction RunGraph** component runs multiple Integrator ETL graphs within a single Endeca Server outer transaction.





The **Transaction RunGraph** component starts an outer transaction and runs one or more sub-graphs within that transaction. If all sub-graphs complete successfully, the transaction as a whole completes. If any of the graphs in the transaction fail, the transaction fails as a whole.

- You can run one graph or multiple graphs.
  - To run one graph, specify the URL of the graph in the **Graph URL** property.
  - To run multiple graphs, specify the names of the graphs you want to run in file, and read the file in using a **UniversalDataReader**. Pass this data to the input port of the **Transaction RunGraph** component.
- The **Transaction RunGraph** starts and commits an outer transaction using the Transaction Web Service. Only one outer transaction can be open at a time.

A graph that is included in an outer transaction should not be run on its own while the outer transaction is running.

- By default, if a transaction fails, the component rolls back to the state before the transaction started and commits the transaction, but you can configure the component to commit any changes that were made successfully before the failure, or to do nothing, leaving the transaction running.
- The **Transaction RunGraph** component overrides the value of the transaction ID with the string `transaction` for the duration of the transaction. This behavior assumes that the value of the outer transaction ID parameter in `workspace.prm` is empty: `OUTER_TRANSACTION_ID=`
- All components in sub-graphs that:
  - Submit a request to the data domain using either a web service or the Bulk Load Interface, and
  - Run inside **Transaction RunGraph**

must reference the outer transaction ID when run as part of an outer transaction. The value of the outer transaction ID property of any of these components should use the outer transaction ID global variable (typically `${OUTER_TRANSACTION_ID}`).

- If you use a **WebServiceClient** component that is configured to run any of the data domain Web services, and plan to use this component inside **Transaction RunGraph**, the **Request Structure** field for the component must include an `OuterTransactionId` as the first element, with a value of an outer transaction.



**Note:** If you do not use outer transactions, then your Web service-based components should still use the `OuterTransactionID` referencing the value in `workspace.prm`. If the value is empty, the transaction ID attribute is ignored by the data domain. This practice allows components to run outside of transactions, without having to modify `workspace.prm`.

For example, the following request specified in the **Request Structure** references the outer transaction ID as a parameter:

```
<config-service:configTransaction
xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
<config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
<config-service:putGroups
xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
...
</config-service:putGroups>
</config-service:configTransaction>
```

In this example, the element `OuterTransactionId` specifies the ID of the outer transaction listed in the `workspace.prm` file for your project.

## Use cases

Use the **Transaction RunGraph** component when you need to run multiple graphs within a single transaction. The most common situation that calls for this functionality is implementing initial loads and baseline updates of data domains.

- The typical initial load starts with a **Reset Data Domain** component to provision the data domain, followed by one or more graphs to load the configuration and data. All are wrapped in a transaction using the **Transaction RunGraph** component.
- The typical baseline update starts with a graph that exports the configuration and schema using **Export Config** component. Next, a graph with a **Reset Data Domain** component removes all records and re-provisions the Endeca data domain. An **Import Config** component reloads the previously saved configuration and schema, after which one or more graphs reload the records and record attribute values. All are wrapped in a transaction using the **Transaction RunGraph** component.

## Metadata schema

The metadata schema for the **Transaction RunGraph** component is not fixed.

The metadata type of the Integrator ETL field (as shown in the Edit Metadata dialog on the edge connecting to the connector) translates to the `mdex` property type. For example, the Integrator ETL `integer` data type translates to the `mdex:int` data type. Note that you must override this behavior to support Integrator ETL non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`). For details, see [Creating mdexType Custom properties on page 39](#).

## Configuration properties



**Note:** For details about visual properties for all connectors, see [Visual properties of components on page 205](#). For details about configuration properties common to all connectors, see [Common configuration properties of components on page 206](#).

The following table describes the configuration properties available for the **Transaction RunGraph** component.

Table 9.27: Transaction RunGraph properties

Name	Description	Valid Values	Example
Graph URL	Path to an individual graph you want to run within the transaction.  Enter a value in this field only if you want to run a single graph in the transaction. If you want to run multiple graphs, create an input file and read the values using a reader component.	Valid path to a graph.	
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. You can use localhost.	MyEndecaServer 255.255.255.0
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports.  The default Endeca Server port is 7001, but it can be changed to another port.	7001
Data Domain Name	Name of the data domain that will be modified.  The data domain should be running when the graph containing the connector is run.	Valid data domain names	quickstart

Name	Description	Valid Values	Example
Upon failure	Specifies Integrator ETL behavior when a transaction fails.	<ul style="list-style-type: none"> <li data-bbox="834 344 1127 609">• Rollback Default. When a transaction fails, roll the data domain back to the state before the transaction started and commit the transaction.</li> <li data-bbox="834 632 1127 896">• Commit When a transaction fails, commit any successful changes to the data domain before the fail occurred and commit the transaction.</li> <li data-bbox="834 919 1127 1629">• Do nothing When a transaction fail, stop and do nothing else. When you choose this option, investigate the logs to determine whether you want to apply any successful changes manually. You also need to manually end the transaction by running a commit transaction operation. You can either use a graph with a <b>Web Services Client</b> component, or you can execute the web service operation directly.</li> </ul>	

Name	Description	Valid Values	Example
SSL Enabled	Enables or disables SSL for the component.  SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	
Timeout (ms)	Specifies the timeout of operations of the component.  If timeouts occur when running graphs, operations on the Endeca Server may be taking too long. Change the value of this parameter to allow more time for operations on the Endeca Server.  The default configures a one minute timeout.	Integers	60000

## Visual properties of components

All components share the same set of visual properties.

Visual properties affect the appearance of the component. For more information on the purpose of these properties, see the *Oracle Endeca Information Discovery Integrator ETL Designer Guide*.

Table 9.28: Visual properties of Information Discovery components

Visual property	Purpose	Valid values
<b>Component name</b>	Displays the component name when the component is placed on a graph.	You can change the default name to a more descriptive one.
<b>Location</b>	Describes the location (using an X-axis and Y-axis) of the component icon within the graph.	Do not edit this field. Instead, use your cursor to move the component in the graph to the desired position.

Visual property	Purpose	Valid values
<b>Size</b>	Describes the dimensions (size) of the component icon within the graph.	Do not edit this field.
<b>Click here to edit component description</b>	Located in the header of the component, this field lets you add some descriptive text that is displayed in the component icon in the graph.	Text describing what this component does (for example, text that best describes what this component does in the graph).

## Common configuration properties of components

Common configuration properties are available on all Information Discovery components.

For more information on the purpose of these properties, see the *Oracle Endeca Information Discovery Integrator ETL Designer Guide*.

Table 9.29: Common properties of Information Discovery components

Common property	Purpose	Valid values
<b>ID</b>	Identifies the component among all of the other components within the same component type.	Do not edit this field.
<b>Component type</b>	Describes the type of the component. By adding a number to this component type, you can get a component ID.	Do not edit this field.
<b>Specification</b>	Describes what this component can do.	Do not edit this field.
<b>Phase</b>	Sets the phase number for the component. Because each graph runs in parallel within the same phase number, all components and edges that have the same phase number run simultaneously.	An integer number of the phase to which the component belongs.

Common property	Purpose	Valid values
<b>Enabled</b>	Enables or disables the component for parsing data.	<ul style="list-style-type: none"> <li>enabled (the default) means the component can parse data.</li> <li>disabled means the component does not parse data.</li> <li>passThrough puts the component in passThrough mode, in which data records will pass through the component from input to output ports and the component will not change them.</li> </ul>
<b>Pass Through Input Port</b>	If the component runs in passThrough mode, you can specify which input port should receive the data records.	Select the input port from the list of all input ports.
<b>Pass Through Output Port</b>	If the component runs in passThrough mode, you can specify which output port should send the data records out.	Select the output port from the list of all output ports.
<b>Allocation</b>	If the graph is executed by a cluster of Integrator ETL Servers, this attribute must be specified in the graph.	For information on this property, see the <i>Oracle Endeca Information Discovery Integrator ETL Designer Guide</i> .

## Multi-assign delimiter

Multiple components include a **multi-assign delimiter** field.

The **multi-assign delimiter** field specifies the character used to separate tags when writing multiple tag values to the target field. The default value is the Unicode Delete character (\U007F).

Always specify the same multi-assign delimiter for all components in the same graph. If you specify different characters in different components, the set of tags output by earlier components are read by later components as a single value.

For example, if you specify a pipe as the delimiter in a **Text Tagger** component, and the default separator in the **Bulk Loader** component, a set of tags output by the **Text Tagger** are read by the **Bulk Loader** as a single value, rather than as multiple values.

## Batch size adjustments by components

Several components allow you to specify a batch size, but specific records may cause a batch to exceed the specified size.

The following components allow you to specify a batch size:

- **Add KVPs**
- **Delete Records**
- **Merge Records**
- **Modify Records**
- **Record Store Reader**

Regardless of the batch size you have specified (assuming you have not disabled batching by specifying zero or a negative number), the component adjusts the batch size on the fly to ensure that all the assignments for a given record fit in its batch. This behavior ensures that assignments for a given record are not split between different batches. Note that only the individual batch is extended, the overall batch size setting is not adjusted for future batches.

## Valid characters for ingest

A valid character for ingest must be a character according to the XML specification.

See the [Second Edition of the XML 1.0 Specification](#) for details about valid characters.

If the Endeca Server detects an invalid character, it rejects the record and returns the following message to Integrator ETL:

```
Error: Character <c> is not legal in XML 1.0
```

The error message is added to the log for the run.

Only the record that includes the invalid character is rejected. The rest of the ingest operation continues.

To clean your data, you can add a **Reformat** component to the graph that includes this component and use the following code:

```
//#CTL2

// Transforms input record into output record.
function integer transform() {
  string regex = "([^\u0009\u000a\u000d\u0020-\u007F\u00FF]|[\u0092\u007F]+)";
  $0.YourDataCleanData = replace($YourDatawithInvalidPattern,regex,"");

  return ALL;
}
```

Compatibility characters are also not valid. The code above removes compatibility characters.





## Appendix A

---

# Implementing SSL

This section describes how to implement SSL to provide secure communication between Integrator ETL and Integrator ETL Server and Endeca Server.

You must use the same certificate for Integrator ETL, Integrator ETL Server, and Endeca Server.

[\*Configuring SSL for Information Discovery components\*](#)

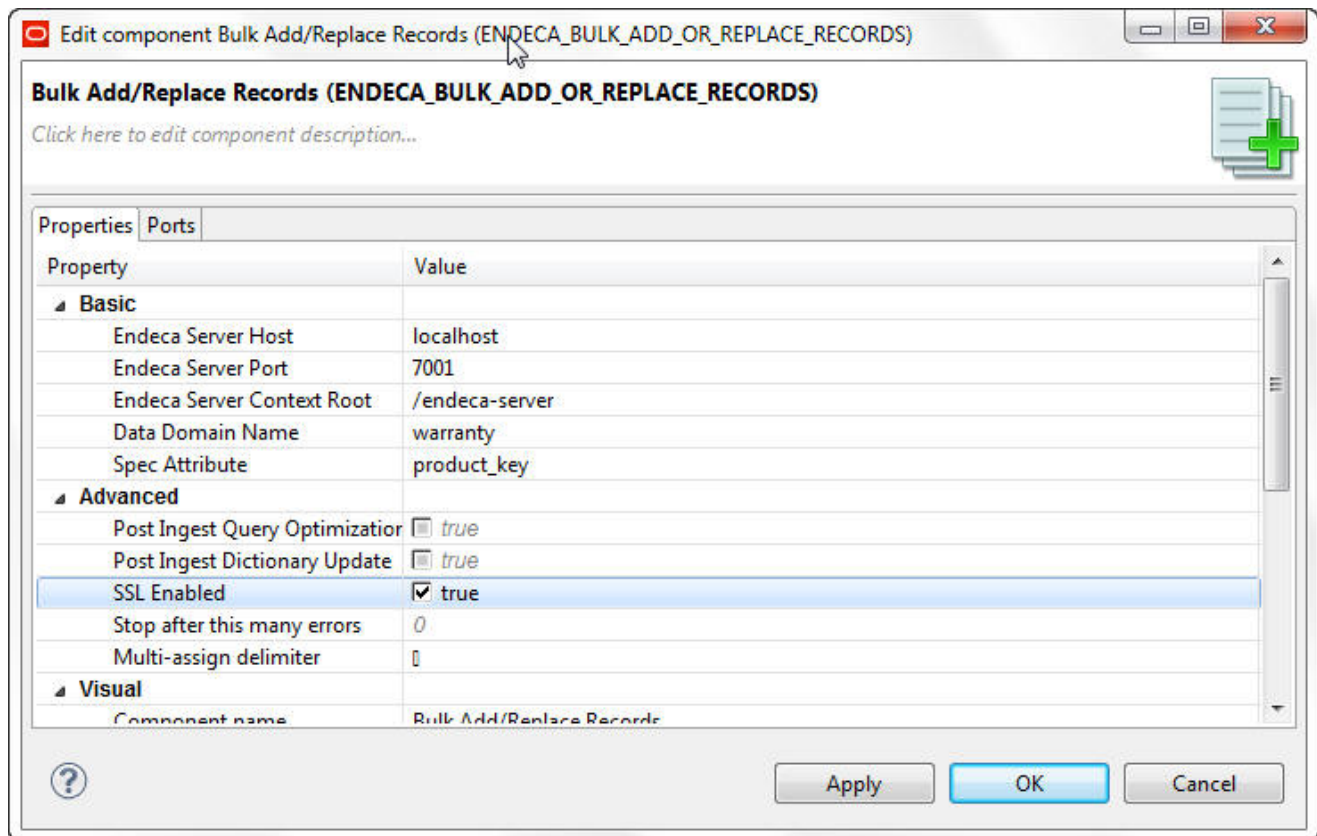
[\*Configuring SSL for the Web Service Client component\*](#)

[\*Configuring Integrator ETL Server to support SSL\*](#)

## Configuring SSL for Information Discovery components

All Information Discovery components support SSL connections to an SSL-enabled Dgraph.

This procedure assumes that you have used the Integrator ETL **Edit component** dialog for the Information Discovery component and set the **SSL Enabled** configuration property to `true`. For example, this **Bulk Add/Replace Records** component has been enabled for SSL:



The procedure also assumes that you have created the necessary SSL keystore and truststore certificates.

To configure SSL support for a graph using an Information Discovery component:

1. Select **Preferences** from the **Window** menu.
2. From the **Preferences** menu, select **Java>Installed JREs**.
3. In the **Installed JREs** menu, click on the checked JRE and then click **Edit**.

The **Edit JRE** menu is displayed.

4. In the **Default VM Arguments** field, enter the following on a single line. Replace the filenames with the ones you created:

```
-Djavax.net.ssl.keyStore=yourcertkeystorefile.jks
-Djavax.net.ssl.keyStorePassword=keystorepass
-Djavax.net.ssl.trustStore=yourtruststorefile.jks
-Djavax.net.ssl.trustStorePassword=truststorepass
```

5. Click **Finish** to apply your change and close the **Edit JRE** menu.

6. Click **OK** to close the **Preferences** menu.

## Configuring SSL for the Web Service Client component

The **Web Service Client** component requires special configuration to support secure communication with Endeca Server over SSL.

If you want to use the **Web Service Client** in an SSL environment, you must check the **Disable SSL Certificate Validation** box on the instance of the **Web Service Client** in your graph.

You must also add the following code to the `IntegratorETL.ini` file in the root of your Integrator ETL installation. Add this code under `"-vmargs"`:

```
-Djavax.net.ssl.keyStore=yourcertkeystorefile.jks
-Djavax.net.ssl.keyStorePassword=keystorepass
-Djavax.net.ssl.trustStore=yourtruststorefile.jks
-Djavax.net.ssl.trustStorePassword=truststorepass
```

Replace the file names with the names of your keystore and truststore files.

For example:

```
-vmargs
-Dosgi.requiredJavaVersion=1.5
-XX:MaxPermSize=256m
-Xms40m
-Xmx512m
-Djavax.net.ssl.trustStore=/endeca_server/ssl/endecaServerTrustStore.ks
-Djavax.net.ssl.trustStorePassword=endeca
-Djavax.net.ssl.keyStore=/endeca_server/ssl/endecaServerClientCert.ks
-Djavax.net.ssl.keyStorePassword=endeca
```

When using SSL, be sure all Web service requests use https rather than http.

## Configuring Integrator ETL Server to support SSL

To implement SSL on Integrator ETL Server, copy the keystore and truststore files from the Endeca Server installation to the Integrator ETL Server container, then add Java options with the path and password of the keystore and truststore files.

### WebLogic Server container on Linux

Add the keystore and truststore files to your WebLogic installation.

In the file `$DOMAIN_HOME/bin/setDomainEnv.sh`, after setting the `USER_MEM_ARGS` variable, add the `JAVA_OPTIONS` argument with the properties for the keystore and the truststore.

For example:

```
export JAVA_OPTIONS="
-Djavax.net.ssl.trustStore=/endeca_server/ssl/endecaServerTrustStore.ks
-Djavax.net.ssl.trustStorePassword=endeca
-Djavax.net.ssl.keyStore=/endeca_server/ssl/endecaServerClientCert.ks
-Djavax.net.ssl.keyStorePassword=endeca"
```



**Note:** The formatting above is for clarity on the printed page. In your file, the variables should be entered on one line.

## WebLogic Server container on Windows

Add the keystore and truststore files to your WebLogic installation.

In the file `$DOMAIN_HOME\bin\setDomainEnv.cmd`, after setting the `USER_MEM_ARGS` variable, add the `JAVA_OPTIONS` argument with the properties for the keystore and the truststore.

For example:

```
set JAVA_OPTIONS=  
-Djavax.net.ssl.trustStore=C:\endeca_server\ssl\endecaServerTrustStore.ks  
-Djavax.net.ssl.trustStorePassword=endeca  
-Djavax.net.ssl.keyStore=C:\endeca_server\ssl\endecaServerClientCert.ks  
-Djavax.net.ssl.keyStorePassword=endeca
```



**Note:** The formatting above is for clarity on the printed page. In your file, the variables should be entered on one line.

## Tomcat container on Linux

Add the keystore and truststore files to your Tomcat installation.

In the file `$CATALINA_BASE/bin/setenv.sh`, add the keystore and truststore properties to `JAVA_OPTS`.

For example:

```
export JAVA_OPTS="-Xms128m -Xmx2048m -XX:MaxPermSize=256m -server  
-Djavax.net.ssl.trustStore=/endeca_server/ssl/endecaServerTrustStore.ks  
-Djavax.net.ssl.trustStorePassword=endeca  
-Djavax.net.ssl.keyStore=/endeca_server/ssl/endecaServerClientCert.ks  
-Djavax.net.ssl.keyStorePassword=endeca $JAVA_OPTS"
```



**Note:** The formatting above is for clarity on the printed page. In your file, the variables should be entered on one line.

## Tomcat container on Windows

Add the keystore and truststore files to your Tomcat installation.

In the file `$CATALINA_BASE/bin/setenv.bat`, add the keystore and truststore properties to `JAVA_OPTS`.

For example:

```
set "JAVA_OPTS=-Xms128m -Xmx2048m -XX:MaxPermSize=256m -server  
-Djavax.net.ssl.trustStore=C:\endeca_server\ssl\endecaServerTrustStore.ks  
-Djavax.net.ssl.trustStorePassword=endeca  
-Djavax.net.ssl.keyStore=C:\endeca_server\ssl\endecaServerClientCert.ks  
-Djavax.net.ssl.keyStorePassword=endeca %JAVA_OPTS%"
```



**Note:** The formatting above is for clarity on the printed page. In your file, the variables should be entered on one line.



---

# Troubleshooting Problems

This section provides information and solutions to problems you may encounter when working with components and graphs.

[OutOfMemory errors](#)

[Transaction-related errors](#)

[Connection errors](#)

[Multi-assign delimiter error](#)

## OutOfMemory errors

If insufficient memory is allocated to the Java process, `OutOfMemory` may occur when you run graphs.

When a run fails due to inadequate memory allocation, the Console Tab includes messages such as the following:

```
ERROR [DataIngestBatchConsumer-0] - Failed with the following exception:
    java.lang.OutOfMemoryError: Java heap space
Exception in thread "DataIngestBatchConsumer-0" java.lang.OutOfMemoryError:
Java heap space
```

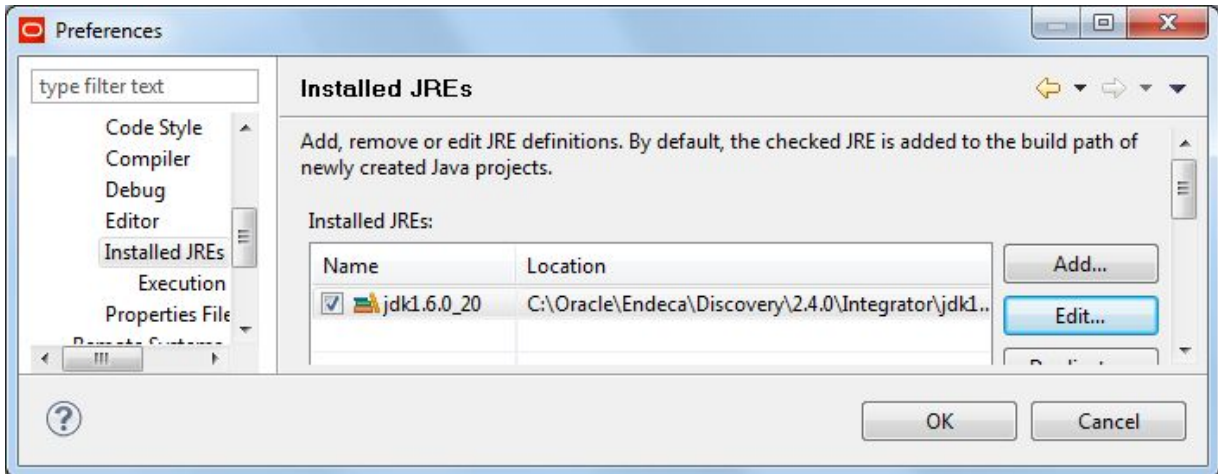
To avoid out `OutOfMemory` errors, increase the memory allocated to the Java process running the service. Use the **Edit JRE** dialog to modify the memory allocation to the Java process.

To modify memory allocations:

1. In the menu bar, choose **Window > Preferences**  
Integrator ETL displays the **Preferences** dialog.

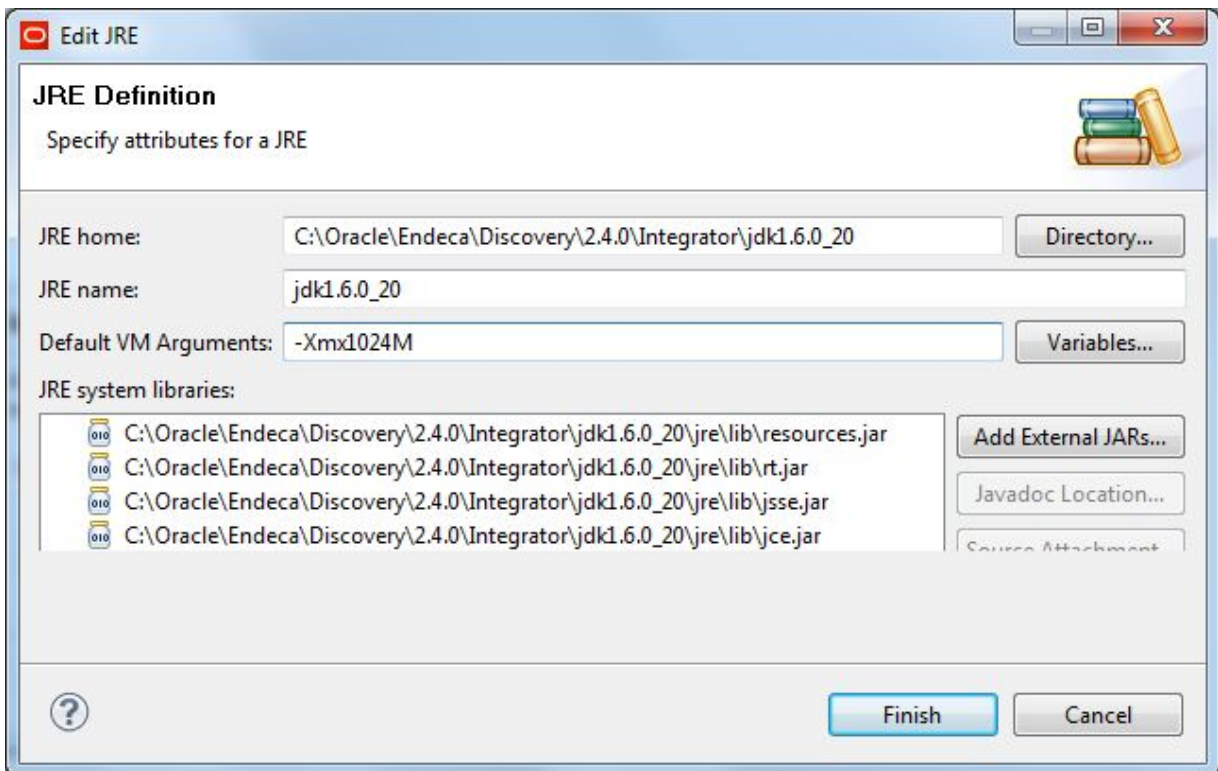
- In the left navigation pane of the **Preferences** dialog, expand the **Java** node. Click **Installed JREs** node.

The **Preferences** dialog displays the **Installed JREs** page.



- In the **Installed JREs** menu, click on the checked JRE and then click **Edit**.  
Integrator ETL displays the **Edit JRE** page.
- In the **Default VM Arguments** field, specify a Java option to set the heap size, such as `-Xmx1024M`.

The **Edit JRE** menu should look like this example:



- Click **Finish** to apply your change and close the **Edit JRE** menu.

6. Click **OK** to close the **Preferences** menu.

## Transaction-related errors

Transaction errors can occur for a variety of reasons.

When running an graph that starts an outer transaction, the graph fails and an error such as the following is reported in the Console Log:

```
ERROR [ENDECA_TRANSACTION_RUN_GRAPH0_0] - Connection refused: connect Error starting transaction ID
transaction. Ensure another transaction isn't already in progress.
ERROR [ENDECA_TRANSACTION_RUN_GRAPH0_0] - Connection refused: connect Error running internal sub
graphs
INFO [ENDECA_TRANSACTION_RUN_GRAPH0_0] - Transaction ID 'transaction' still open, but configured to
do nothing
```

When you run a graph, it fails with an error similar to the following:

These errors may occur for the following reasons:

- You have attempted to run multiple graphs that start outer transactions (in other words, multiple graphs that include the **Transaction RunGraph** component) at the same time.
  - This situation can occur if a graph that runs an outer transaction fails and continues running, and you start another graph that runs an outer transaction.
 

To determine whether an outer transaction is currently running, issue a `listOuterTransaction` request through the Transaction Web Service. If the response includes a value for the outer transaction ID, an uncommitted transaction is still running. To terminate the running transaction, commit it.
  - This situation also occurs if you start multiple graphs that initiate outer transactions at the same time. You can only run one outer transaction at a time. Allow a running outer transaction to complete before starting another outer transaction.
- You have attempted to run a graph that is included in an outer transaction at the same time as are running the outer transaction that includes the graph.

For example, suppose Graph A includes the **Transaction RunGraph** component and also includes Graph B as one of the graphs to run within the transaction. If Graph A is running, and you start Graph B, Graph B will fail.

## Connection errors

Connection errors occur when an Integrator ETL component cannot connect to the Endeca Server.

When a connection error occurs, an error similar to the following is reported in the Console Log:

```
ERROR [ENDECA_ADD_KVPS1_0] - Connection refused: connect Error connecting
to the Endeca Server. If applicable, ensure your SSL settings are correct
ERROR [ENDECA_ADD_KVPS1_0] - Failed with the following exception:
  java.rmi.RemoteException: Connection refused: connect Error connecting
to the Endeca Server. If applicable, ensure your SSL settings are correct;
nested exception is:
  org.apache.axis2.AxisFault: Connection refused: connect
ERROR [WatchDog] - Graph execution finished with error
...
```

Connection errors occur for the following reasons:

- The configuration of a component in the graph specifies an incorrect host for the Endeca Server.  
Review the components in the graph and ensure that any component that specifies an Endeca Server host specifies the correct host.
- The configuration of a component in the graph specifies an incorrect port for the Endeca Server.  
Review the components in the graph and ensure that any component that specifies an Endeca Server specifies the correct port.
- A component in the graph does not have SSL enabled but is trying to connect to an Endeca Server that does have SSL enabled.  
If SSL is enabled on the Endeca Server, review the components and ensure that all components that connect to an Endeca Server have SSL enabled.
- The configuration of a component in the graph specifies an Endeca Server that is not running.  
Check the status of the specified Endeca Server. If the Endeca Server is not running, start it.
- The configuration of a component in the graph specifies an Endeca data domain that is not running.  
Check the status of the specified Endeca data domain. If the Endeca data domain is not running, start it.

## Multi-assign delimiter error

A multi-assign delimiter must be specified when loading multi-assign data.

When loading multi-assign data, errors similar to the following may occur:

```
ERROR [SocketReader] - Received error message from server: Attempt to
add/replace record ProductID:34699 with unknown dimension value
"Red;Green" within dimension "ProductType"
ERROR [WatchDog] - Graph execution finished with error
ERROR [WatchDog] - Node ENDECA_BULK_ADD_OR_REPLACE_RECORDS0 finished
with status: ERROR
```

This error may occur for either of the following reasons:

- You attempted to process multi-assign data, but do not specify a multi-assign delimiter.  
Ensure that all components in the graph specify a multi-assign delimiter.
- You attempted to process multi-assign data, but specified different multi-assign delimiters in different components.  
Ensure that all components in the graph specify the same multi-assign delimiter.

In the example above, the multi-assign source is "Red;Green" (with the semicolon being the delimiter). To correct the problem, specify the correct multi-assign delimiter in the **Multi-assign delimiter** field of the component's configuration screen.





# Managing Studio Metadata

Metadata helps define the attributes, attribute groups, predefined metrics, and views in an Endeca Server data domain. Metadata may be included in physical records stored in an Endeca Server data domain, or defined as logical constructs as part of a view definition.

When metadata is included in physical attributes and attribute groups, it is stored as part of their Endeca Server definition record. Attributes are defined using Property Description Records (PDRs), and groups are defined using Group Description Records (GDRs). Physically stored metadata is managed using the Configuration Web Service.

When included in logical attributes and attribute groups defined as part of a view, metadata is stored as part of the view definition (Entity Description Record). Logically defined metadata is managed using the Entity Configuration Web Service.

Whether derived from physical records or logical views, Studio expects the same metadata properties and the same types of values. While the process of populating these values is different for physical records and logical constructs, the content should be the same for both.

Table C.1: Attribute metadata properties

Metadata property	Physical record name	View name	Description	Values
Available aggregation methods	<i>system-eid_available_aggregations</i>	<i>available_aggregations</i>	List of valid aggregation methods for the attribute.	For details, see <a href="#">Configuring aggregation methods for attributes on page 227</a>
Default aggregation method	<i>system-eid_default_aggregation</i>	<i>default_aggregation</i>	The default aggregation method selected when a user adds the attribute to a component as a metric, for example, when configuring a Chart or an aggregated Results Table.	mdex:string See also <a href="#">Configuring aggregation methods for attributes on page 227</a>

Metadata property	Physical record name	View name	Description	Values
Description	<i>system-eid_description</i>	<i>description</i>	Default description displayed for the attribute.  This value is used unless a locale-specific description is defined for the attribute.	mdex:string
Display format settings	<i>system-eid_formatSettings</i>	<i>formatSettings</i>	Defines any custom format properties for the attribute.	mdex:string; JSON string
Localized display names	<i>system-eid_localizedDn</i>	<i>localizedDn</i>	List of localized display names of the attribute.  For additional details, see <a href="#">Localizing display names and descriptions on page 236</a>	mdex:string; JSON string
Localized description	<i>system-eid_localizedDescription</i>	<i>localizedDescription</i>	List of locale-specific descriptions of the attribute.  For additional details, see <a href="#">Localizing display names and descriptions on page 236</a> .	mdex:string; JSON string
Is dimension?	<i>system-eid_isDimension</i>	<i>isDimension</i>	Specifies whether the attribute is available to be used as a dimension to aggregate metric values	mdex:boolean
Usage and display	<i>system-eid_approxCardinality</i>	<i>approxCardinality</i>	Specifies how Studio display and uses the attribute.	For details, see <a href="#">Configuring Studio usage and display of attributes on page 233</a>

Metadata property	Physical record name	View name	Description	Values
Date time granularity	<i>system-eid_datetimeFinestLevel</i>	<i>datetimeFinestLevel</i>	Used for date time attributes (mdex:dateTime) to specify the finest level of granularity allowed for the attribute.	For details see <a href="#">Date and time metadata properties on page 242</a>
Supported levels of date time granularity	<i>system-eid_datetimeCombosEnabled</i>	<i>datetimeCombosEnabled</i>	Use for date time attributes (mdex:dateTime) to specify the combinations of date grains available.	For details, see <a href="#">Date and time metadata properties on page 242</a>

Table C.2: Attribute group metadata properties

Metadata property	Physical record name	View name	Description	Values
Include in navigation	<i>system-eid_group_includeInNavigation</i>	<i>includeInNavigation</i>	Specifies whether the attribute group is displayed by default on the <b>Available Refinements</b> Studio component.	mdex:boolean
Include in record	<i>system-eid_group_includeInRecord</i>	<i>includeInRecord</i>	specifies whether the attribute group is displayed by default on the <b>Record Details</b> and <b>Compare</b> dialogs in Studio.	mdex:boolean

Metadata property	Physical record name	View name	Description	Values
Localized display name	<i>system-eid_group_localizedDn</i>	<i>localizedDn</i>	List of locale-specific display names for the attribute group.  For additional details, see <a href="#">Localizing display names and descriptions on page 236</a> .	mdex:string; JSON string

[Managing metadata for physical attributes and attribute groups](#)

[Managing metadata for views](#)

[Studio features and their associated metadata](#)

[Localization in Studio](#)

[Date and time metadata properties](#)

## Managing metadata for physical attributes and attribute groups

You must define metadata properties in the Endeca Server data domain before you can populate them with formatting, display names, and other metadata. Use the Configuration Web Service to populate these physical records in the Endeca Server data domain.

[Creating Information Discovery metadata properties](#)

[Assigning values to attribute metadata properties](#)

[Assigning values to attribute group configurations](#)

## Creating Information Discovery metadata properties

You must add a record for each Information Discovery attribute property you want to use.

Metadata properties are added to the Property Description Record (PDR). Use the `updateProperties` operation of the Configuration Web Service to add these records.

The sample SOAP request below illustrates a simple example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns:configTransaction xmlns:ns="http://www.endeca.com/MDEX/config/services/types/2/0" xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
      <ns:updateProperties>
        <ns1:record>
          <mdex-property_DisplayName>availableAggregations</mdex-property_DisplayName>
          <mdex-property_IsTextSearchable>>false</mdex-property_IsTextSearchable>
        </ns1:record>
      </ns:updateProperties>
    </ns:configTransaction>
  </soap:Body>
</soap:Envelope>
```

```

    <mdex-property_IsPropertyValueSearchable>>false</mdex-property_IsPropertyValueSearchable>
    <mdex-property_Type>mdex:string</mdex-property_Type>
    <mdex-property_Key>system-eid_available_aggregations</mdex-property_Key>
  </ns1:record>
  <ns1:record>
    <mdex-property_DisplayName>defaultAggregation</mdex-property_DisplayName>
    <mdex-property_IsTextSearchable>>false</mdex-property_IsTextSearchable>
    <mdex-property_IsPropertyValueSearchable>>false</mdex-property_IsPropertyValueSearchable>
    <mdex-property_Type>mdex:string</mdex-property_Type>
    <mdex-property_Key>system-eid_default_aggregation</mdex-property_Key>
  </ns1:record>
  <ns1:record>
    <mdex-property_DisplayName>localizedDisplayName</mdex-property_DisplayName>
    <mdex-property_IsTextSearchable>>false</mdex-property_IsTextSearchable>
    <mdex-property_IsPropertyValueSearchable>>false</mdex-property_IsPropertyValueSearchable>
    <mdex-property_Type>mdex:string</mdex-property_Type>
    <mdex-property_Key>system-eid_group_localizedDn</mdex-property_Key>
  </ns1:record>
</ns:updateProperties>
</ns:configTransaction>
</soap:Body>
</soap:Envelope>

```

Note that this example creates multiple attribute properties in one `updateProperties` operation. The request includes one `<record>` node for each attribute property created.

You only need to run the request to create attribute metadata once. If you repeat the request, it produces the same result and does not return an error.

## Assigning values to attribute metadata properties

Use the `updateProperties` operation of the Configuration Web Service to assign values to attribute metadata properties. Before you can assign values to properties, you must create the properties.

The sample SOAP request below illustrates a Web services request to assign values to extended attribute metadata properties for a record. This request assumes that the following attributes have already been created, as explained in [Creating Information Discovery metadata properties on page 220](#):

- `system-eid_isDimension`
- `system-eid_available_aggregations`
- `system-eid_default_aggregation`
- `system-eid_default_aggregation`
- `system-eid_formatSettings`
- `system-eid_localizedDn`

The request specifies the `mdex-property_Key` of the record to which to assign values.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <config-service:configTransaction xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/2/0">
      <config-service:updateProperties xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
        <mdex:record>
          <mdex-property_Key>DealerPrice</mdex-property_Key>
          <system-eid_isDimension type="mdex:boolean" xmlns="">>false</system-eid_isDimension>
          <system-eid_available_aggregations type="mdex:string">SUM,AVG,MIN,MAX,VARIANCE,STDDEV</system-eid_available_aggregations>
          <system-eid_default_aggregation type="mdex:string" xmlns="">AVG</system-eid_default_aggregation>
        </mdex:record>
      </config-service:updateProperties>
    </config-service:configTransaction>
  </soap:Body>
</soap:Envelope>

```

```

<system-eid_formatSettings>{"type":"CURRENCY",
@class":"com.endeca.portal.format.NumberFormatter", "currencySymbol":"$"}</system-eid_formatSettings>
<system-eid_localizedDn>{"de_DE":"Händlerpreis", "fr_FR":"Prix marchand", "es_ES":"Precio de
los concesionarios"}</system-eid_formatSettings>
<mdex:record>
</config-service:updateProperties>
</config-service:configTransaction>
</soap:Body>
</soap:Envelope>

```

This example code is derived from the Endeca Information Discovery Getting Started project. This project includes an example pipeline that assigns values to both extended metadata properties and primordial properties. In the `LoadConfiguration.grf` graph, see the Load Attribute Metadata pipeline.

## Assigning values to attribute group configurations

Use the `updateGroupConfigs` operation of the Configuration Web Service to add metadata properties to attribute group configurations. You can assign the value to the property at the same time that you add it to the attribute group configuration.

The sample SOAP request below illustrates a Web service request to assign a value to the `system-eid_group_localizedDn` property for the Products attribute group. This request also assigns a value to the `system-group_DisplayName` property.

This request assumes that the `system-eid_group_localizedDn` property has already been created. For details, see [Creating Information Discovery metadata properties on page 220](#).

For details about the value assigned in this example, see [Localizing display names and descriptions on page 236](#).

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://
/www.endeca.com/MDEX/config/services/types/2/0" xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery
/2009/09">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:configTransaction>
      <ns:updateGroupConfigs>
        <ns1:record>
          <system-group_DisplayName>Product</system-group_DisplayName>
          <system-group_Key>Product</system-group_Key>
          <system-eid_group_localizedDn>{"de_DE":"Produkt", "fr_FR":"Produit"}<
/system-eid_group_localizedDn>
        </ns1:record>
      </ns:updateGroupConfigs>
    </ns:configTransaction>
  </soapenv:Body>
</soapenv:Envelope>

```

## Managing metadata for views

Views are logical constructs that define a view of data relevant to a particular business problem or visualization.

View definitions include:

- The EQL expression that defines how the view is derived
- The attributes, attribute groups, and predefined metrics associated with the records exposed by the view

Metadata associated with these attributes and attribute groups is stored as part of the view definition in the Endeca Server data domain.

Use the Entity Configuration Web Service to populate the metadata properties of views.

The sample SOAP request below creates a view (semantic entity), and sets values for both default and Information Discovery properties:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns:putEntities xmlns:ns="http://www.endeca.com/endeca-server/sconfig/2/0">
      <ns:semanticEntity key="Products" displayName="Products" isActive="false"><ns:definition>
        /*Calculate Total Sales respecting Navigation*/
        /DEFINE GlobalSales2 as select sum(FactSales_SalesAmount) AS TotalSales Group
        ; DEFINE Products AS SELECT ProductSubcategoryName AS ProductSubcategoryName, ProductCategoryName AS
        ProductCategoryName, coalesce(ProductName, 'N
        /A') as ProductName, Description AS Description, Color as Color, arb(SurveyResponse) as
        "SurveyResponse", avg(FactSales_SalesAmount) AS AvgSales, sum(FactSales_SalesAmount) AS SalesSum,
        avg(FactSales_ProductStandardCost) AS AvgStandardCost, avg(ListPrice) AS AvgListPrice,
        avg(FactSales_UnitPrice) AS AvgUnitPrice, Avg(FactSales_OrderQuantity) as AvgQuantity,
        Sum(FactSales_SalesAmount-(FactSales_OrderQuantity*FactSales_ProductStandardCost)) as MonthlyProfit,
        Avg((FactSales_SalesAmount-(FactSales_OrderQuantity*FactSales_ProductStandardCost))
        /FactSales_OrderQuantity) as AvgMargin, sum(FactSales_SalesAmount)
        /GlobalSales2[.TotalSales as SalesShare, DimDate_FiscalYear*100
        +DimDate_MonthNumberOfYear as "Year-Month",
        DimDate_FiscalYear as DimDate_FiscalYear GROUP BY ProductName, "Year-Month"</ns:definition>
        <ns:description>This view is grouped to a year/month and product name.</ns:description>
        <ns:attributes>
          <ns:semanticAttribute name="Color" displayName="Color" datatype="mdex:string" isDimension
          ="true" isKeyColumn="false">
            <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
            <ns:property key="defaultAggregation"></ns:property>
            <ns:property key="localizedDn">{"de_DE":"Farbe" "fr_FR":"Couleur", "es_ES":"Color"}</ns:property>
          /ns:property>
          </ns:semanticAttribute>
          <ns:semanticAttribute name="Description" displayName="Description" datatype
          ="mdex:string" isDimension="true" isKeyColumn="false">
            <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
            <ns:property key="defaultAggregation"></ns:property>
            <ns:property key
            ="localizedDn">{"de_DE":"Beschreibung", "fr_FR":"Description", "es_ES":"Descripción"}</ns:property>
          </ns:semanticAttribute>
          <ns:semanticAttribute name="ProductCategoryName" displayName="Product Category Name" datatype
          ="mdex:string" isDimension="true" isKeyColumn="false">
            <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
            <ns:property key="defaultAggregation"></ns:property>
            <ns:property key
            ="localizedDn">{"de_DE":"Produktbezeichnung", "fr_FR":"Nom de catégorie", "es_ES":"Categorías des
            Productos"}</ns:property>
          </ns:semanticAttribute>
          <ns:semanticAttribute name="ProductName" displayName="Product Name" datatype
          ="mdex:string" isDimension="true" isKeyColumn="true">
            <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
            <ns:property key="defaultAggregation"></ns:property>
            <ns:property key
            ="localizedDn">{"de_DE":"Produktname", "fr_FR":"Nom de produit", "es_ES":"Nombre del producto"}</ns:property>
          </ns:semanticAttribute>
          <ns:semanticAttribute name="SalesShare" displayName="Sales Share" datatype
          ="mdex:double" isDimension="false" isKeyColumn="false">
            <ns:property key="availableAggregations">SUM,AVG,MEDIAN,MIN,MAX,VARIANCE,STDEV</ns:property>
          /ns:property>
            <ns:property key="defaultAggregation">SUM</ns:property>
            <ns:property key="formatSettings">{"type":"PERCENTAGE", "
            @class":"com.endeca.portal.format.NumberFormatter"}</ns:property>
            <ns:property key
            ="localizedDn">{"de_DE":"Umsatzanteil", "fr_FR":"Division de ventes", "es_ES":"División de Ventas"}</ns:property>
          /ns:property>
        </ns:semanticAttribute>
      </ns:semanticEntity>
    </ns:putEntities>
  </soap:Body>
</soap:Envelope>
```

```

        </ns:semanticAttribute>
        <ns:semanticAttribute name="SalesSum" displayName="Sales Sum" datatype
="mdex:double" isDimension="false" isKeyColumn="false">
          <ns:property key="availableAggregations">SUM,AVG,MEDIAN,MIN,MAX,VARIANCE,STDDEV<
/ns:property>
          <ns:property key="defaultAggregation">SUM</ns:property>
          <ns:property key
="localizedDn">{"de_DE":"Umsatz", "fr_FR":"somme de ventes", "es_ES":"Ventas totales"}</ns:property>
          <ns:property key="formatSettings">{"type":"CURRENCY", "
@class":"com.endeca.portal.format.NumberFormatter", "currencySymbol":"$"}</ns:property>
        </ns:semanticAttribute>
        <ns:semanticAttribute name="SurveyResponse" displayName="Survey Response" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
          <ns:property key="availableAggregations">COUNT,ARB</ns:property>
          <ns:property key="defaultAggregation">ARB</ns:property>
          <ns:property key
="localizedDn">{"de_DE":"Umfrageantwort", "fr_FR":"Réponse à l'enquête", "es_ES":"Respuesta a la
encuesta"}</ns:property>
        </ns:semanticAttribute>
      </ns:attributes>
      <ns:metrics/>
      <ns:groups>
        <group displayName="Products" key="Products">
          <semanticAttributeKey name="ProductName"/>
          <semanticAttributeKey name="ProductCategoryName"/>
          <semanticAttributeKey name="Color"/>
          <semanticAttributeKey name="Description"/>
          <property key="includeInNavigation">true</property>
          <property key="includeInRecord">true</property>
          <property key="localizedDn">{"de_DE":"Produkte", "fr_FR":"Produits", "es_ES":"Productos"}<
/property>
        </group>
      </ns:groups>
    </ns:semanticEntity>
  </ns:putEntities>
</soap:Body>
</soap:Envelope>

```

This example shows a view definition (stored in Endeca Server as a Semantic Entity) being stored in an Endeca Server data domain. This view definition includes:

Table C.3: Example view definition elements

View property	Description
definition	The EQL definition that derives the view from physical data stored in the Endeca Server data domain
Key	The unique identifier of the view
displayName	The name of the view
description	The description of the view
attributes	The list of attributes associated with the view, as well as all of the metadata for those attributes



View property	Description
metrics	The list of predefined metrics associated with the view, and all of the metadata for those metric attributes
groups	The list of groups associated with the view, and all of the metadata for those groups

## Studio features and their associated metadata

Metadata is associated with specific Studio features.

[General attribute metadata](#)

[General attribute group metadata](#)

[Aggregation method metadata properties](#)

[Display formats for attributes](#)

[Configuring Studio usage and display of attributes](#)

### General attribute metadata

General attribute metadata includes the text description of the attribute and an indication of whether the attribute can be used as a dimension to aggregate metric values.

The description can be localized. For details, see [Localizing display names and descriptions on page 236](#).

Table C.4: General attribute metadata physically stored in the data domain

Property	Description	Type of value
system-eid_description	Text description of the attribute	Text string
system-eid_isDimension	Whether the attribute can be used as a dimension for aggregation	Boolean (true or false)

Use the Configuration Web Service to populate these metadata properties.

Table C.5: General attribute metadata specified in logical views

Property	Description	Type of value
description	Text description of the attribute	Text string
isDimension	Whether the attribute can be used as a dimension for aggregation	Boolean (true or false)

Use the Entity Configuration Web Service to populate these metadata properties.

## General attribute group metadata

General attribute group metadata includes indications of whether to display the attribute group by default on the **Available Refinements** component and whether to display the attribute group by default on the **Record Details** and **Compare** dialogs.

Table C.6: General attribute group metadata physically stored in the data domain

Property	Description	Type of value
system-eid_group_includeInNavigation	Whether to include the attribute group by default in <b>Available Refinements</b> components	Boolean (true or false)
system-id_group_includeInRecord	Whether to include the attribute group by default in <b>Record Details</b> and <b>Compare</b> dialogs	Boolean (true or false)

Use the Configuration Web Service to populate these metadata properties.

Table C.7: General attribute group metadata specified in logical views

Property	Description	Type of value
includeInNavigation	Whether to include the attribute group by default in <b>Available Refinements</b> components	Boolean (true or false)
includeInRecord	Whether to include the attribute group by default in <b>Record Details</b> and <b>Compare</b> dialogs	Boolean (true or false)

Use the Entity Configuration Web Service to populate these metadata properties.

## Aggregation method metadata properties

Aggregation method metadata includes the available aggregation methods for each attribute, as well as the default aggregation method.

Table C.8: Physical aggregation metadata properties

Property name	Description
system-eid_available_aggregations	A list of valid aggregation methods for the property. For example: SUM, AVG, MEDIAN, MIN, MAX, VARIANCE, STDDEV
system-eid_default_aggregation	A single default aggregation method used when the property is added as a metric to a Studio component. For example: AVG

Table C.9: Logical aggregation metadata properties

Property name	Description
available_aggregations	A list of valid aggregation methods for the property. For example: SUM, AVG, MEDIAN, MIN, MAX, VARIANCE, STDDEV
default_aggregation	A single default aggregation method used when the property is added as a metric to a Studio component. For example: AVG

For an example of an Entity Configuration Web Service request that populates these properties, see [Managing metadata for views on page 222](#).

## Display formats for attributes

The display format options available for attributes depend on the data type of the attribute.

[Studio features and their associated metadata](#)

[Display format metadata properties](#)

[About the formatting specification](#)

[Boolean display format settings](#)

[Date/Time display format settings](#)

[Duration display format settings](#)

[Geocode display format settings](#)

[Number display format settings](#)

[Time display format settings](#)

## Display format metadata properties

To define display formatting for attributes physically stored in the Endeca Server data domain, Studio uses the `system-eid_formatSettings` metadata property. The value of this property is a JSON string.

For an example of a Web service request that populates this property, see [Assigning values to attribute metadata properties on page 221](#).

To define display formatting for attributes and predefined metrics defined as part of a view definition, Studio uses the `formatSettings` metadata property.

For an example of an Entity Configuration Web Service request that populates this property, see [Managing metadata for views on page 222](#).

## About the formatting specification

The default display format for an attribute is specified using a JSON string that contains a formatting specification.

The formatting specification must be appropriate to the data type of the attribute. For example, formatting specifications for date/time data types are not appropriate for Boolean data types.

See also "Configuring the default display format for an attribute" in the *Oracle Endeca Information Discovery Studio User's Guide*.





**Note:** Values in JSON strings require JSON escaping. For information on JSON syntax and working with JSON in general, see <http://json.org>.

## Boolean display format settings

For the Boolean display format, the metadata includes the values to use for true and false.

Table C.10: Boolean display format settings

Formatting option	Values
<code>type</code>	BOOLEAN
<code>useDefaultValues</code>	DEFAULT CUSTOM

Formatting option	Values
customTrueValue  <b>Note:</b> Only valid if the value of useDefaultValues is CUSTOM	String to display if the value of the attribute is TRUE
customFalseValue  <b>Note:</b> only valid if the value of useDefaultValues is CUSTOM	String to display if the value of the attribute is FALSE

### Example Boolean display format JSON

```
{
  "@class": "com.endeca.portal.format.BooleanFormatter",
  "type": "BOOLEAN",
  "useDefaultValues": "CUSTOM",
  "customTrueValue": "Yes",
  "customFalseValue": "No"
}
```

### Date/Time display format settings

For date/time values, the display format metadata includes the formats for the date and the time.

Table C.11: Date/Time display format settings

Formatting option	Values
type	DATETIME
dateDisplayFormat	DEFAULT SHORT_NUM LONG_NUM SHORT_TXT LONG_TXT FULL
timeDisplayFormat	DEFAULT SHORT LONG NONE

### Example date/time display format JSON

```
{
"@class": "com.endeca.portal.format.DateTimeFormatter",
"type": "DATETIME",
"dateDisplayFormat": "LONG_TXT",
"timeDisplayFormat": "LONG"
}
```

### Duration display format settings

For duration values, the display format settings include the precision level, whether to display the unit, and the decimal places.

Table C.12: Duration display format settings

Formatting option	Values
type	DUR
precision	DEFAULT DAY HOU MIN SEC MIL
includePrecisionUnit	DEFAULT YES NO
decimalPlacesMode	DEFAULT CUSTOM
decimalPlaces	Positive integers

### Example duration display format JSON

```
{
"@class": "com.endeca.portal.format.DurationFormatter",
"type": "DUR",
"precision": "MIL",
"includePrecisionUnit": "YES",
"decimalPlacesMode": "CUSTOM",
"decimalPlaces": 1
}
```

## Geocode display format settings

For geocode data, the display format metadata controls the decimal places.

Table C.13: Geocode display format settings

Formatting option	Values
type	GEOCODE
decimalPlacesMode	DEFAULT CUSTOM
decimalPlaces	Positive integers

### Example geocode display format JSON



```
{
  "@class": "com.endeca.portal.format.GeocodeFormatter",
  "type": "GEOCODE",
  "decimalPlacesMode": "CUSTOM",
  "decimalPlaces": 2
}
```

## Number display format settings

For numeric data, the display format metadata includes whether the number is a currency or percentage, the decimal places, and the grouping separator.

Table C.14: Number display format settings

Formatting option	Values
type	NUMBER CURRENCY PERCENTAGE
includeGroupingSeparator	DEFAULT YES NO
decimalPlacesMode	DEFAULT AUTO CUSTOM

Formatting option	Values
decimalPlaces	Positive integers
numberFormat	DEFAULT STANDARD ACCOUNTING
decimalSeparator	DEFAULT COMMA PERIOD
groupingSeparator	DEFAULT COMMA PERIOD SPACE APOSTROPHE
currencySymbol  <b>Note:</b> Only valid if the value of type is CURRENCY	\$ € £ ¥ ₩ NT\$ AUD CAN CNY EUR GBP JPY KRW TWD USD
includePercentageSign  <b>Note:</b> Only valid if the value of type is PERCENTAGE	DEFAULT YES NO

### Example number display format JSON: set decimal places

```
{
"@class": "com.endeca.portal.format.NumberFormatter",
"type": "NUMBER",
"decimalPlacesMode": "CUSTOM",
"decimalPlaces": 4,
"decimalSeparator": "DEFAULT",
"groupingSeparator": "DEFAULT",
"includeGroupingSeparator": "DEFAULT",
"numberFormat": "DEFAULT"
}
```

### Example number display format JSON: set currency and decimal places

```
{
"@class": "com.endeca.portal.format.NumberFormatter",
"type": "CURRENCY",
"decimalPlacesMode": "CUSTOM",
"decimalPlaces": 3,
"decimalSeparator": "DEFAULT",
"groupingSeparator": "DEFAULT",
}
```



```
"includeGroupingSeparator": "DEFAULT",
"numberFormat": "DEFAULT",
"currencySymbol": "€"
}
```

### Example number display format JSON - set percentage and decimal places

```
{
"@class": "com.endeca.portal.format.NumberFormatter",
"type": "PERCENTAGE",
"decimalPlacesMode": "CUSTOM",
"decimalPlaces": 1,
"decimalSeparator": "DEFAULT",
"groupingSeparator": "DEFAULT",
"includeGroupingSeparator": "DEFAULT",
"numberFormat": "DEFAULT",
"includePercentageSign": "DEFAULT"
}
```

## Time display format settings

For time data, the display format metadata controls whether to use the short or long time format.

Table C.15: Time display format settings

Formatting option	Values
type	DATETIME
dateDisplayFormat	DEFAULT SHORT LONG

### Example time display format JSON

```
{
"@class": "com.endeca.portal.format.TimeFormatter",
"type": "TIME",
"timeDisplayFormat": "LONG"
}
```

## Configuring Studio usage and display of attributes

You can define properties Studio can use to determine how to display and process attributes.

For example, some numeric data falls within a range of possible values and can effectively be displayed in a range filter. Other numeric data, however, is not meaningful within a range and should be displayed as a list of values.

To configure the usage and display of physical attribute data, use the following values:

Table C.16: Usage and display properties for physical attributes and pre-defined metrics

Property	Description
system-eid_approxCardinality	<p>Valid values include:</p> <ul style="list-style-type: none"> <li>• high The value is a range and should be displayed using a range filter</li> <li>• low The value is not a range, and should be displayed as a list of values.</li> <li>• not applicable Neither a range nor a list of values is an appropriate display of the data. Usually affects data such as geocodes that is neither a range nor a list, and is processed in some other way.</li> </ul>

Use the Configuration Web Service to populate these metadata properties.

To configure the usage and display of attributes and predefined metrics defined in a view definition, use the following metadata properties:

Table C.17: Usage and display properties for logical attributes and pre-defined metrics

Property	Description
approxCardinality	<p>Valid values include:</p> <ul style="list-style-type: none"> <li>• high The value is a range and should be displayed using a range filter</li> <li>• low The value is not a range, and should be displayed as a list of values.</li> <li>• not applicable Neither a range nor a list of values is an appropriate display of the data. Usually affects data such as geocodes that is neither a range nor a list, and is processed in some other way.</li> </ul>

Use the Entity Configuration Web Service to populate these metadata properties.

## Localization in Studio

Users can change their locale in Studio

Studio supports the following locales:

Table C.18: Supported locales in Studio

Studio locale	Endeca Server locale
de_DE	de
en_US	en
es_ES	es
fr_FR	fr
it_IT	it
ja_JP	ja
ko_KR	ko
pt_PT	pt
zh_CN	zh_CN
zh_TW	zh_TW

When a user changes to a supported locale, Studio applies any localization you have defined for that locale. The following localization options are available:

- attributes
  - display name and description
 

Localize the display name and description if the data of the attribute does not change when the locale changes. For example, numeric counts remain the same from one locale to another.
  - the attribute itself
 

Localize the attribute if the value of the attribute does change when the locale changes. For example, if the value of the attribute is a short string, such as a color, or a monetary unit, you might want to localize the value of the attribute.
- pre-defined metrics
  - display name and description
- views
  - display name and description
- attribute groups

display name only

[Localizing display names and descriptions of attributes and predefined metrics](#)

[Localizing attribute group display names](#)

[Localizing view display names and descriptions](#)

[Localizing attributes](#)

## Localizing display names and descriptions of attributes and predefined metrics

You can localize the display name and description of attributes and predefined metrics.

To localize the display names and descriptions of attributes and predefined metrics physically stored in an Endeca Server data domain, use the following metadata properties:

Table C.19: Localization properties for physical attributes and pre-defined metrics

Property	Description
system-eid_localizedDn	List of localized display names of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized display name for locale 1",   "locale2": "localized display name for locale 2" }</pre>
system-eid_localizedDescription	List of localized descriptions of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized description for locale 1",   "locale2": "localized description for locale 2" }</pre>

Use the Configuration Web Service to populate these metadata properties.

For an example of a Web service request that illustrates how to populate a localization metadata property, see [Assigning values to attribute metadata properties on page 221](#).

To localize the display name of attributes and predefined metrics defined in a view definition, use the following metadata properties:

Table C.20: Localization properties for logical attributes and pre-defined metrics

Property	Description
localizedDn	List of localized display names of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized display name for locale 1",   "locale2": "localized display name for locale 2" }</pre>

Property	Description
localizedDescription	List of localized descriptions of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized description for locale 1",   "locale2": "localized description for locale 2" }</pre>

Use the Entity Configuration Web Service to populate these metadata properties.

For an example of an Entity Configuration Web Service request that populates localization properties, see [Managing metadata for views on page 222](#).

## Localizing attribute group display names

You can localize the display names of attribute groups.

To localize the display names of attribute groups physically stored in an Endeca Server data domain, use:

Table C.21: Localization properties for physical attribute groups

Property	Description
system-eid_localizedDn	List of localized display names of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized display name for locale 1",   "locale2": "localized display name for locale 2" }</pre>

Use the Configuration Web Service to populate this metadata properties.

For an example of a Web service request that illustrates how to populate a localization metadata property, see [Assigning values to attribute metadata properties on page 221](#).

To localize the display name of attribute groups defined in a view definition, use the following metadata property:

Table C.22: Localization properties for logical attribute groups

Property	Description
localizedDn	List of localized display names of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized display name for locale 1",   "locale2": "localized display name for locale 2" }</pre>

Use the Entity Configuration Web Service to populate these metadata properties.

For an example of an Entity Configuration Web Service request that populates localization properties, see [Managing metadata for views on page 222](#).

## Localizing view display names and descriptions

You can localize the display names and descriptions of views.

To localize view display names and descriptions, use the following properties:

Table C.23: Localization properties for views

Property	Description
localizedDn	List of localized display names of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized display name for locale 1",   "locale2": "localized display name for locale 2" }</pre>
localizedDescription	List of localized descriptions of the attribute or predefined metric as a JSON string in the format: <pre>{ "locale1": "localized description for locale 1",   "locale2": "localized description for locale 2" }</pre>

Use the Entity Configuration Web Service to populate these metadata properties.

For an example of an Entity Configuration Web Service request that populates localization properties, see [Managing metadata for views on page 222](#).

## Localizing attributes

Localize attributes when the data of the attribute is different in different locales.

To localize an attribute:

1. Define the attribute for the default locale.
2. Define a locale-specific version of the attribute for each locale to which you want to localize the attribute.

In some cases, the same string may represent the same concept in different languages. Therefore, when defining the key for the locale-specific attributes, you should use a naming convention that ensures that each key is unique. For example, append the locale string to the attribute key from the default locale.

3. Add the `system-eid_localizedAttribute` metadata property to the attribute in the default locale. The value of the property is a JSON string that specifies the locale-specific attribute for each locale. For example:  

```
{ "fr": "attributeKey_fr", "es": "attributeKey_es", "it": "attributeKey_it", "de": "attributeKey_de" }
```

## Localizing attributes

In this example, we are supporting the following locales:

- German (de\_DE)
- French (fr\_FR)
- Spanish (es\_ES)

We have an attribute Color defined in the default locale (en-us). The value of this attributes is a short string that will be different in different locales, so we want to localize the attribute. In addition to defining the Color attribute, we define the following attributes:

Table C.24: Example localized attributes

Attribute Key	Display Name
Color_de_DE	Farbe
Color_fr_FR	Couleur
Color_es_ES	Color

We then add the `system-eid_localizedAttribute` metadata property to the Color attribute in the default locale. The value of this attribute is `{"fr_FR": "Color_fr_FR", "es_ES": "Color_es_ES", "de_DE": "Color_de_DE"}`

In the example view definition below, the Color and Description attributes have been localized.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns:putEntities xmlns:ns="http://www.endeca.com/endeca-server/sconfig/2/0">
      <ns:semanticEntity key="Products" displayName="Products" isActive="false"><ns:definition>
/*Calculate Total Sales respecting Navigation*
/DEFINE GlobalSales2 as select sum(FactSales_SalesAmount) AS TotalSales Group
; DEFINE Products AS SELECT ProductSubcategoryName AS ProductSubcategoryName, ProductCategoryName AS
ProductCategoryName, coalesce(ProductName, 'N
/A') as ProductName, Description AS Description, Color as Color, arb(SurveyResponse) as
"SurveyResponse", avg(FactSales_SalesAmount) AS AvgSales, sum(FactSales_SalesAmount) AS SalesSum,
avg(FactSales_ProductStandardCost) AS AvgStandardCost, avg(ListPrice) AS AvgListPrice,
avg(FactSales_UnitPrice) AS AvgUnitPrice, Avg(FactSales_OrderQuantity) as AvgQuantity,
Sum(FactSales_SalesAmount-(FactSales_OrderQuantity*FactSales_ProductStandardCost)) as MonthlyProfit,
Avg((FactSales_SalesAmount-(FactSales_OrderQuantity*FactSales_ProductStandardCost))
/FactSales_OrderQuantity) as AvgMargin, sum(FactSales_SalesAmount)
/GlobalSales2[.TotalSales as SalesShare, DimDate_FiscalYear*100
+DimDate_MonthNumberOfYear as "Year-Month",
DimDate_FiscalYear as DimDate_FiscalYear GROUP BY ProductName, "Year-Month"</ns:definition>
      <ns:description>This view is grouped to a year/month and product name.</ns:description>
      <ns:attributes>
        <ns:semanticAttribute name="Color_de_DE" displayName="Farbe" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
          <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
          <ns:property key="defaultAggregation"></ns:property>
        </ns:semanticAttribute>
        <ns:semanticAttribute name="Color_fr_FR" displayName="Couleur" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
          <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
          <ns:property key="defaultAggregation"></ns:property>
        </ns:semanticAttribute>
        <ns:semanticAttribute name="Color_es_ES" displayName="Color" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
```

```

    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="Color" displayName="Color" datatype="mdex:string" isDimension
="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
    <ns:property key
="localizedAttributeMetadata">{"de_DE":"Color_de_DE", "fr_FR":"Color_fr_FR", "es_ES":"Color_es_ES"}<
/property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="Description_de_DE" displayName="Beschreibung" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="Description_fr_FR" displayName="Description" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="Description_es_ES" displayName="Descripción" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="Description" displayName="Description" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
    <ns:property key
="localizedAttributeMetadata" >{"de_DE":"Description_de_DE", "fr_FR":"Description_fr_FR",
"es_ES":"Description_es_ES"}</ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="ProductCategoryName" displayName="Product Category Name" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
    <ns:property key
="localizedDn">{"de_DE":"Produktbezeichnung", "fr_FR":"Nom de catégorie", "es_ES":"Categorías des
Productos"}</ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="ProductName" displayName="Product Name" datatype
="mdex:string" isDimension="true" isKeyColumn="true">
    <ns:property key="availableAggregations">COUNT,COUNTDISTINCT</ns:property>
    <ns:property key="defaultAggregation"></ns:property>
    <ns:property key
="localizedDn">{"de_DE":"Produktname", "fr_FR":"Nom de produit", "es_ES":"Nombre del producto"}<
/property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="SalesShare" displayName="Sales Share" datatype
="mdex:double" isDimension="false" isKeyColumn="false">
    <ns:property key="availableAggregations">SUM,AVG,MEDIAN,MIN,MAX,VARIANCE,STDDEV<
/property>
  </ns:property>
    <ns:property key="defaultAggregation">SUM</ns:property>
    <ns:property key="formatSettings">{"type":"PERCENTAGE", "
@class":"com.endeca.portal.format.NumberFormatter"}</ns:property>
    <ns:property key
="localizedDn">{"de_DE":"Umsatzanteil", "fr_FR":"Division de ventes", "es_ES":"División de Ventas"}<
/property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="SalesSum" displayName="Sales Sum" datatype
="mdex:double" isDimension="false" isKeyColumn="false">
    <ns:property key="availableAggregations">SUM,AVG,MEDIAN,MIN,MAX,VARIANCE,STDDEV<
/property>
  </ns:property>
    <ns:property key="defaultAggregation">SUM</ns:property>
    <ns:property key
="localizedDn">{"de_DE":"Umsatz", "fr_FR":"somme de ventes", "es_ES":"Ventas totales"}</ns:property>

```



```


    <ns:property key="formatSettings">{"type":"CURRENCY", "
@class":"com.endeca.portal.format.NumberFormatter", "currencySymbol":"$"}</ns:property>
  </ns:semanticAttribute>
  <ns:semanticAttribute name="SurveyResponse" displayName="Survey Response" datatype
="mdex:string" isDimension="true" isKeyColumn="false">
    <ns:property key="availableAggregations">COUNT,ARB</ns:property>
    <ns:property key="defaultAggregation">ARB</ns:property>
    <ns:property key
="localizedDn">{"de_DE":"Umfrageantwort", "fr_FR":"Réponse à l'enquête", "es_ES":"Respuesta a la
encuesta"}</ns:property>
  </ns:semanticAttribute>
</ns:attributes>
<ns:metrics/>
<ns:groups>
  <group displayName="Products" key="Products">
    <semanticAttributeKey name="ProductName"/>
    <semanticAttributeKey name="ProductCategoryName"/>
    <semanticAttributeKey name="Color"/>
    <semanticAttributeKey name="Description"/>
    <property key="includeInNavigation">true</property>
    <property key="includeInRecord">true</property>
    <property key="localizedDn">{"de_DE":"Produkte", "fr_FR":"Produits", "es_ES":"Productos"}<
/property>
  </group>
</ns:groups>
</ns:semanticEntity>
</ns:putEntities>
</soap:Body>
</soap:Envelope>

```

## Date and time metadata properties

Date and time metadata properties define the available levels of granularity for date and time data, and the finest level of granularity

Table C.25: Date and time metadata physically stored in the data domain

Property	Description	Type of value
system-eid_datetimeFinestLevel	<p>Specifies the finest level of granularity allowed for the attribute.</p> <p>The following values are valid:</p> <ul style="list-style-type: none"> <li>• YEAR</li> <li>• MONTH</li> <li>• DAY_OF_MONTH</li> <li>• HOUR</li> <li>• MINUTE</li> <li>• SECOND</li> </ul> <p> <b>Note:</b> Case is significant. The value of this property must be specified in ALL CAPS.</p>	Text string
system-eid_datetimeCombosEnabled	Specifies the combinations of date grains available for the attribute.	<p>mdex:string</p> <p>A JSON array of JSON arrays.</p> <p>For example:</p> <pre>[ [ "YEAR" ], [ "YEAR", "MONTH" ], [ "YEAR", "MONTH", "DAY_OF_MONTH" ] ]</pre>

Use the Configuration Web Service to populate these metadata properties.



---

# Managing Collections for Studio

The *collections* feature in Endeca Server is called *data sets* in Studio.

Recommended practice when specifying the value of the collection key (the name of the collection) is to use a string that is human readable, meaningful, and relevant to the collection data. For example, use terms such as "Products" or "Claims". These terms are easy for users to remember and to enter when defining EQL queries on collection data.

Avoid using abstract strings that have no meaning to users, such as job IDs. These string are difficult to remember and to enter when defining EQL queries.

[Creating and updating collections](#)

[Collection key and attribute names](#)

[Prepending collection keys to attribute names in metadata](#)

## Creating and updating collections

The collection key and Spec Attribute are required properties of Information Discovery components that write to a data domain, such as the **Bulk Add/Update Records** component or the **Merge Records** component.

The Spec attribute is a standard attribute. It is a primary key, so it should be configured as unique and single-assign. Configure this attribute when configuring standard attributes for your data domain prior to loading.

It is possible to create the collection during data store configuration as well, but it is not necessary. Collection properties, such as the display name and description, can be modified after the collection is created and records have been added to the collection. Therefore, you can create the collection when loading data, and update the collection properties later.

To create a collection when you load data, specify the **Collection key** and **Spec attribute** when configuring the load component. Note that all subsequent loads to the same collection must specify the same value for the **Spec attribute** property. If you specify a different property, the graph will fail.

When updating collection properties, all properties are initially deleted before the update. Therefore you must preserve the following properties and reload them when updating collection properties. Studio requires these collection properties.

- *system-eid-ds\_createDate*
- *system-eid-ds\_lastLoadDate*
- *system-eid-ds\_baseFilters*

## Collection key and attribute names

The recommended practice is to prepend the collection key to the name of all attributes in a collection.

Collections cannot share attributes. An attribute can only belong to one collection at a time. Very often, however, you will want to include an attribute in multiple collections. For example, suppose you were working with auto insurance data. This data includes two collections, one for claims and one for work orders. Both collections might include such data as claim numbers and vehicle identification numbers (VINs). You would need to upload this data to each collection.

Studio requires that all attributes have unique names. To ensure that the name of each property is unique to its collection, recommended practice is to prepend the collection name to the property name. When loading data using Integrator ETL, the recommended separator when prepending the collection key to the attribute name is an underscore: `collectionKey_attributeName`. For example: `Claims_ClaimNumber`, `Claims_VIN`, `WorkOrders_ClaimNumber`, `WorkOrders_VIN`.

When loading data using the **Bulk Add/Replace Records** component, recommended practice is to check the **Prefix Attributes With Collection Key** box. When this box is checked, the collection key name is automatically prepended to each attribute name when loading data.

When loading data using other writer components:

1. Create a new metadata with the collection key prepended to each field name in the metadata.

You can implement a graph to prepend the collection key to the field names. See [Prepending collection keys to attribute names in metadata on page 244](#) for details about this graph.

2. Add a **Reformat** component to your graph immediately before the writer component.

In the **Reformat** component, use the `copyByPosition()` function to transform the input attribute names to the names with the collection key prepended:

```
copyByPosition($out.0, $in.0);
```

3. Add a new edge from the **Reformat** component to the writer component.
4. Apply the new metadata created in step 1 to the edge between the **Reformat** component and the writer component.

## Prepending collection keys to attribute names in metadata

You can use a graph to prepend collection keys to attribute names in the metadata.

To prepend collection keys to attribute names in a graph:

1. Create a new graph.

You can create the new graph anywhere in your Integrator ETL workspace, as you may want to reuse it. You do not have to create it in a specific project.

2. On the **Source** tab of the graph, replace the `<Global>` with the following `<Global>` node code:

```
<Global>
<Metadata id="Metadata0" previewAttachmentCharset="ISO-8859-1">
<Record fieldDelimiter="|" name="row" previewAttachmentCharset="ISO-8859-1" recordDelimiter
="\n" type="delimited">
<Field delimiter="\r" name="row" type="string"/>
</Record>
</Metadata>
```

```

<Property fileURL="workspace.prm" id="GraphParameter0"/>
<Note alignment="1" backgroundColorB="225" backgroundColorG="255" backgroundColorR
="255" folded="false" height="177" id="Note0" textColorB="0" textColorG="0" textColorR
="0" textFontSize="8" title="Description" titleColorB="0" titleColorG="0" titleColorR
="0" titleFontSize="10" width="824" x="20" y="56">
<attr name
="text"><![CDATA[This graph reads metadata from a single external source. It adds a prefix
and rewrites the metadata to a new .fmt]]></attr>
</Note>
<Dictionary/>
</Global>
<Phase number="0">
<Node enabled="enabled" fileURL="{META_DIR}/no_prefix.fmt" guiHeight="87" guiName
="DATA_READER0" guiWidth="128" guiX="30" guiY="126" id="DATA_READER0" type="DATA_READER"/>
<Node enabled="enabled" guiHeight="65" guiName="Filter Header" guiWidth="128" guiX="255" guiY
="126" id="EXT_FILTER0" type="EXT_FILTER">
<attr name="filterExpression"><![CDATA[ $0.row =~ "<Field.*>" ]]></attr>
</Node>
<Node enabled="enabled" guiHeight="65" guiName="Prefix" guiWidth="128" guiX="480" guiY
="126" id="REFORMAT0" type="REFORMAT">
<attr name="transform"><![CDATA[// #CTL2

// Transforms input record into output record.
function integer transform() {
    $out.0.row = replace($0.row,"name=\"", "name=\"Vehicle_");

    return ALL;
}
]]></attr>
</Node>
<Node enabled="enabled" fileURL="{META_DIR}/prefix_after.fmt" footer="&lt;/Record&gt
;" guiHeight="87" guiName="Create Metadata" guiWidth="128" guiX="705" guiY="126" header="&lt;
?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;&#10;&lt;Record fieldDelimiter
=&quot;|&quot; name=&quot;ProductCategory&quot; previewAttachmentCharset=&quot
;ISO-8859-1&quot; recordDelimiter=&quot;\r\n&quot; type=&quot;delimited&quot;&gt;&#10;" id
="STRUCTURE_WRITER0" mask="$row \n" type="STRUCTURE_WRITER"/>
<Edge debugMode="true" fromNode="DATA_READER0:0" guiBendpoints="" guiRouter="Manhattan" id
="Edge1" inPort="Port 0 (in)" metadata="Metadata0" outPort="Port 0 (output)" toNode
="EXT_FILTER0:0"/>
<Edge debugMode="true" fromNode="EXT_FILTER0:0" guiBendpoints="" guiRouter="Manhattan" id
="Edge3" inPort="Port 0 (in)" metadata="Metadata0" outPort="Port 0 (accepted)" toNode
="REFORMAT0:0"/>
<Edge debugMode="true" fromNode="REFORMAT0:0" guiBendpoints="" guiRouter="Manhattan" id
="Edge5" inPort="Port 0 (Body port)" metadata="Metadata0" outPort="Port 0 (out)" toNode
="STRUCTURE_WRITER0:0"/>
</Phase>

```

3. Save the graph.
4. Edit the **Date Reader0** data reader component. In the **File URL** property, specify the path to the metadata file with the attribute names to which you want to prepend the collection key.

If the metadata is in a different project than the graph, be sure you specify the fully-qualified path to the metadata file.



**Note:** You may want to give this metadata file a name that indicates that the collection key is not prepended to the attribute names..

5. Edit the **Prefix Reformat** component. In the **Transform** property, modify the following line: `$out.0.row = replace($0.row, "name=\"", "name=\"Vehicle_");`. Replace `Vehicle_` with the collection key of the collection in the graph for which you are creating the metadata.

For example, if the collection key is "Insurance", the updated code would be `$out.0.row = replace($0.row, "name=\"", "name=\"Insurance_");`

6. Edit the **Create Metadata** structure writer component. In the **File URL** property, specify the path and name of the new metadata file you want to create.

In the metadata is in a different project that the prepend graph, be sure you specify the fully-qualified path to the metadata file.



**Note:** You may want to give this metadata file a name that indicates that the collection key has been prepended to the attribute names.

7. Save your changes.
8. Run the graph.

A new metadata is created with the name you specified. In the metadata, the collection key you specified is appended to the attribute names.

You can re-use this graph to create additional metadata. For each new metadata you want to create, repeat Steps 3 through 8.

# Index

## A

- adding records and assignments 53
  - input data format 54
- Add KVPs component
  - batch size adjustments 208
  - reference details 152
- aggregation edge, creating 85
- assignments adding to records 53
- attribute names, prepending collection keys to 244
- attribute schema
  - configuration input file 67
  - loading 67
  - managed attributes input file 69
- attribute schema load, about 66

## B

- backing up data domains 59
- batch size adjustments 208
- Boolean display format metadata 228
- Bulk Add/Replace Records component
  - choosing 47
  - configuring for initial load 52
  - reference details 156
- Bulk Loader, using 30

## C

- checking output 28
- Chinese, text enrichment for 148
- Clover Log 28
- collection key prepended to attribute names 244
- Collections
  - adding to OBI project 119
  - attribute names and 244
  - creating and updating 243
  - Studio 243
- Commit Transaction graph 100
- Common configuration properties for components 206
- components
  - adding data to 22
  - adding to graph 21
  - connecting 26
- configuration
  - data domain 34
  - exporting 87
  - importing 87
  - Integrator ETL 13
- configuration graphs

- CSV-based 86
  - xml-based 84
- configuring a data domain 63
- connection errors 215
- console window 28
- CSV-based configuration graphs 86
- Custom properties 39

## D

- data domain
  - backing up and restoring 59
  - configuration process 34
  - configuring 63
  - configuring as iterative process 34
  - creating 44
  - deleting data from 59
  - exporting configuration 87
  - exporting configurations 86
  - importing configuration 87
  - importing configurations 86
  - loading and maintaining data 45
- data, sending to Endeca data domain 30
- data sets
  - See Collections
- data types, supported 36
- date/time display format metadata 229
- DDR
  - See Dimension Description Records
- debugging the graph 29
- Delete Records component
  - batch size adjustments 208
  - defining the Record Set Specifier Attributes 58
  - reference details 163
- deleting data 59
  - Delete Records graph 61
  - records 59
- delimiters, specifying multiple 43
- descriptions, localizing 236, 237, 238
- Designer
  - See Integrator ETL
- detecting language 149
- Dimension Description Record 39
  - configuring 66, 69
- dimsearch\_config 77
- display names, localizing 236
- Document Summary, Text Enrichment 133
- duration display format metadata 230

**E**

- edge
  - assigning metadata 27
  - connecting components 26
  - for export and import graphs 88
- Enabled configuration property for components 207
- Endeca data domain, sending data to 30
- Export Config component 170
- exporting
  - data domain configuration edge 88
  - data domain configurations 86
  - view definitions 91
- exporting data domain configuration 87

**F**

- French, text enrichment for 148
- full index load source format 50

**G**

- GCR
  - See Global Configuration Record
- geocode display format settings 231
- German, text enrichment for 148
- Global Configuration Record
  - about 64
  - example 65
  - graph 65
  - Web services code to load 65
- graph
  - adding component 21
  - building 21
  - debugging 29
  - running 28
  - secure parameters 14
  - underlying XML 29

**I**

- IAS
  - See Integrator Acquisition System
- Import Config component 171
- importing
  - data domain configuration 87
  - data domain configurations edge 88
  - view definitions 93
- Information Discovery components
  - Add KVPs 152
  - Bulk Add/Replace Records 156
  - common configuration properties 206
  - Delete Records 163
  - enabling SSL 210
  - Export Config 170
  - Import Config 171
  - Language Detector 173
  - Merge Managed Values 175

- Merge Records 178
- Modify Records 183
- Record Store Reader 190
- Reset Data Domain 192
- Text Enrichment 194
- Text Tagger Regex 196
- Text Tagger Whitelist 197
- TransactionRunGraph 201
- visual properties 205

- initial load, primary key for 52
- input data format
  - Merge Records component 54
  - Modify Records component 57
- Integrator Acquisition System 102
- Integrator ETL
  - additional documentation 17
  - configuring 13
  - overview 11
  - Server 12
  - starting 19

internationalization 37

**J**

- Java heap space errors 213

**L**

- language detection 149
- Language Detector
  - component reference details 173
  - edges 150
- languages 37
- Lexalytics
  - See Text Enrichment
- list data type 49
- load graph, building 49
- loading configuration
  - creating aggregation edge 85
  - generating single XML string 85
- loading data 45
  - choosing a strategy 46
- localizing display names and descriptions 236, 237, 238

**M**

- maintaining data 45
- Managed attribute values
  - default 39
  - managing 72
- master password, setting 14
- mdexType Custom properties 39
- Merge Managed Values component 175
- Merge Records component
  - batch size adjustments 208



- choosing 47
- configuring 55
- graph 54
- input format 54
- reference details 178
- metadata
  - assigning to an edge 27
  - defining 22
  - prepending collection keys to attribute names in 244
  - supported data types 36
- modifying records 56
  - input data format 57
- Modify Records
  - component reference details 183
  - defining the Record Set Specifier Attributes 58
  - graph 58
  - input data format 57
- multi-assign data
  - about 51
  - errors from misconfiguration 216
- Multi-assign delimiter 207
- multi-value data 49
- MVals
  - See Managed attribute values

## N

- Named Entities extraction, Text Enrichment 133
- normalizing themes 148
- number format display settings 231

## O

- OBI
  - adding collection to project 119
  - Integrator ETL Server connection 124
  - SSL 123
  - SSL graph configuration 124
- Oracle Business Intelligence Server 108
- outer transaction
  - about 96
  - committing manually 100
  - creating graph 98, 99
  - error scenarios 215
  - performance impact 101
  - rolling back 100
  - steps input file 98
  - when to use in Integrator ETL graphs 97
- Outer transactions
  - setting up 97
  - Transaction RunGraph component 97
- OutOfMemory errors, troubleshooting 213
- output, checking 28

## P

- PDR

- See Property Description Record
- Phase configuration property for components 207
- populating a data domain
  - See loading data
- Portuguese, text enrichment for 148
- precedence rules 80
  - configuration input file 83
  - graph 83
  - schema 81
- primary key 51
- project
  - creating 19
  - loading sample data 20
- Property Description Record 38
  - configuring 66
  - loading 67

## Q

- query topics definition file 144
- Quotation extraction, Text Enrichment 133

## R

- record delimiters, specifying multiple 43
- records adding to data domain 53
- record schema load
  - See attribute schema load
- record search 76
- Record Set Specifier Attributes 58
- record spec property
  - See primary key
- Record Store Reader component 102
  - batch size adjustments 208
  - configuring 103
  - reference details 190
- Record Store Wizard 103
- RECSEARCH\_STRATEGY 76
- relevance ranking 78
- relink\_strategies 78
- Reset Data Domain component 192
- restoring data domains 59
- Rollback Transaction graph 100

## S

- sample data 20
- secure graph parameters 14
- Sentiment Analysis 133
- Server, overview of Integrator ETL 12
- Social Media, adding to Text Enrichment 149
- source data format
  - attribute schema 67
  - full initial load 50

- managed attribute schema 69
- precedence rules 83
- transaction graph 98
- Spanish, text enrichment for 148
- SSL 209
  - enabling for components 210
  - Integrator ETL Server 211
  - OBI graph configuration for 124
  - Web Service Client 211
- standard attribute default values 38
- stop words 78
- Studio
  - localization 235
  - localizing 238
- Studio metadata 217
  - aggregation method metadata 227
  - assigning values to physical attribute groups 222
  - assigning values to physical properties 221
  - attribute group metadata 226
  - attribute metadata 225
  - Boolean display format metadata 228
  - creating logical properties 222
  - creating physical properties 220
  - date/time display format metadata 229
  - display format metadata 228
  - duration display format metadata 230
  - geocode display format settings 231
  - localizing 236, 237, 238
  - managing logical properties 222
  - number format display settings 231
  - time display format metadata 233
  - usage and display 233
- supported languages 37

## T

- tags, appending and overwriting 127
- text enrichment 132
  - adding to a graph 144
  - all caps text 148
  - choosing 126
  - component reference details 194
  - configuration properties 135
  - Document Summary 133
  - edge 145
  - edge metadata 147
  - foreign language processing 148
  - multiple languages 149
  - Named Entities 133
  - normalizing themes 148
  - prerequisites 134
  - Quotations 133
  - Sentiment Analysis 133
  - Social Media 149
  - Themes 133
  - Twitter 149
- Text Tagger Regex
  - about 130

- adding to graph 131
- component details 196
- edges 132
- input patterns 131

### Text Taggers

- appending target values 127
- available components 126
- metadata 127
- overwriting target values 127

### Text Tagger Whitelist

- about 127
- adding to graph 128
- component details 197
- edges 129
- input tags-rules 128
- search term maximum character length 129

### Theme extraction, Text Enrichment 133

- themes, normalizing 148

- thesaurus 79

- time display format metadata 233

- time zone for data, setting 13

- transaction

- See outer transaction

- Transaction RunGraph component 97, 201

- Trash component, adding 25

- troubleshooting

- connection errors 215
- multi-assign delimiter error 216
- outer transactions 215
- out of memory 213

- Twitter, adding to Text Enrichment 149

## V

- valid characters for ingest 208

- value search 77

- versions of web services, verifying 14

- view definitions

- about 91
- exporting 91
- importing 93

- Visual configuration properties for components 205

## W

- web services, verifying versions of 14

- wizards

- Load Data from OBI Server 108, 123
- Record Store 104

## X

- XML-based configuration graphs 84

- XML characters 208

- XML source of graph 29