Oracle® Endeca Information Discovery

Integrator User's Guide

Version 2.4.0 • November 2012

Copyright and disclaimer

Table of Contents

Preface

Oracle® Endeca Information Discovery Studio is an enterprise data discovery platform for advanced, yet intuitive, exploration and analysis of complex and varied data.

Information is loaded from disparate source systems and stored in a faceted data model that dynamically supports changing data. This integrated and enriched data is made available for search, discovery, and analysis via interactive and configurable applications. Oracle Information Discovery Studio includes a Provisioning Service that allows you to upload data directly from spreadsheet files.

Oracle Endeca Information Discovery Studio enables an iterative “model-as-you-go” approach that simultaneously frees IT from the burdens of traditional data modeling and supports the broad exploration and analysis needs of business users.

About this guide

This guide describes how to use the Oracle Endeca Information Discovery Integrator to ingest data into an Endeca data store.

The Integrator is used to load records, taxonomies, and configuration documents into the Endeca data store.

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 following sections:

- Creating an Integrator Project
- Creating Basic Graphs

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

- Working with data stores
- Outer Transactions
- Using Endeca Components

Data architects may also want to consult the wiki page on *Integrator Design Patterns: https://wikis.oracle.com/display/endecainformationdiscovery/Integrator+Design+Patterns*.

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

- Managing data stores
- Managing View Definitions

Who should use this guide

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

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 documentation, it is recommended that Integrator users read the Oracle Endeca Server documentation.

In particular, it is recommended that users read and become familiar with the *Oracle Endeca Server Developer's Guide*.

Users may find the information in the *Oracle Endeca Server Data Loading Guide* valuable as well.

Chapter 1

Integrator Overview

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

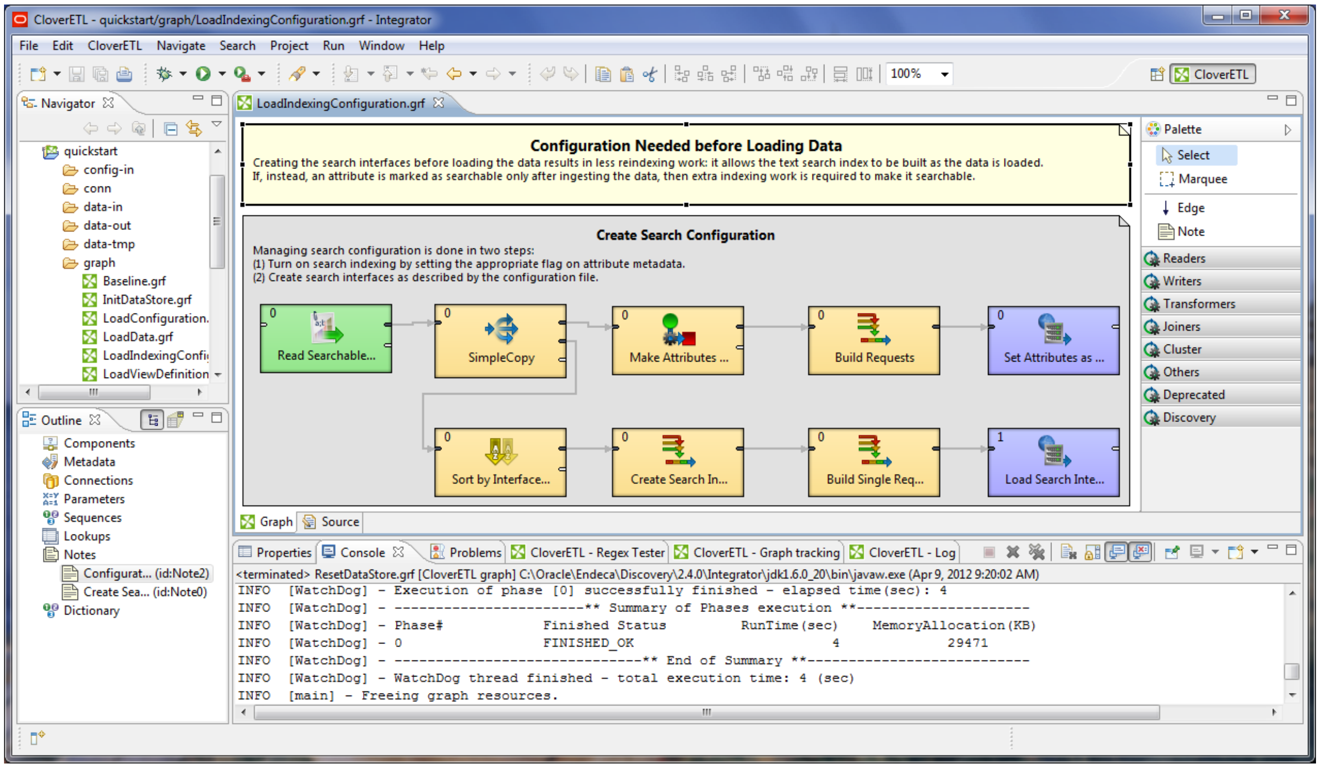
Integrator Designer

The Integrator 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 store. More complex graphs will use additional components, such as Transformer and Joiner components.

The Integrator, 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 on the component icon.

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



Configuration Needed before Loading Data

Creating the search interfaces before loading the data results in less reindexing work: it allows the text search index to be built as the data is loaded.
If, instead, an attribute is marked as searchable only after ingesting the data, then extra indexing work is required to make it searchable.

Create Search Configuration

Managing search configuration is done in two steps:
(1) Turn on search indexing by setting the appropriate flag on attribute metadata.
(2) Create search interfaces as described by the configuration file.

```
<terminated> ResetDataStore.grf [CloverETL graph] C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20\bin\javaw.exe (Apr 9, 2012 9:20:02 AM)
INFO  [WatchDog] - Execution of phase [0] successfully finished - elapsed time(sec): 4
INFO  [WatchDog] - ----------------------** Summary of Phases execution **--------------------
INFO  [WatchDog] - Phase#            Finished Status         RunTime(sec)    MemoryAllocation(KB)
INFO  [WatchDog] - 0                 FINISHED_OK                        4             29471
INFO  [WatchDog] - ----------------------------** End of Summary **---------------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 4 (sec)
INFO  [main] - Freeing graph resources.
```

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 lets you create a graph and configure its components.
- The **Palette** lets you select a component and drag it to the Graph Editor.

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

Integrator Server

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

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

You use the 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.

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

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

Configuring Integrator

This section provides information about configuration options for Integrator.

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 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 three 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 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, to treat the time stamp 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:

```
$0.OrderDate = str2date($0.OrderDate + " UTC", "dd.MM.yyyy HH:mm:ss Z");
```

Method 3: Configure the JVM

The third method is to change the default time zone in the JVM running the Integrator graph to UTC. This can be accomplished by specifying the following argument (the quotes are important):

```
"-Duser.timezone=UTC"
```

Add this argument in: **Run > Run Configurations > launch-config > Arguments tab > VM arguments** (where **launch-config** is the name of the configuration you want to change).

Verifying installed Web service versions

Before making any calls to the Oracle Endeca Server with the **WebServiceClient** component of the Integrator, 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 1.0 is represented in the namespace for the Configuration Web Service:

```
xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0
```

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

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 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 the string similar to the following in the **Operation name** field for the component:

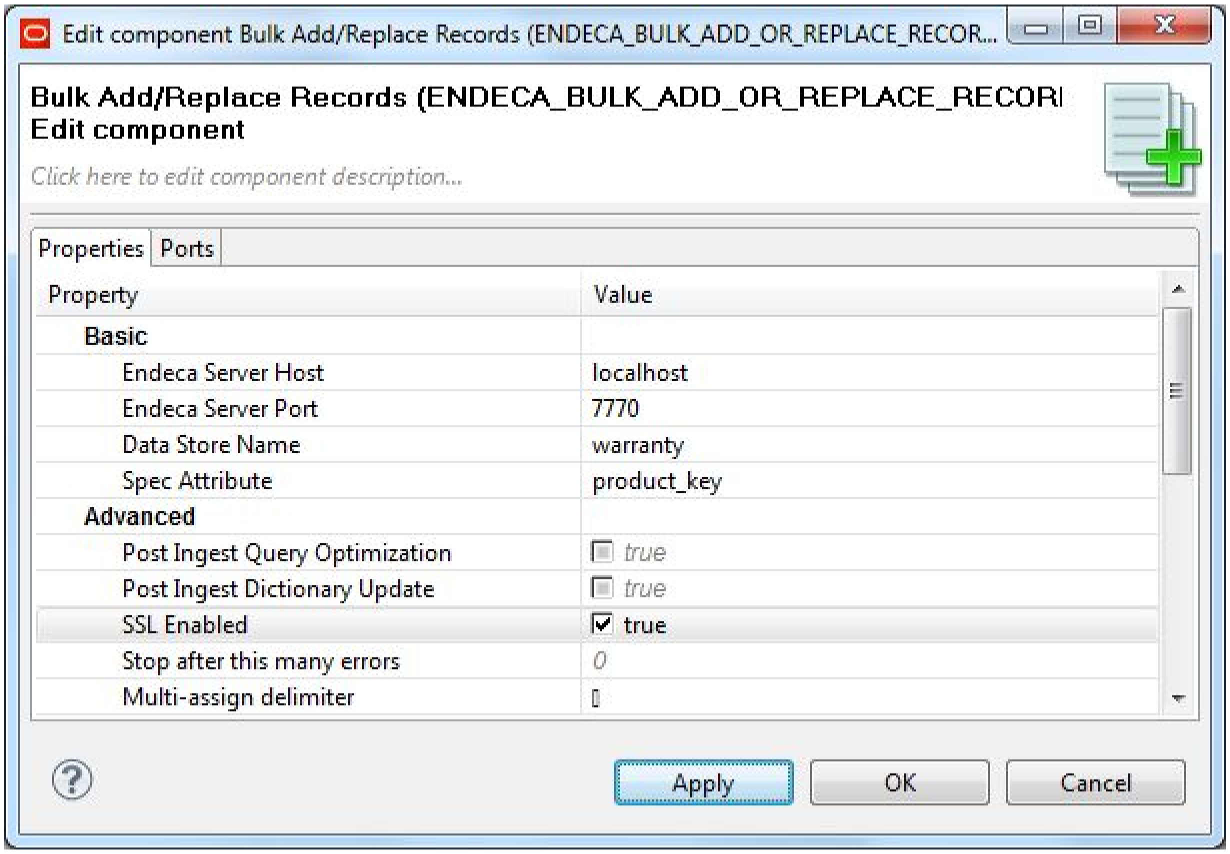
```
{http://www.endeca.com/MDEX/config/services/config/1}Config#ConfigPort#DoConfigTransaction
```

Notice that this string includes a target namespace for the Web service that accurately reflects the major version of the Web service as `/1` in this example.

Configuring SSL

All Information Discovery components support SSL connections to an SSL-enabled Dgraph.

This procedure assumes that you have used the Integrator **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:


Edit component Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECOR...
Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORI
Edit component
Click here to edit component description...
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
Endeca Server Host	localhost
Endeca Server Port	7770
Data Store Name	warranty
Spec Attribute	product_key
Advanced	
Post Ingest Query Optimization	☐ true
Post Ingest Dictionary Update	☐ true
SSL Enabled	☑ true
Stop after this many errors	0
Multi-assign delimiter	▯

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.

About the Integrator Quick Start project

The Integrator comes with a sample project and data, which you can launch from the Integrator Welcome screen.

The Integrator Quick Start project demonstrates the Oracle Endeca Information Discovery product in action, using sales and product data from a fictitious bicycle manufacturer.

Note: For details of working with the Integrator Quick Start project, see the *Oracle Endeca Information Discovery Quick Start Guide*.

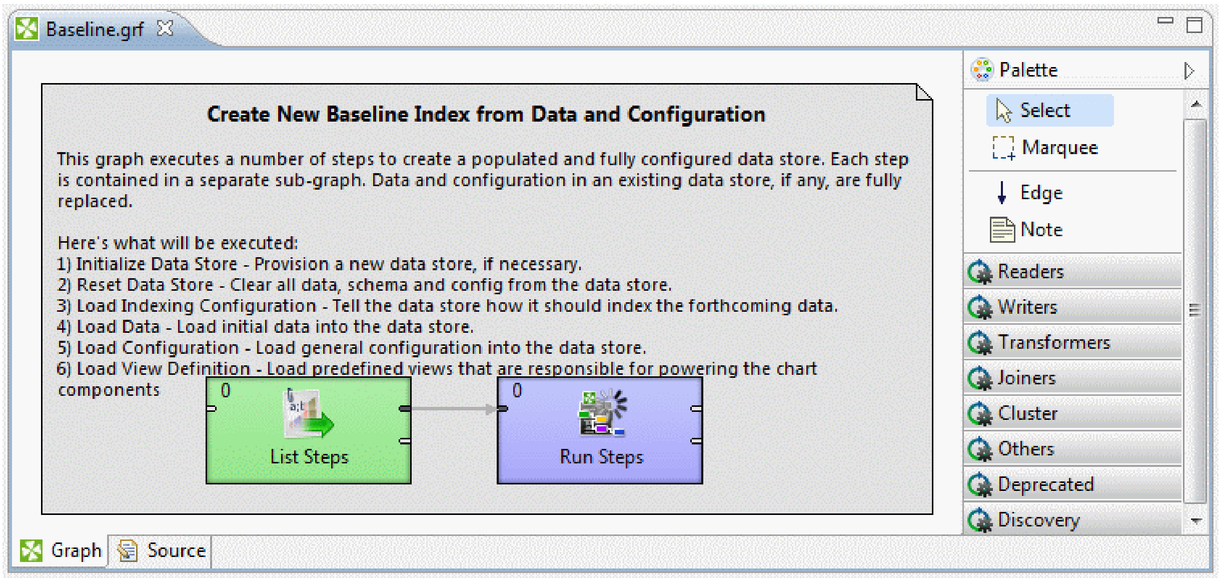
Launching the Integrator Quick Start project

The Quick Start project is installed with the Integrator.

To launch the Integrator Quick Start project:

1. Do one of the following:
   - In the Integrator Welcome screen, click "Open and view the Quick Start project."
   - In the Integrator, select **File>New>Quick Start Example**.

   Integrator opens with a fully-formed project, ready for you to explore or run.



Create New Baseline Index from Data and Configuration

This graph executes a number of steps to create a populated and fully configured data store. Each step is contained in a separate sub-graph. Data and configuration in an existing data store, if any, are fully replaced.

Here's what will be executed:
1) Initialize Data Store - Provision a new data store, if necessary.
2) Reset Data Store - Clear all data, schema and config from the data store.
3) Load Indexing Configuration - Tell the data store how it should index the forthcoming data.
4) Load Data - Load initial data into the data store.
5) Load Configuration - Load general configuration into the data store.
6) Load View Definition - Load predefined views that are responsible for powering the chart components

Additional documentation

Additional Integrator documentation is available online and as part of the Oracle Endeca Information Discovery documentation set.

Information Discovery documentation set

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

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

Documentation online

You can access online documentation from within Integrator 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 Designer Guide*.

Chapter 2

Building a Simple Project

This section describes how to build a simple project from scratch.

Starting the Integrator

This topic describes how to start the Integrator.

To start the Integrator:

1. From the **Start** menu, choose **All Programs>Oracle>EID Integrator 2.4.0>Oracle EID Integrator**.
2. Depending on how your Integrator is configured, you may be asked to select or confirm your workspace.

   The workspace is the directory where Integrator creates and stores your projects.
3. The first time you launch Integrator, a **Welcome** screen appears. Use this screen to navigate to launch Integrator or to load the Quick Start sample application.

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

Creating a project

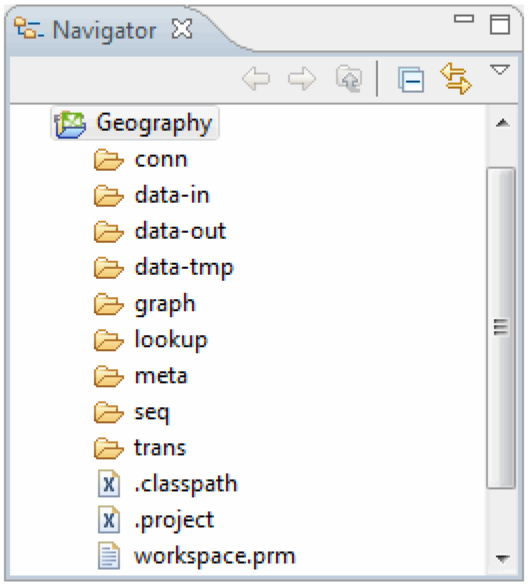
This topic describes how to create a new Integrator project.

To create a new Integrator project:

1. In the Integrator, select **File>New >Clover ETL Project**.
   If you are running Integrator 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 **Create a new Clover ETL 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 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.



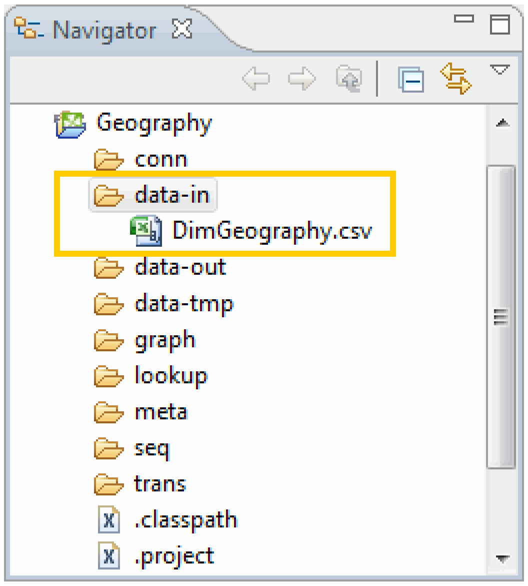
Adding the sample data

This topic illustrates how to add sample data to the simple project.

To get going quickly, you will copy a sample data file from the Quick Start project.

To load the sample data:

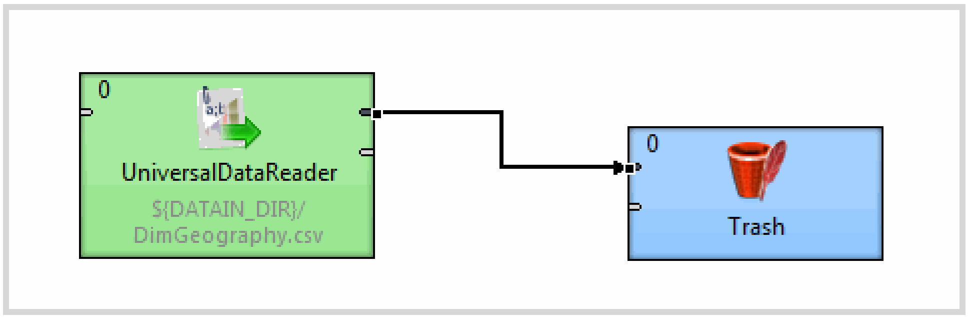
1. In Integrator, load the Quick Start application with **File>New>QuickStart Example**.
2. In the Quick Start project, open the `data-in` folder and copy the `DimGeography.csv` file.
3. Paste the `DimGeography.csv` file in the `data-in` directory of your new **Geography** project. Your `data-in` directory should look like this:



Building your first graph

We are now ready to start building a transformation graph.

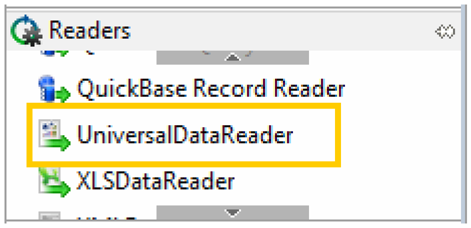
This simple graph contains two components connected by a single edge.



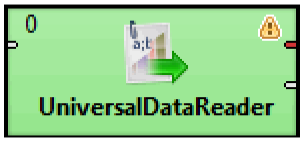
Adding a new component

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 **Next** and then 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.



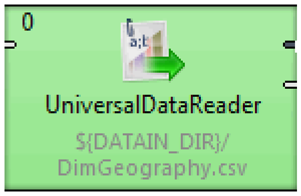
The **LoadGeography.grf** now contains a single **UniversalDataReader** component.



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.
   

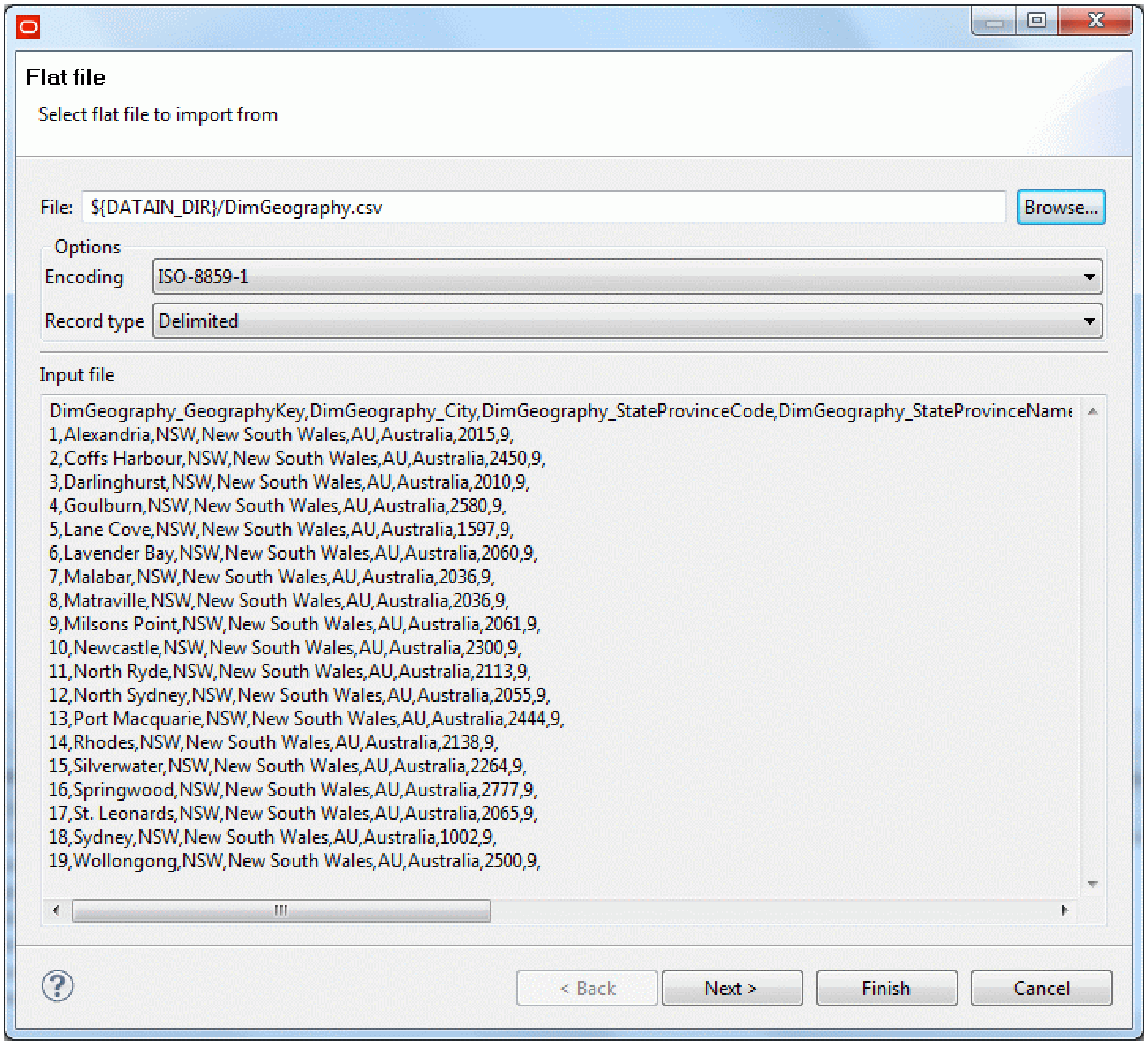
9. Save the **LoadGeography** graph.

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**.



Flat file

Select flat file to import from

File: ${DATAIN_DIR}/DimGeography.csv

Options

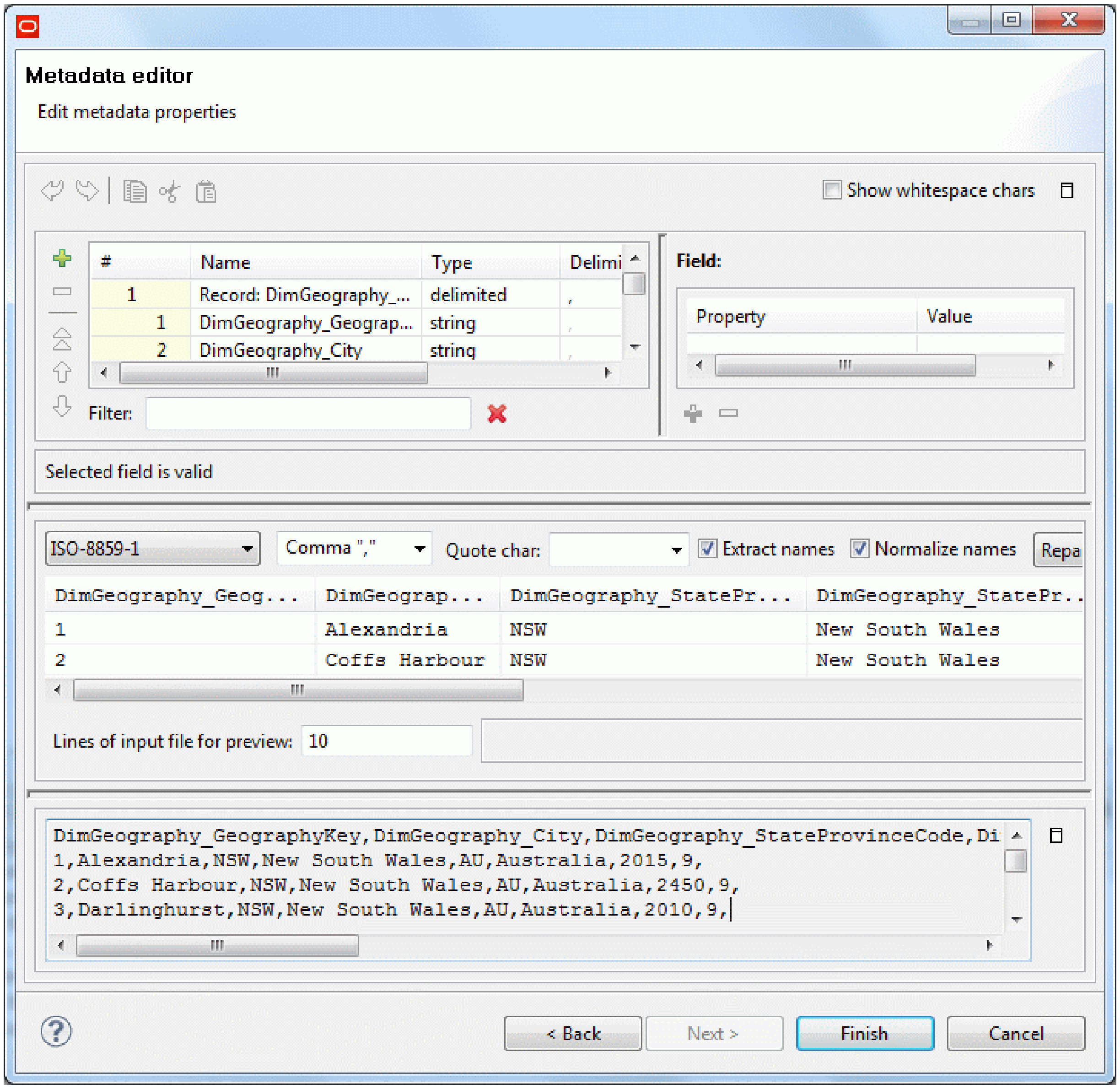
Encoding ISO-8859-1

Record type Delimited

Input file

```
DimGeography_GeographyKey,DimGeography_City,DimGeography_StateProvinceCode,DimGeography_StateProvinceName
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,
4,Goulburn,NSW,New South Wales,AU,Australia,2580,9,
5,Lane Cove,NSW,New South Wales,AU,Australia,1597,9,
6,Lavender Bay,NSW,New South Wales,AU,Australia,2060,9,
7,Malabar,NSW,New South Wales,AU,Australia,2036,9,
8,Matraville,NSW,New South Wales,AU,Australia,2036,9,
9,Milsons Point,NSW,New South Wales,AU,Australia,2061,9,
10,Newcastle,NSW,New South Wales,AU,Australia,2300,9,
11,North Ryde,NSW,New South Wales,AU,Australia,2113,9,
12,North Sydney,NSW,New South Wales,AU,Australia,2055,9,
13,Port Macquarie,NSW,New South Wales,AU,Australia,2444,9,
14,Rhodes,NSW,New South Wales,AU,Australia,2138,9,
15,Silverwater,NSW,New South Wales,AU,Australia,2264,9,
16,Springwood,NSW,New South Wales,AU,Australia,2777,9,
17,St. Leonards,NSW,New South Wales,AU,Australia,2065,9,
18,Sydney,NSW,New South Wales,AU,Australia,1002,9,
19,Wollongong,NSW,New South Wales,AU,Australia,2500,9,
```

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



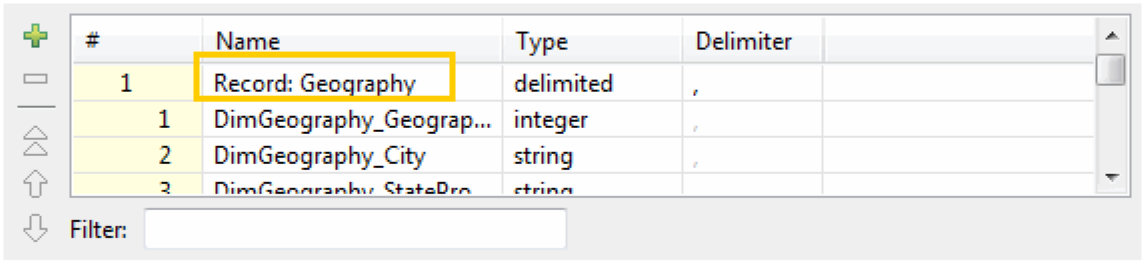
#	Name	Type	Delimi
1	Record: DimGeography_...	delimited	,
1	DimGeography_Geograp...	string	,
2	DimGeography_City	string	,

ISO-8859-1 · Comma "," · Quote char: · [✓] Extract names · [✓] Normalize names

DimGeography_Geog...	DimGeograp...	DimGeography_StatePr...	DimGeography_StatePr..
1	Alexandria	NSW	New South Wales
2	Coffs Harbour	NSW	New South Wales

```
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,
```

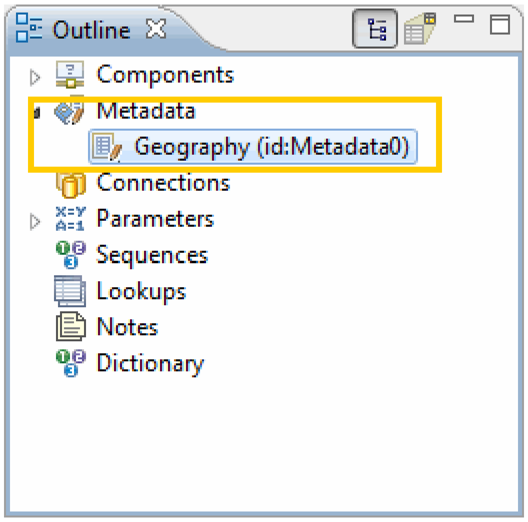
4. To give the metadata a useful name, rename the topmost record in the **Fields** list to **Geography**.



#	Name	Type	Delimiter
1	Record: Geography	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.

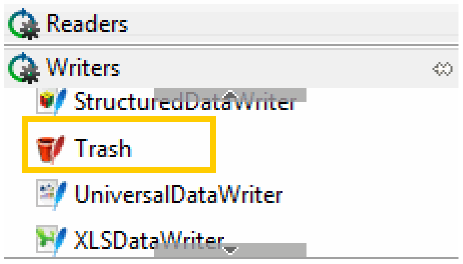
Adding a Trash component

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

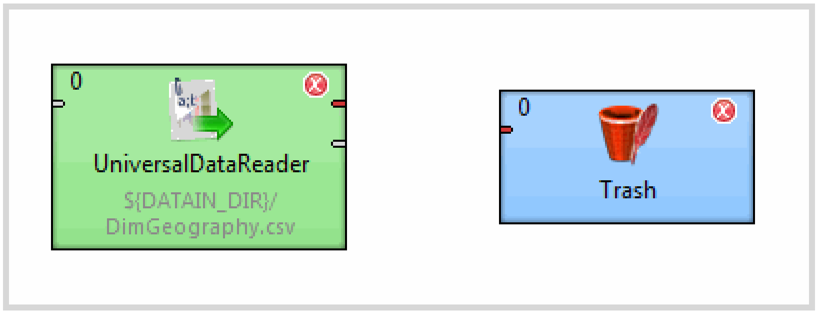
The **Trash** component tests the end points in a graph. Any data that arrives in the **Trash** component is discarded, which means that there is no need to create a file or database output. The **Trash** component also allows you to use some of the debugging capabilities of the Integrator to monitor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

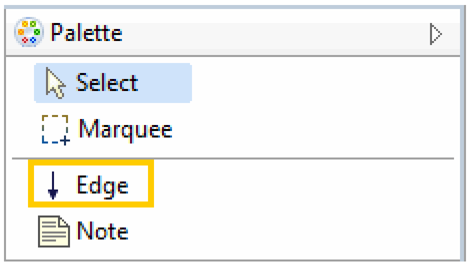


Connecting two components with an edge

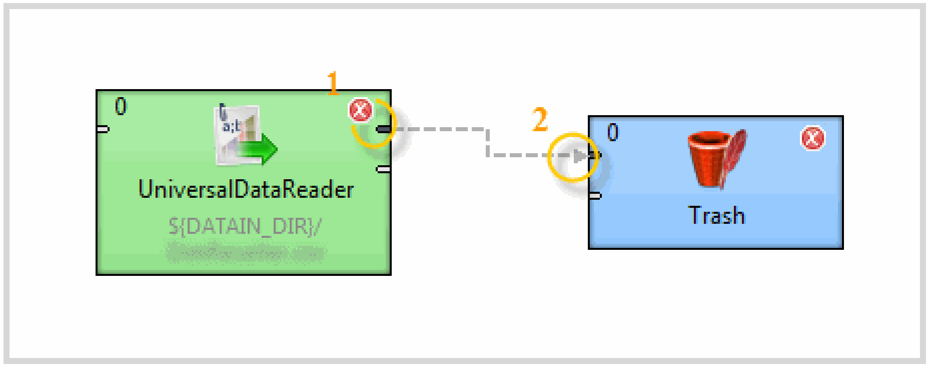
In this topic, you connect the **Trash** component to the **UniversalDataReader** component with an edge.

To connect two components with an edge:

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

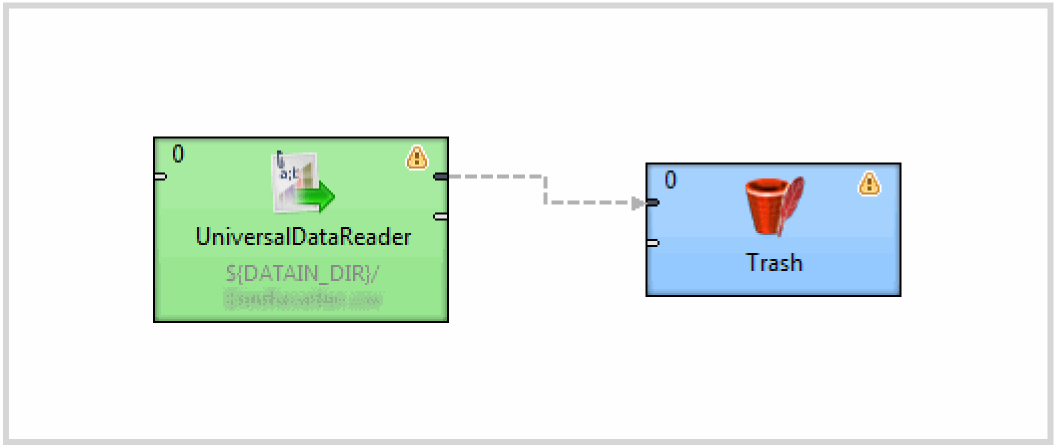


2. Click on the upper output port of the **UniversalDataReader** (number 1 in the image below) and drag across to the upper input port of the **Trash** component (number 2 in the image below).
You have to click on the target component to connect the edge.



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

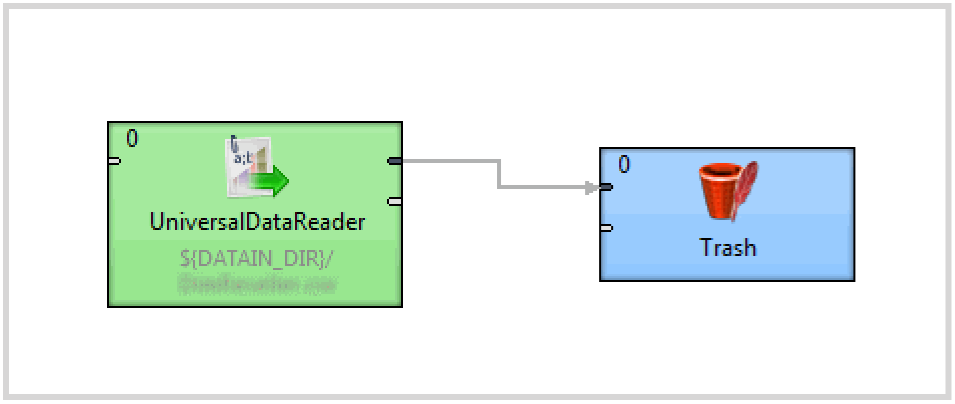
This topic illustrates how to apply metadata to an edge.

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

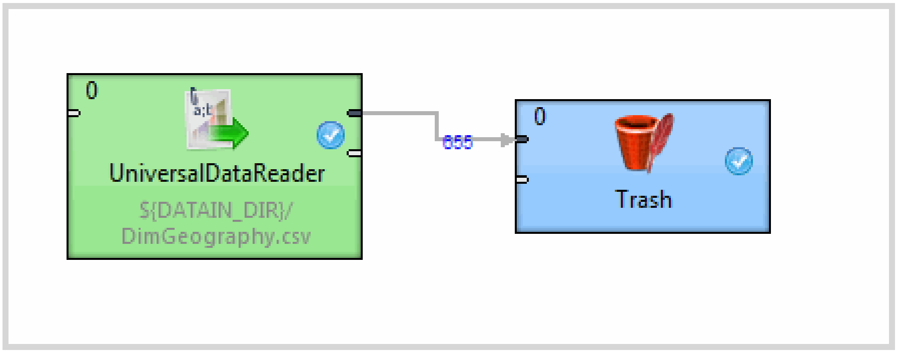
After creating the graph and configuring the components, you can run the graph.

To run your Integrator 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 select **Run As>CloverETL graph**.
- In the toolbar, click the run icon .

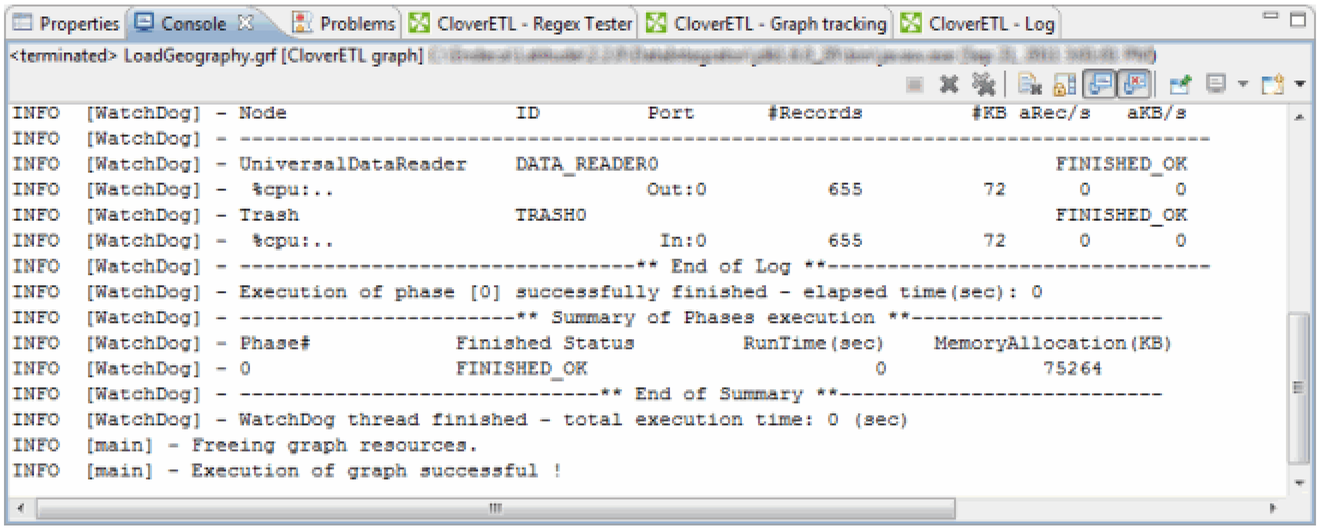
Upon successful execution, the components are flagged with a check mark, and the edge displays the number of records processed.



Checking the output

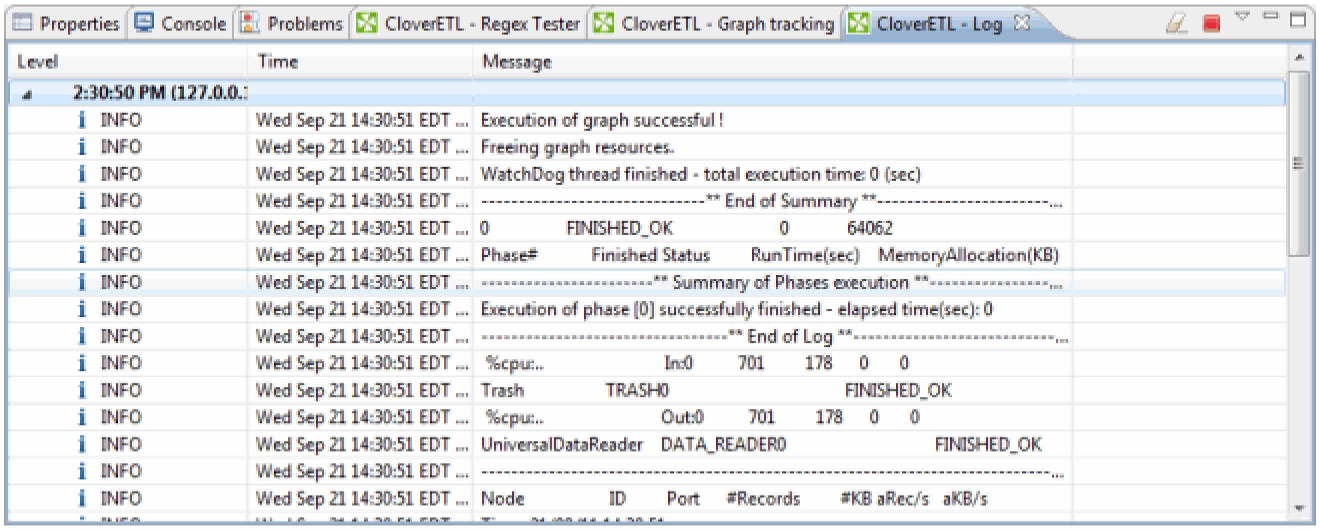
Using the **Console** and the **Clover Log**, you can check the details of graph execution.

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]
INFO  [WatchDog] - Node                    ID          Port       #Records        #KB aRec/s    aKB/s
INFO  [WatchDog] - ---------------------------------------------------------------------------------
INFO  [WatchDog] - UniversalDataReader     DATA_READER0                                 FINISHED_OK
INFO  [WatchDog] -  %cpu:..                            Out:0          655         72       0        0
INFO  [WatchDog] - Trash                   TRASH0                                       FINISHED_OK
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.:		
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_READER0 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 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 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 select **Enable Debug**.

   The debug icon appears on the edge.

2. Re-run the graph so that Integrator can generate the debug data.

3. Right-click the edge and select **View data**.
   The **View data** window shows all of the data fields correctly parsed and loaded.

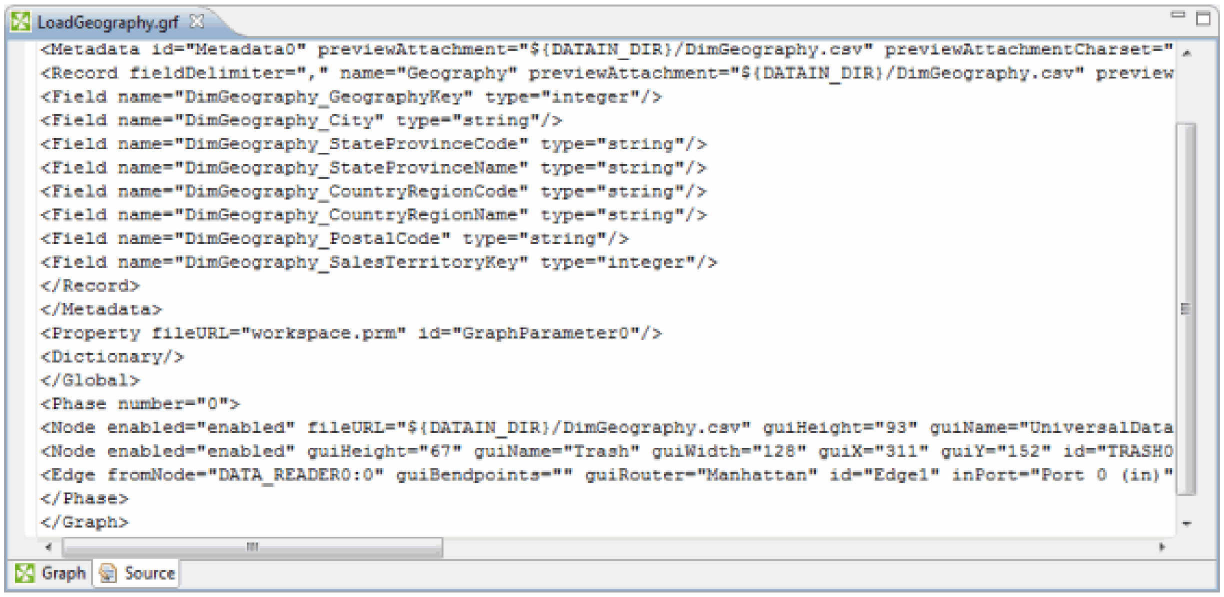
Viewing the XML source for the graph

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 store

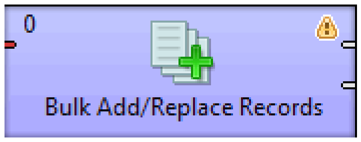
In this topic, you will replace the **Trash** component with a component that sends records to the Endeca data store.

The procedure below assumes that the Endeca Server is running (using the default 7770 port) and that you have created a running Endeca data store named **geography**. Creating Endeca data stores is documented in the *Oracle Endeca Server Administrator's Guide*.

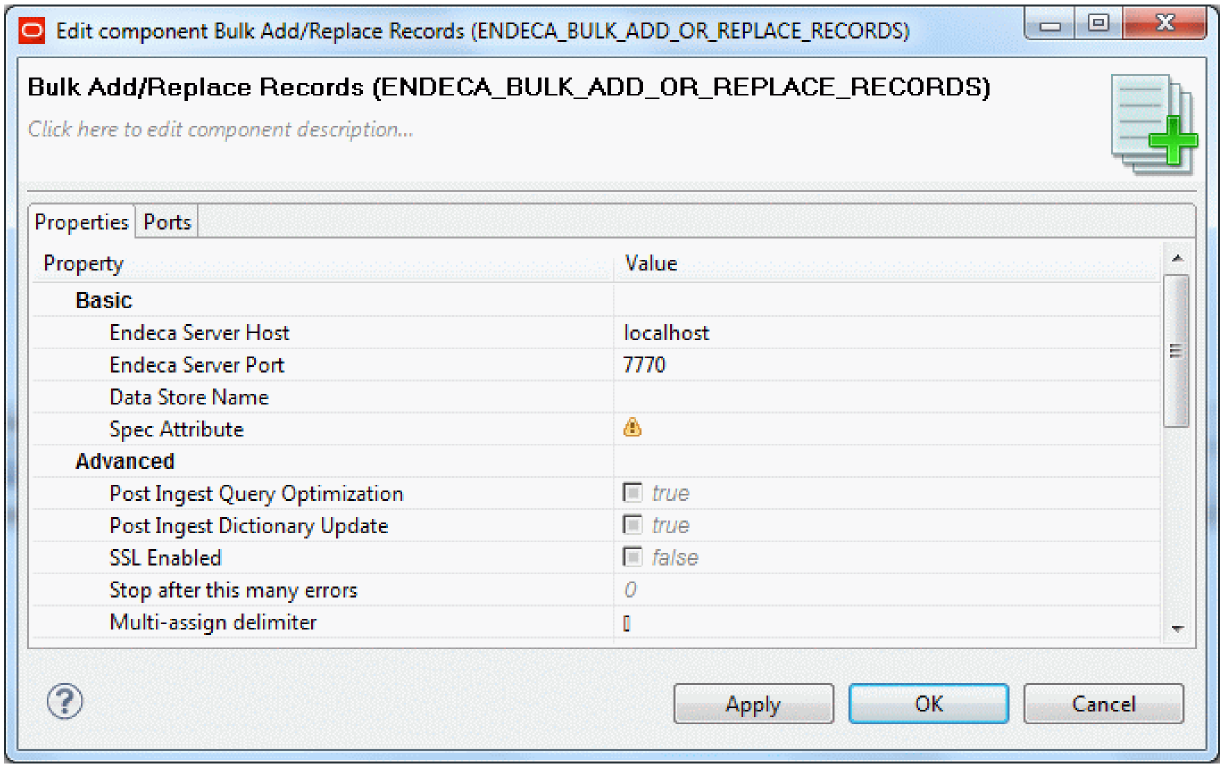
For details about the Endeca data store load process, see *Recommended data store creation and configuration process on page 23*.

To send data to an Endeca data store:

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.


Edit component Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)

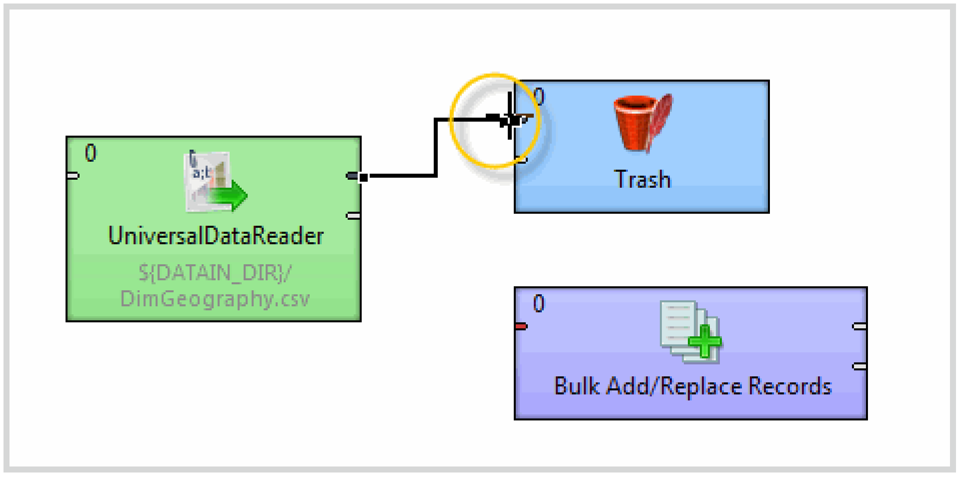
Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)

Click here to edit component description...

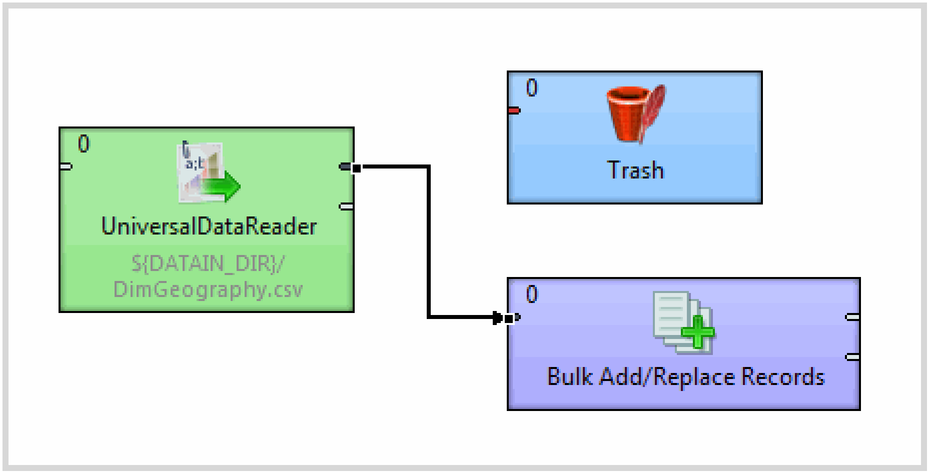
Property	Value
Basic	
Endeca Server Host	localhost
Endeca Server Port	7770
Data Store Name	
Spec Attribute	⚠
Advanced	
Post Ingest Query Optimization	☐ *true*
Post Ingest Dictionary Update	☐ *true*
SSL Enabled	☐ *false*
Stop after this many errors	*0*
Multi-assign delimiter	▯

5. Set the following mandatory properties in the **Basic** section, and then click **OK**:
   (a) **Endeca Server Host**: The name or IP address of the machine. The default value (`localhost`) can be used as the name.
   (b) **Endeca Server Port**: 7770 (the value of the default Endeca Server port)
   (c) **Data Store Name**: The name of the Endeca data store. In this example, `geography` is the name.
   (d) **Spec Attribute**: The name of the primary key. In this example, `DimGeography_GeographyKey` is the name.

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



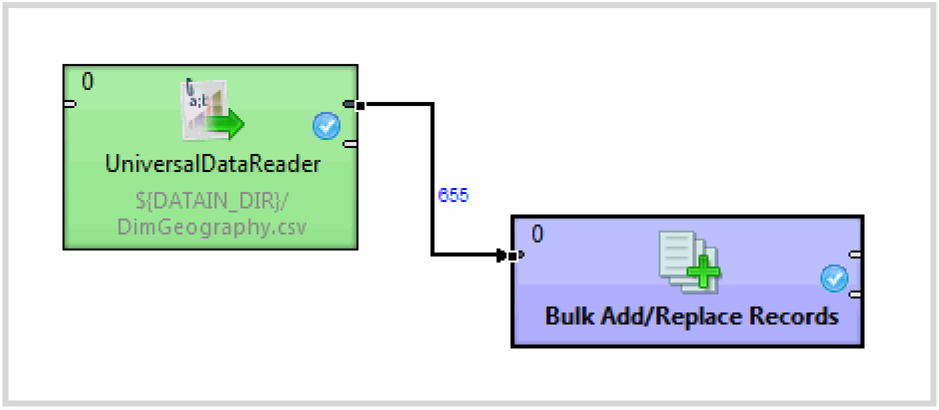
7. Drag the edge endpoint from the **Trash** component to the **Bulk Add/Replace Records** component that you just configured.



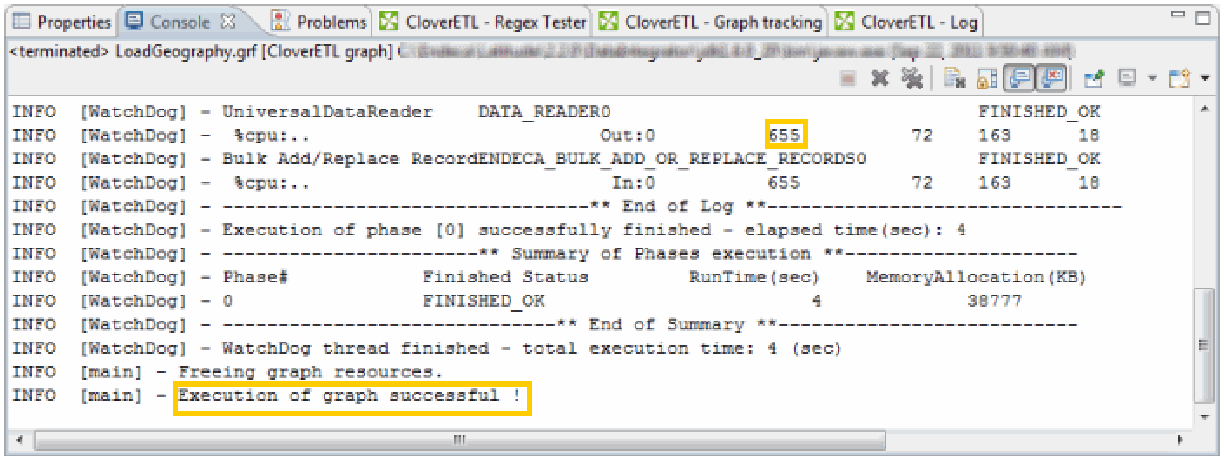
When you move the edge, its associated metadata is also moved.

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**:



<terminated> LoadGeography.grf [CloverETL graph] [redacted]

```
INFO  [WatchDog] - UniversalDataReader    DATA_READER0                                   FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           Out:0          655          72     163      18
INFO  [WatchDog] - Bulk Add/Replace RecordENDECA_BULK_ADD_OR_REPLACE_RECORDS0            FINISHED_OK
INFO  [WatchDog] -  %cpu:..                            In:0          655          72     163      18
INFO  [WatchDog] - ---------------------------------** End of Log **--------------------------------
INFO  [WatchDog] - Execution of phase [0] successfully finished - elapsed time(sec): 4
INFO  [WatchDog] - -----------------------** Summary of Phases execution **---------------------
INFO  [WatchDog] - Phase#             Finished Status         RunTime(sec)     MemoryAllocation(KB)
INFO  [WatchDog] - 0                  FINISHED_OK                         4              38777
INFO  [WatchDog] - ------------------------------** End of Summary **---------------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 4 (sec)
INFO  [main] - Freeing graph resources.
INFO  [main] - Execution of graph successful !
```

Chapter 3

Working with Data Stores

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

About working with data stores

This topic describes general information about working with data stores

The configuration and loading of a data store may be an iterative process. A data store 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 store, including the data store configuration, modify and reload the data store 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 store creation and configuration process

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

Use the following process for creating and, configuring, and populating a data store. 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 store.
2. Update the Global Configuration Record (GCR).

   The Global Configuration Record sets the global configuration settings for the Endeca data store. See *Global Configuration Record on page 48*.

3. Update the attribute schema configuration. See *Configuring Attributes on page 49*.

   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 than 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 (DDRs), which define the relation and hierarchy between the values of the attribute.

   3. Managed attribute values, or mvals.

4. Configuration documents, loaded in the order listed below:

   1. relrank_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 57*.

   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 58*.

   3. dimsearch_config

      This document sets the configuration for value search. For more information, see *Configuring value search on page 59*.

   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 60*.

   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 60*.

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

The following configuration documents can be loaded at any point in the sequence:

- Stemming files

The default stemming file installed with the Endeca Server is the English stemming file. If you want to support searches in another supported language in the data store, you must upload the stemming file for that language. For more information, see *Loading stemming files on page 61*.

- Precedence rules

  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 63*.

About data store data

This section describes the data loaded into the data store

Supported data types

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

The table also shows how the Integrator 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 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`
`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 27*.

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
`mdex-property_Key`	Set to the standard attribute name specified in the request.
`mdex-property_Type`	Set to the standard attribute type specified in the request. If no type was specified, defaults to the `mdex:string` type.
`mdex-property_IsPropertyValueSearchable`	`true` (the standard attribute will be enabled for value search)
`mdex-property_IsSingleAssign`	`false` (a record may have multiple value assignments for the standard attribute)
`mdex-property_IsTextSearchable`	`false` (the standard attribute will be disabled for record search)
`mdex-property_IsUnique`	`false` (more than one record may have the same value of this standard attribute)
`mdex-property_TextSearchAllowsWildcards`	`false` (wildcard search is disabled for this standard attribute)
`system-navigation_Select`	`single` (allows selecting only one refinement from this standard attribute)
`system-navigation_ShowRecordCounts`	`true` (record counts will be shown for a refinement)
`system-navigation_Sorting`	`record-count` (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
`mdex-dimension_Key`	Set to the managed attribute name specified in the request.
`mdex-dimension_EnableRefinements`	`true` (refinements will be displayed)
`mdex-dimension_IsDimensionSearchHierarchical`	`false` (hierarchical search is disabled during value searches)
`mdex-dimension_IsRecordSearchHierarchical`	`false` (hierarchical search is disabled during record searches)

Creating mdexType Custom properties

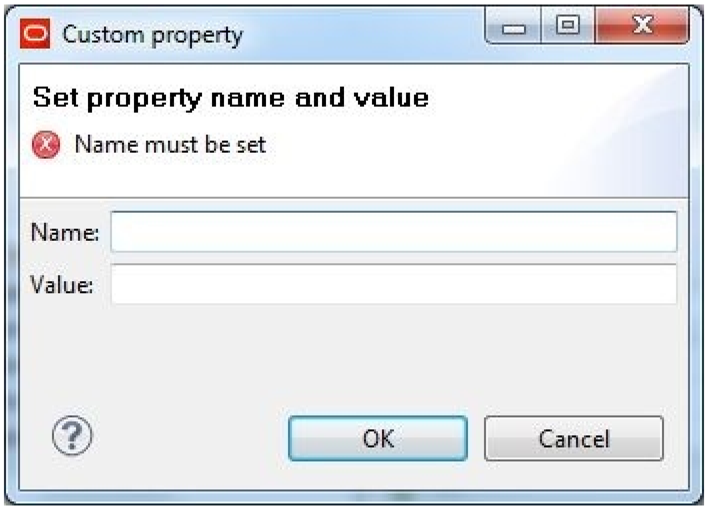
The Integrator allows you to create an **mdexType** Custom property that you can use to explicitly specify the MDEX type to which a particular Endeca standard attribute should map.

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 the Integrator. In this case, the ETL developer has to send a string through the Integrator, 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 native property type when the records are sent to the Dgraph.

This functionality is particularly useful for non-String multi-assign properties, because the Integrator natively has to treat the property as a string since 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 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:



Set property name and value

Name must be set

Name:

Value:

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
```

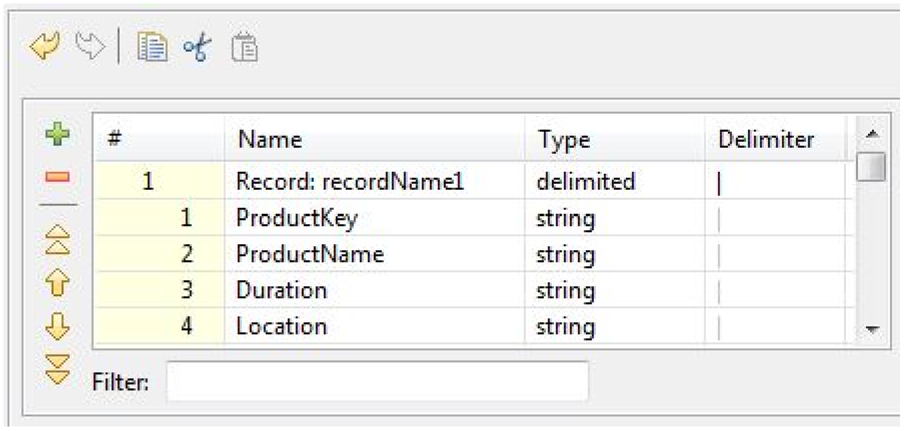
It creates only one record with four standard attributes:

- 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 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.

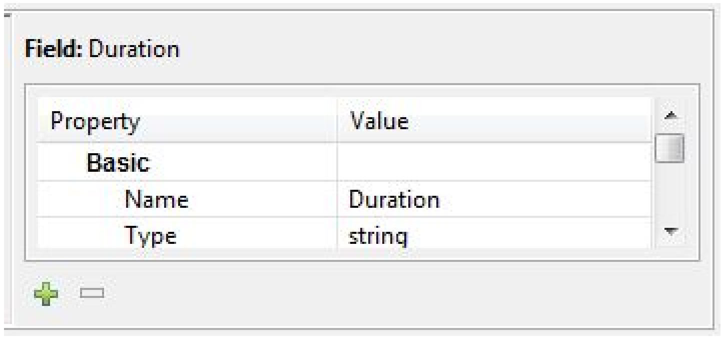
   At this point, the Record pane of the Metadata editor should look like this:



#	Name	Type	Delimiter
1	Record: recordName1	delimited	|
1	ProductKey	string	|
2	ProductName	string	|
3	Duration	string	|
4	Location	string	|

Filter:

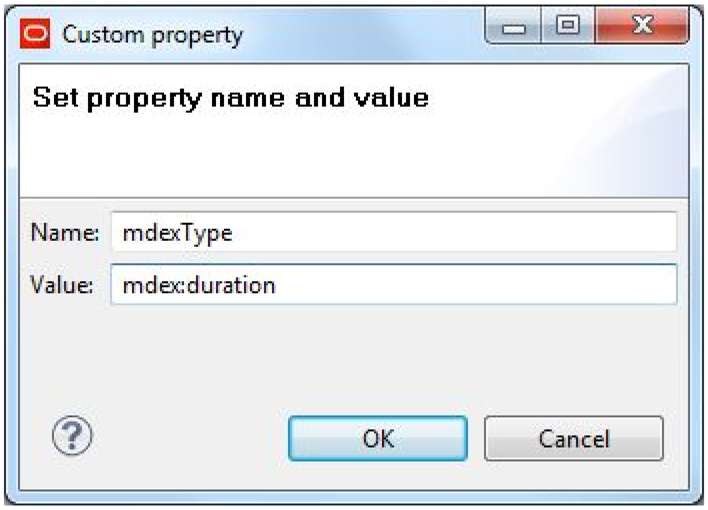
5. In the Record pane of the Metadata editor, make these changes:
   (a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a more descriptive name.
   (b) Change the ProductKey Type to **integer**.
   (c) Leave the ProductName Type as **string**.
6. To create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



Field: Duration

Property	Value
Basic	
Name	Duration
Type	string

   (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:



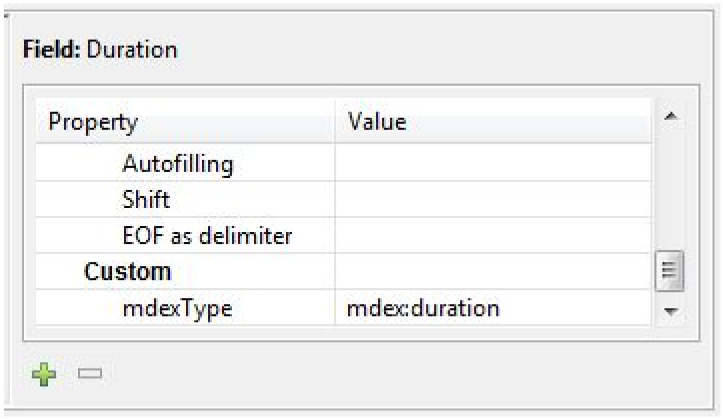
Custom property

Set property name and value

Name: mdexType

Value: mdex:duration

(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:



Field: Duration

Property	Value
Autofilling	
Shift	
EOF as delimiter	
Custom	
mdexType	mdex:duration

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.

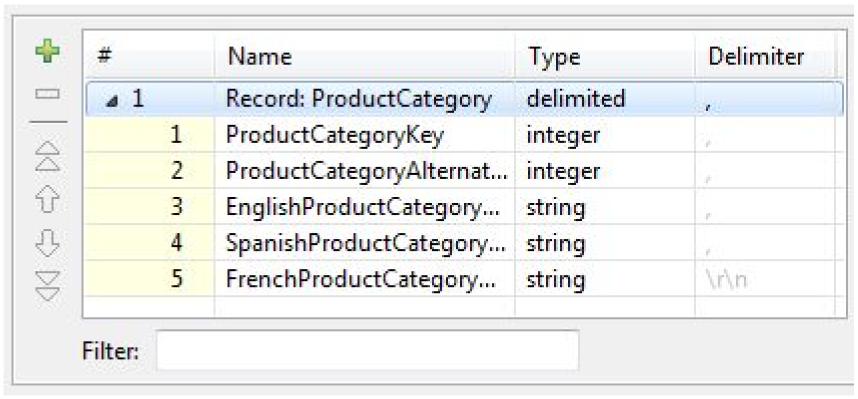
However, you may have files that were created on different platforms (for example, if you have input files that you check out of a version control system, the files' line endings will vary according to the platform). In this case, you would want the record delimiter to be set to both values, so that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

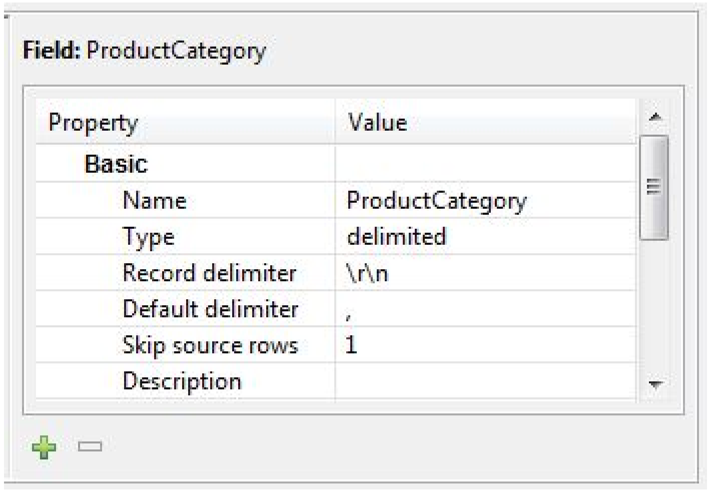
To specify multiple record delimiters in the metadata:

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

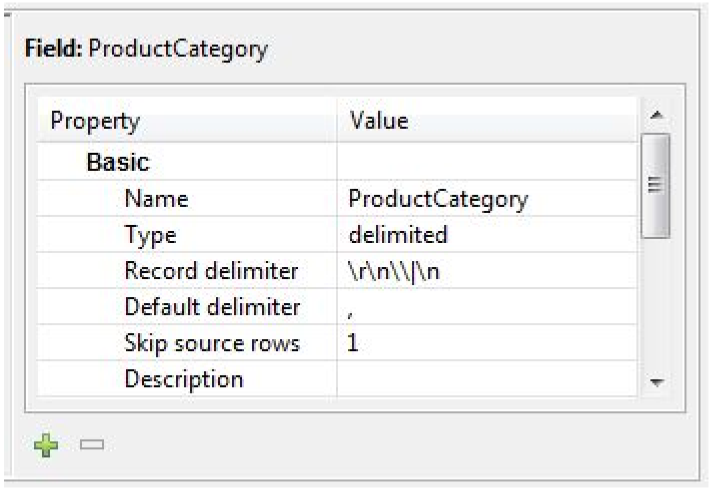
2. In the Details pane (to the right of the Record pane), check the **Record delimiter** property to see the default setting.
   In this example, **\r\n** is set as the record delimiter.



Field: ProductCategory

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

3. 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:



Field: ProductCategory

Property	Value
Basic	
Name	ProductCategory
Type	delimited
Record delimiter	\r\n\\|\n
Default delimiter	,
Skip source rows	1
Description	

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

Creating data stores

This section describes options for creating a data store.

Two options are available for creating data stores:

- server command
- Web services

In general, using the server command is the preferable option when creating a data store manually. While you can create a data store using Web services, this procedure is more complicated and prone to error.

After creating the data store, you may want to configure it. See *Configuring Data Stores on page 47*.

Creating a data store using the server command

This topic describes how to create a data store using the server command.

The Endeca Server must be running before you can create a data store.

You may want to add the **Oracle/Endeca/Server/<version>/endeca-cmd** directory to your Path before running server commands.

The following procedure illustrates the creation of an example data store named "bikestore".

To create a data store using the server command

1. Open a command prompt or terminal.
2. If you did not add the **Oracle/Endeca/Server/<version>/endeca-cmd** directory to your Path, change to that directory.
3. Enter the command `endeca-cmd create-ds bikestore`.

   The data store is created with the name you specified.
4. To test the data store, enter the command `endeca-cmd status-ds bikestore`.

   If the data store was created successfully and is running, the server returns the following message:

   ```
   Current State: Started

   Data Files: C:\Oracle\Endeca\Server\2.4.0\endeca-server\data\biekstore
   WS Port: 7777
   Bulk Load Port: 7778
   Startup Timeout (s): 60
   Shutdown Timeout (s): 60
   ```

   For additional information about creating a data store using the server command, see "create-ds command" in the *Oracle Endeca Server Administrator's Guide*.

   After creating the data store, you may want to configure it. See *Configuring Data Stores on page 47*.

Creating a data store using web services

This topic describes how to create a data store using web services.

The Endeca Server must be running before you can create a data store.

The following procedure illustrates the creation of an example data store named "bikestore".

To create a data store using web services:

1. Create an XML file following the model illustrated in the topic "Creating an Endeca data store" example in the *Oracle Endeca Server Developer's Guide*.
2. Submit the XML you created to the Endeca Server.
   It is recommended that you use a tool such as SOAPUI to submit the web services XML to the server.

   If the request is executed successfully, the server will return a success response similar to the following example:

   ```
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
   <createDataStoreResponse xmlns="http://www.endeca.com/endeca-server/control/1/0"/>
   </soap:Body>
   </soap:Envelope>
   ```

After creating the data store, you may want to configure it. See *Configuring Data Stores on page 47*.

Populating a data store

This section explains options for populating a data store.

Before populating the data store, you may want to configure it. See *Configuring Data Stores on page 47*.

Determining which updates to run

This topic discusses at a high level which types of updates are typically run, which helps you decide which types of graphs you need to create in the Integrator for your update purposes.

Typical data update strategies for an Endeca data store application include the following:

Table 3.1: Data loading strategies for Endeca data stores

Type of update	Description
Full initial load	This update is also known as 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 store. This update assumes that the Endeca data store is empty, and that the configuration and schema have only their default values acquired at when the Endeca data store was first created. As an example, the Baseline graph from the Quick Start project performs an initial data load.

Type of update	Description
Baseline update	This update is also known as a subsequent baseline or a re-baseline. Its basic idea is to replace almost everything in an Endeca data store index, and to avoid losing configuration changes that you may have already made interactively. This type of update is typically repeatable. It implies loading of the data into an Endeca data store index that already contains previously loaded data. Such an index may also contain configuration that has been changed from its defaults. Similarly, the attributes schema may have been modified. For a re-baseline, a typical graph would contain an **Export Config** component to export an existing configuration and schema, a **Reset Data Store** component that removes all records and schema and provisions a new Endeca data store, an **Import Config** component that imports the previously-exported configuration, and, finally, a set of components that load data. This set of sub-graphs may be run inside a **Transaction RunGraph**, in case you want them to take place atomically.
Incremental update	This update, also called a partial update, includes adding new records and making changes to the records and configuration that already exist in the Endeca data store. For an incremental update, a typical graph contains a **UniversalDataReader** and an **Add/Update Records** component.

Choosing a loader

This topic outlines the characteristics of the Endeca components used to load data into the data store. This information helps users determine which loader to choose.

The Integrator provides two loader components to support the loading of data into an Endeca data store: the Bulk Add/Replace Records component (often called the 'Bulk Loader"), and the Add/Update Records component.

When loading data, at least a portion of processing resources is devoted to loading processing; as a result, query performance is reduced while data is loading. The Add/Update 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 store, 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 store any time after the initial upload. Any new standard attributes that do not exist in the data store are automatically created with default system values.
- You want to replace existing records in the data store. 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 store configuration documents

Use the Add/Update 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 store with new records after the initial load of records. Any new attributes that do not exist in the data store are automatically created with system defaults.
- You are incrementally updating existing records in the data store following initial upload. The behavior depends on the multi-assign configuration of the standard attribute. If the multi-assign property is configured as true, uploaded data is totally additive; in other words, the loaded key-value pair will be merged into the existing record. If the multi-assign property is configured as false, and attribute values with the same.
- You are adding new records to the data store any time.

Note that this component cannot be used to load:

- the Global Configuration Record (GCR)
- managed attribute values (mvals)
- any data store 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 Add/Update Records connect 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.

Initial load

This section describes how to create graphs to perform the initial load of data into an Endeca data store.

An initial load of data is often called a “baseline update”. As the source records are ingested, they are converted into Endeca records and indexed by the Dgraph that services the data store.

You can perform an initial load either before or after loading the attribute schema, as described in *Configuring Attributes on page 49*. If you perform the initial load 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 (loading data, determining what standard and managed attributes and values you need, deleting your data, loading the updated configuration, and reloading your data) before you determine the configuration required to meet your needs.

Full load graphs

The Integrator Sample Applications includes a full load graph, named LoadData, that you can use as a model for building your own full load graphs.

Full 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 data base 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 10*.

- 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 to create a primary key, see *Configuring a Reformat component to generate a primary key on page 39*.

- 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 store. 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 132*.

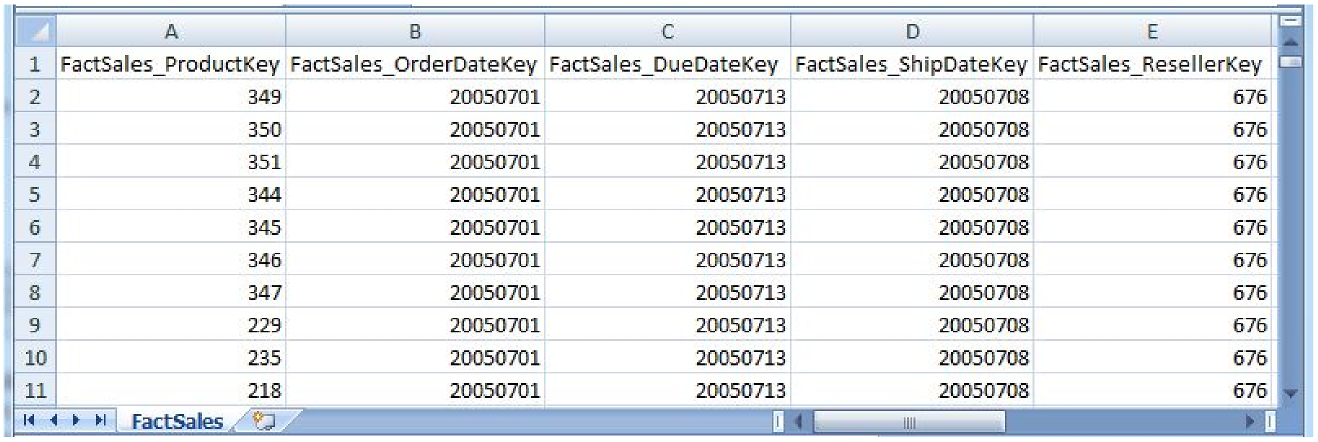
Source data format

Integrator 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 graph flow.

Use of hyphens in input property names

Although the Dgraph will accept attribute names with hyphens (because hyphens are valid NCName characters), the Integrator 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 **Spec Attribute** property. You may also want to configure other properties listed below.

- **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 39*.

- **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 store. For additional details, see the "Post ingest behavior" subsection of *Bulk Add/Replace Records component on page 132*.

  Defaults to `true`.

- **Post Ingest Dictionary Update**

  This field determines when the spelling dictionary will be updated. If the value of the field is `true`, the dictionary is updated immediately after the after the ingest operation finishes. If the value of the field is `false`, updating of the dictionary is disabled; the dictionary can be updated later use the `updateaspell` administrative command to the data store. For additional details, see the "Post ingest behavior" subsection of *Bulk Add/Replace Records component on page 132*.

  Defaults to `true`.

- **SSL Enabled**

  This field toggles communication with the Server and data store via SSL. Only set this property to `true` if SSL has been enabled on both the Endeca Server and the data store. 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 than the one used to separate records.

Incremental updates

You can incrementally update the data set in an Endeca data store, including adding new records.

Use the **Add/Update Records** component to incrementally update data in Endeca data stores. Incremental updates include:

- Adding a new record to the data set in the Endeca data store.
- Update an existing record by adding key-value pairs (assuming the attribute is configured to allow multi-assign).

You cannot delete records or record data using the **Add/Update Records** component. Nor can you overwrite an attribute value using this component. If you want to modify an attribute value, you must delete the value then add the new value. See *Selectively deleting data store data on page 44*.

Format of the incremental source input file

When you add incremental data to the data store, the records are generally in a similar format to the records that have already been loaded. Therefore, the format of incremental inputs is similar to the input for an initial load as described in *Source data format on page 37*.

How updates are applied

Records added incrementally are totally additive. In other words, if a record with the same primary key already exists in the Endeca data store, the key-value pairs list of the added record will be merged into the existing record.

If an Endeca attribute allows multi-assign (in other words, if the `mdex-property_IsSingleAssign` property is set to `false`)and data with the same name for the attribute but a different assigned value, the value of the key-value pair will be added as an additional value to the existing attribute. For example, if the existing record has one standard attribute named **Color** with a value of "red" and the request adds a **Color** property with a value of "blue", then the resulting record will have two **Color** key-value pair assignments.

If the attribute does not support multi-assign (in other words, if the `mdex-property_IsSingleAssign` property is set to `true`), you cannot add a second value to a single-assign attribute. In the **Color** example, if **Color** were a single-assign attribute and the record already had a value of "blue" and you attempted to add another assignment of "red", the operation would fail.

When adding standard attributes, if you add a new attribute:

- If the new attribute already exists in the Endeca data store but with a different type, an error is thrown and the new attribute is not added.
- If the new attribute already exists in the Endeca data store and is of the same type, no error is thrown and nothing is done.
- If the new attribute is supposed to be a primary-key attribute but a managed attribute already exists with the same name, an error is thrown and the new standard attribute is not added.

Note that updating a record can cause it to change place in the default order. That is, if you have records ordered A, B, C, D, and you update record B, records A, C, and D remain ordered. However, record B may move as a result of the update, which means the resulting order might end up as B,A,C,D or A,C,B,D or another order.

Creating an Incremental Update graph

This topic describes how to create a graph to incrementally load new records and updates to existing records.

Uploading new and updated data has the same requirements as the initial bulk upload. The data requires a primary key, and if the data stream includes multiple sources, those sources must be joined. The easiest way to create a graph to load these updates is to copy the initial load graph and substitute the Add/Update Records component for the Bulk Add/Replace Records component. You may also want to update the data readers to read updated data rather than the originally loaded data.

To create an incremental update graph:

1. In the Navigator pane, right-click on the load data graph you want to copy and from the pop-up menu, choose **Copy**.
2. Right-click on the **graph** folder and from the pop-up menu, choose **Paste**.

   Integrator displays the **Name Conflict** dialog. The value in the field defaults to `Copy of <graph name>` where *graph name* is the name of the graph you copied.
3. Enter a new name for the graph.

   For example, *UpdateData*.
4. Click **OK**.

   Integrator creates the new graph and adds it to the **graph** folder.
5. Open the new graph you created.
6. Open the Discovery section of the Palette and drag the Add/Update Records component into the graph.
7. Move the edge that connects to the Bulk Upload/Replace Records component to the Add/Update Records component.
8. Update the properties of the Add/Update Records component.

   You can copy the values of the following properties from the Bulk Add/Replace Records component:

   - **Endeca Server Host**
   - **Endeca Server Port**
   - **Data Store Name**
   - **Record Spec Attribute**
   - **Multi-assign Delimiter**

   You can also configure SSL Enabled to the same value as the Bulk Add/Replace Records component.

   The following properties of the Add/Update Records component are not used by the Bulk Load /Replace Records component:

   - Batch Size (bytes)

     This property sets the size (in bytes) of batches of records to submit to the Endeca server. If the size of the batch would prevent the submission of a complete record, the batch expands to incorporate the complete record. Any expansion of the batch size only affects that specific batch; subsequent batches use the configured size.

     Defaults to *1000000*.
   - Maximum number of failed batches.

     This property specifies that maximum number of batches that can fail before the ingest operation is ended. The default (0) allows no failed batches.

     Leave the default of 0 or specify a positive integer for this property.
9. An update graph should only read modified data. You may need to modify the data sources of the data readers, or you may need to modify the data readers to point at different data sources.
10. After you've made your changes, save the graph.

Managing data stores

This section describes processes for backing up, restoring, and deleting data store data.

Creating a data store snapshot

This topic describes how to create a data store snapshot using the Web Service component and the Endeca Server Administration Web Service.

A snapshot represents a data-layer view of the state of an Endeca data store index at a specific point in time. See "Capturing Snapshots" in the *Oracle Endeca Server Administrator's Guide* for more details about creating and restoring snapshots.

Use the `createSnapshotOperation` operation of the Administration Web Service to create a snapshot. The following code illustrates a trivial example:

```
<ns:request xmlns:ns="http://www.endeca.com/MDEX/admin/1/0">
<ns:createSnapshotOperation name="MySnapshot"
     path="file:///C:/Apps/MyApp/snapshots/" outerTransactionId="${OUTER_TRANSACTION_ID}"/>
</ns:request>
```

The `createSnapshotOperation` takes the following attributes:

- `name` specifies the name of the snapshot (that is, the name of the directory in which the snapshot files are put).
- `path` specifies the absolute URI path to the file system location, for the snapshot. The snapshot must be located on the same file system as the Endeca data store. The path should be specified in this format:

  ```
  file:///localdisk/dir
  ```

  where *dir* is the name of the directory in which the `name` snapshot is created.
- `OuterTransactionId` specifies the transaction ID. You can specify `${OUTER_TRANSACTION_ID}` as its value if you have that variable set in your project's `workspace.prm` file (note that the value can be blank). You can also remove this element if you are not using transactions.

Use a **WebServiceClient** component to run the snapshot operation.

Restoring a data store snapshot

This topic discusses how to restore a data store snapshot.

For information about restoring a data store snapshot, see "Restoring data files from a snapshot" in the *Oracle Endeca Server Administrator's Guide*.

Deleting data from the data store

This section describes options for deleting data from a data store.

Two options are available to delete data from a data store:

- If you want to delete all data, including data store configurations, use the Reset Data Store component to delete all data and reset the data store to a pristine state. Create a graph and add the Reset Data Store component; no other components are necessary. In addition to the **Endeca Server Host** and **Endeca**

Server Port, specify the **Data Store Name** of the data store you want to reset. See *Reset Data Store component on page 147*.

- If you want to delete only some data, use the Delete Data component. See *Selectively deleting data store data on page 44*.

Selectively deleting data store data

This topic describes how to delete complete records from an Endeca data store, or specific Key/value pairs from individual records.

Use the Delete Data component to selectively delete records or specific key/value pairs from and Endeca data store. This component requires an input file that specifies the records or key/value pairs (or both) to delete from the data store.

For details about the Delete Data component, see *Delete Data component on page 137*.

About the delete input file

The delete input file is a delimited file that uses a fixed schema and a specific ordering of the fields to specify the records or key/value pairs to delete. The file should be stored in the **data-in** folder of the project.

- The first row of the input file is the record header row. It requires a fixed schema:

  ```
  specKey|specValue|kvpKey|kvpValue
  ```

  where

  - *specKey* is the primary key (record spec) of the record
  - *specValue* is the primary key value
  - *kvpKey* is the name (key) of the Endeca standard or managed attribute of the key/value pair you want to delete. If both the kvpKey and the kvpValue are blank, the entire record is deleted. If the kvpKey is specified but the kvpValue is not, all values of the key/value pair are deleted. If the kvpValue is specified but the kvpKey is not, an error is thrown.
  - *kvpValue* is the value of the Endeca standard or managed attribute of the key/value pair you want to delete. If both the kvpKey and kvpValue are blank, the entire record is deleted. If the kvpKey is specified but the kvpValue is not, all values of the key/value pair are deleted. If the kvpValue is specified but the kvpKey is not, an error is thrown.
- The second and following rows of the input file specify the records and, optionally, key/value pairs to delete.

The following code illustrates an example input file:

```
specKey|specValue|kvpKey|kvpValue
ProductID|3000|Colors|green
ProductID|4000|Handling|
ProductID|5000||
```

When the component is run with this input file:

- In record 3000, the value "green" is removed from the standard attribute "Colors".
- In record 4000, all values of the standard attribute "Handling" are removed.
- All of Record 5000 is deleted from the Endeca data store.

Selective delete graphs

A selective delete graph usually consists of two components: a reader component (the Universal Data Reader component is usually a good option) and the Delete Data component. See *Delete Data component on page 137*. The two components are joined by an edge. See *Connecting two components with an edge on page 15*.

Configuring the Delete Data component

To configure the Delete Data 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.
- **Data Store Name**

  Enter the name of the data store from which you want to delete the data.
- **SSL Enabled**

  Set this field to `true` if you want to enable SSL. Only enable SSL on the component if you have enabled SSL on the Endeca Server.
- **Batch Size (Bytes)**

  Enter an integer greater than zero (0) to set the batch size, in bytes. To disable batching, specify 0 or a negative number.
- **Maximum number of failed batches**

  Enter a positive integer specifying that largest number of batches that can fail before the operation fails. To configure the operation to fail if any batch fails, enter 0.

Success console output

The following code illustrates the output to the Console tab for a successful selective delete operation:

```
INFO  [WatchDog] - Starting up all nodes in phase [0]
INFO  [WatchDog] - Successfully started all nodes in phase!
INFO  [ENDECA_DELETE_DATA0_0] - Sending in the last batch of deletes
INFO  [WatchDog] - [Clover] Post-execute phase finalization: 0
INFO  [WatchDog] - [Clover] phase: 0 post-execute finalization successfully.
INFO  [WatchDog] - ----------------------** Final tracking Log for phase [0] **---------------------
INFO  [WatchDog] - Time: 27/05/12 10:17:39
INFO  [WatchDog] - Node                   ID         Port      #Records         #KB aRec/s   aKB/s
INFO  [WatchDog] - ---------------------------------------------------------------------------------
INFO  [WatchDog] - UniversalDataReader    DATA_READER0                                 FINISHED_OK
INFO  [WatchDog] -  %cpu:..                          Out:0            3           0      3       0
INFO  [WatchDog] - Delete Data            ENDECA_DELETE_DATA0                          FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           In:0            3           0      3       0
INFO  [WatchDog] - ---------------------------------** End of Log **--------------------------------
INFO  [WatchDog] - Execution of phase [0] successfully finished - elapsed time(sec): 1
INFO  [WatchDog] - -----------------------** Summary of Phases execution **---------------------
INFO  [WatchDog] - Phase#            Finished Status         RunTime(sec)    MemoryAllocation(KB)
INFO  [WatchDog] - 0                 FINISHED_OK                        1              6119
INFO  [WatchDog] - ------------------------------** End of Summary **---------------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 1 (sec)
INFO  [main] - Freeing graph resources.
INFO  [main] - Execution of graph successful !
```

Chapter 4

Configuring Data Stores

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

About configuring data stores

All data store configuration procedures are optional.

Each data store is created with a set of defaults. When you load records, default configurations are also created. You only need to load data store 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 store 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"
- "Working with Search Interfaces"
- "Dgraph Configuration Reference"

Global Configuration Record

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

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, make sure you rebuild the aspell dictionary by running the `admin?op=updateaspell` administrative operation.

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

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 67*.

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 services 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

This section describes how to load standard and managed attribute configurations into a data store.

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

If you load data without first 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.

Recommended practice, however, is to define the attribute schema first by creating PDR and DDR input files, and loading these files to create the standard and managed attributes for the data store. Once you have created these attributes, you can then load managed attribute values (mvals). For details about loading mvals, see *Loading managed attribute values (MVals) on page 54*.

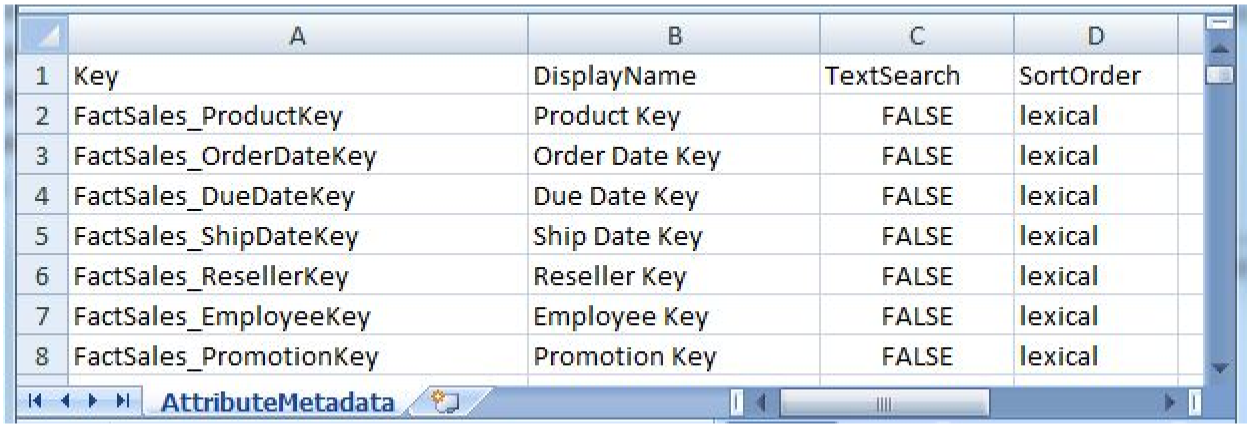
Loading the standard attribute schema

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 68*.

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 26*.

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 user hyphens in the names of standard attributes. Although the data store will accept standard attributes names that include hyphens, the Integrator 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`

Input Header Property	Maps to PDR Property
`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 Transformer 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.
  $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 (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 the performance of both the Integrator 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/1/0">
  <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 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.

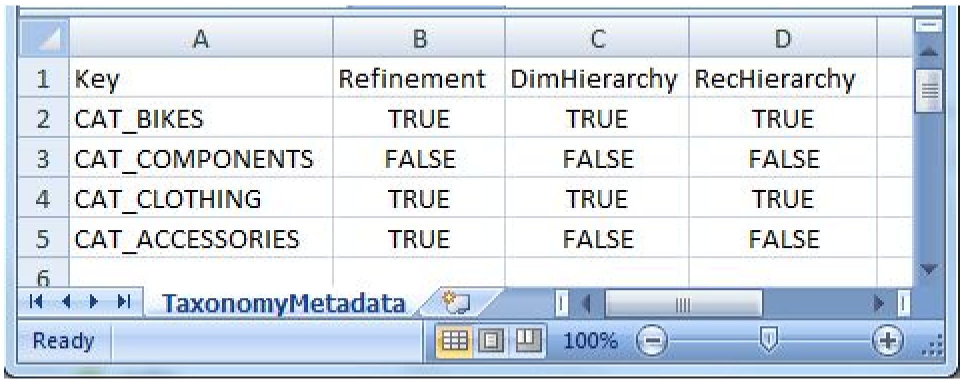
Loading the managed attribute schema

This topic describes implementation details of graphs to load the Dimension Description Records (DDRs) 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 68*.

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`	`mdex-dimension_Key`
`Refinement`	`mdex-dimension_EnableRefinements`
`DimHierarchy`	`mdex-dimension_IsDimensionSearchHierarchical`
`RecHierarchy`	`mdex-dimension_IsRecordSearchHierarchical`

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 (aggKey).

Next, the transform() function builds the DDRs by creating the `<mdex:record>` container, and populating it with `<mdex-dimension_EnableRefinements>`, `<mdex-dimension_IsDimensionSearchHierarchical>`, and `<mdex-dimension_IsRecordSearchHierarchical>`. 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 the Integrator 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/1/0">
  <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 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 managed attribute values (MVals)

This topic describes how to implement a graph to load values for a managed attribute.

Recommended practice is to load managed value attributes after loading standard attributes and managed attributes (PDRs and DDRs). If you add managed attribute values for an attribute that has not already been defined in the data store, the managed attribute is created with system default values for the DDR and, if necessary, for the PDR.

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

Managed attribute value input file

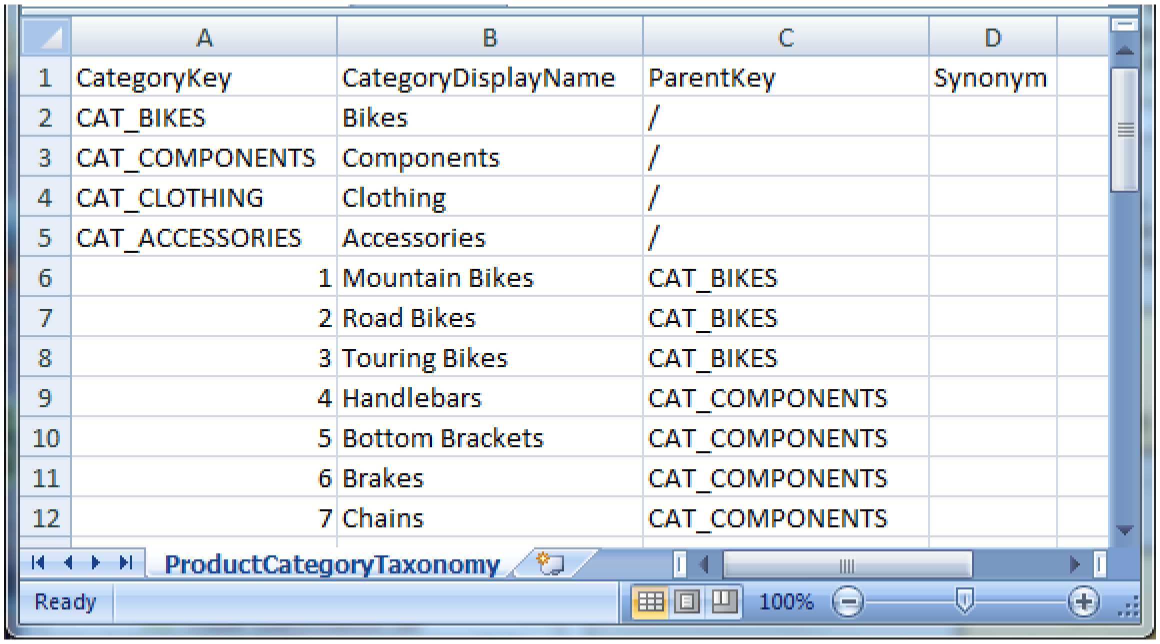
The input schema of the managed attribute input file is predefined. The first row is the header row and must define the following properties:

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

where:

- *spec* is a unique string identifier for the managed value. This ID is used to associate child values with parent values.
- *displayname* is the name displayed in the user interface for the managed value.
- *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 (/) as for this property. If the value is a child managed value, enter the unique ID of the parent managed value.
- *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.

The following graphic illustrates a simple managed attribute input file:



	A	B	C	D
1	CategoryKey	CategoryDisplayName	ParentKey	Synonym
2	CAT_BIKES	Bikes	/	
3	CAT_COMPONENTS	Components	/	
4	CAT_CLOTHING	Clothing	/	
5	CAT_ACCESSORIES	Accessories	/	
6	1	Mountain Bikes	CAT_BIKES	
7	2	Road Bikes	CAT_BIKES	
8	3	Touring Bikes	CAT_BIKES	
9	4	Handlebars	CAT_COMPONENTS	
10	5	Bottom Brackets	CAT_COMPONENTS	
11	6	Brakes	CAT_COMPONENTS	
12	7	Chains	CAT_COMPONENTS	

This example defines several parent categories, including CAT_BIKES and CAT_COMPONENTS. The following values are defined as children of the CAT_BIKES value:

- Mountain Bikes
- Road Bikes
- Touring Bikes

and these values are defined as children of the CAT_COMPONENTS value:

- Handlebars
- Bottom Brackets
- Brakes
- Chains

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

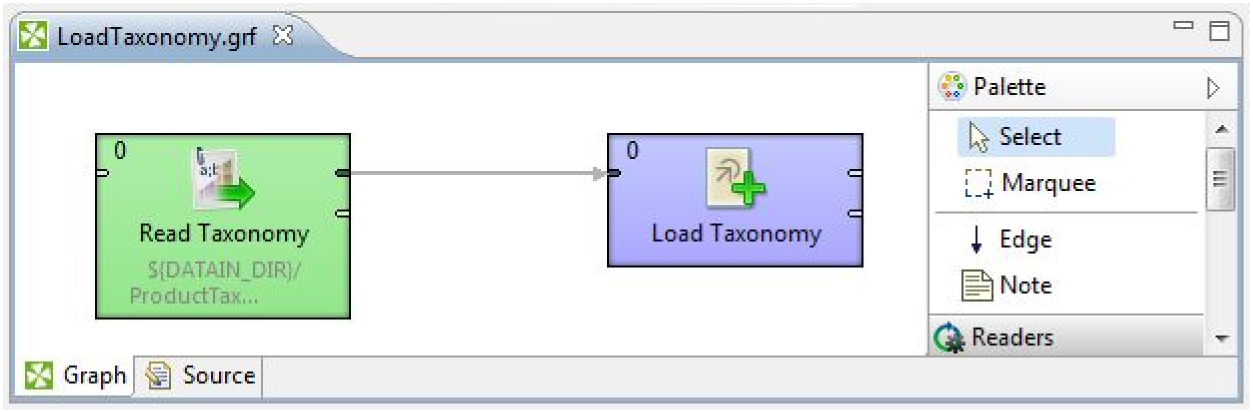
Managed attribute value input graph

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.
- An Add Managed Values component

The two components are joined by a basic edge. See *Connecting two components with an edge on page 15*. The following graphic illustrates a typical configuration:



Configuring an Add Managed Values component to load managed attribute values

This topic describes how to configure an Add Managed Values Connector in a load managed attribute values graph.

To configure an Add Managed Values component:

1. In the Palette, open the Discovery section and drag an Add Managed Values component into the graph.
2. Double-click on the Add Managed Values component.
   Integrator displays the Edit Add Managed Values dialog.
3. The values of the **Endeca Server Host**, **Endeca Server Port**, and **Data Domain Name** field default to the global variables for these values (`${ENDECA_SERVER_HOST}`, `${ENDECA_SERVER_PORT}`, and `${DATA_DOMAIN_NAME}`). You can change these values.
4. Enter the **Managed Attribute Name** for the managed attribute to which you want to add values. You can only enter one name. If you enter more than one name, the graph fails with an error.

5. If your input file includes multiple synonyms for any value, enter the **Synonym Delimiter**.
6. Click OK to save the configured component.

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 67*.

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 (`RELRANK_INTERP`) and Field (`RELRANK_FIELD`) relevance ranking modules:

```
<RELRANK_STRATEGIES>
  <RELRANK_STRATEGY NAME="ProductRelRank">
    <RELRANK_INTERP/>
    <RELRANK_FIELD/>
  </RELRANK_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 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 snippeting for attributes

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

```
<RECSEARCH_CONFIG/>
```

Use the `RELRANK_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_RELRANK_STRATEGY="All" NAME="Surveys">
    <MEMBER_NAME RELEVANCE_RANK="1">SurveyResponse</MEMBER_NAME>
 </SEARCH_INTERFACE>
 <SEARCH_INTERFACE DEFAULT_RELRANK_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_RELRANK_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 have already configured that relevance ranking strategy. See *Configuring relevance ranking on page 57*. 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 67*. 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 `dismsearch_config` document in the server with the value of the `$xmlString` variable in the `DIMSEARCH_CONFIG` 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 67*.

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 67*.

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.

Loading stemming files

This section describes how to build a graph to load stemming files.

The stemming feature of Endeca data store allows the system to consider alternate forms of individual words as equivalent when processing matches for search queries.

Note: For details about the stemming feature, see the *Oracle Endeca Server Developer's Guide*.

The default language for stemming is English, but you can overwrite the default English stemming with stemming for the following languages:

Table 4.1: Language support for stemming

Language	Stemming File
Dutch	`nl_word_forms_collection.xml`
French	`fr_word_forms_collection.xml`
German	`de_word_forms_collection.xml`
Italian	`it_word_forms_collection.xml`
Portuguese	`pt_word_forms_collection.xml`
Spanish	`es_word_forms_collection.xml`

These stemming files are stored in the *lang/stemming* directory in the root of the Integrator installation. The default stemming file (`en_word_forms_collection.xml`) is also stored in this location.

When you load a new stemming file, the Dgraph rebuilds the stemming dictionary with the new word forms.

Note the following considerations when changing the stemming file for the data store:

- A data store only supports one stemming language at a time. When you load a new stemming file (in other words, change the stemming forms to a different language), it overwrites the current stemming file (in other words, the stemming forms for the new language replace the stemming forms for the existing language). Multiple language stemming forms cannot run concurrently.
- When loading all stemming files using the `putConfigDocuments` operation of the Configuration Web Service, you must specify `en_word_forms_collection.xml` as the value of the name attribute of the `<mdex:configDocument>` element of the request, regardless of the language you are loading. Even when loading a stemming file for a language other than English, you must specify this value:

  ```
  <mdex:configDocument name="en_word_forms_collection.xml">
  ```

Load stemming graph

The load stemming graph consists of two components:

- A Universal Data Reader

  This component reads the input stemming file. In the **File URL** field, specify the stemming file you want to load from *lang/stemming* directory.
- A Web Services Client

  This component runs the web services request to update the stemming configuration.

The two components are joined using a basic edge. See *Connecting two components with an edge on page 15*.

Web services request

Use the `putConfigDocuments` operation of the Configuration Web Service to load the stemming file to the data store.

Note: The value of the `name` attribute of the `configDocument` node must be `en_word_forms_collection.xml` regardless of the actual stemming file you are uploading.

The following code illustrates a typical example of the web services request to load stemming files:

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

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

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 4.2: 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 `triggerAttributeValue` 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`	If the trigger is a managed attribute, `isLeafTrigger` specifies a Boolean value (that must be in lower case) that denotes the type of the trigger attribute value: • If `true`, 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). • If `false` (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). Note that `isLeafTrigger` 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.

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
```

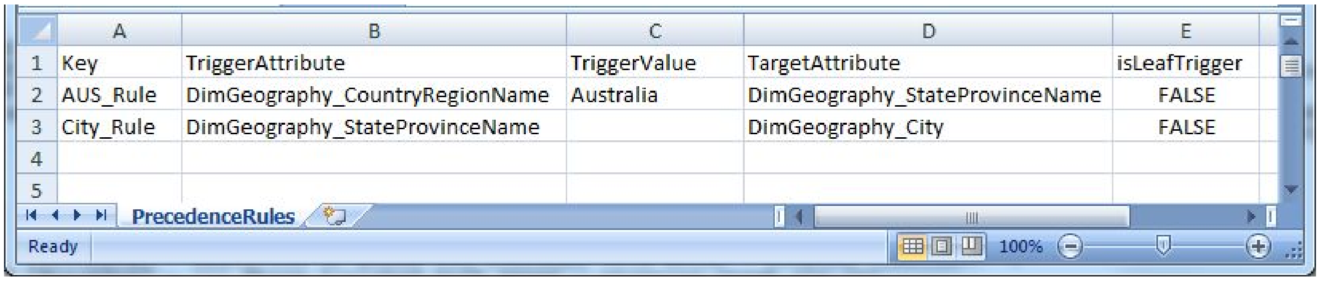
The names of the header properties are arbitrary. For example, you can use `RuleName` instead of `Key`). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).

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

Table 4.3: 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* 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:

- a 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.

- a 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 67* for details about the implementation of this component.

- a 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 15*.
- 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 68*.

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 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:

- a 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.

- a Transformer

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

- a 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 67* for details about the implementation of this component.

- a 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 15*.
- 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 68*.

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 (DDRs) 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 store 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 54* 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

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 71* 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 export.
  - 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 71* for details about creating an export/import edge.

Configuring edges for export and import graphs

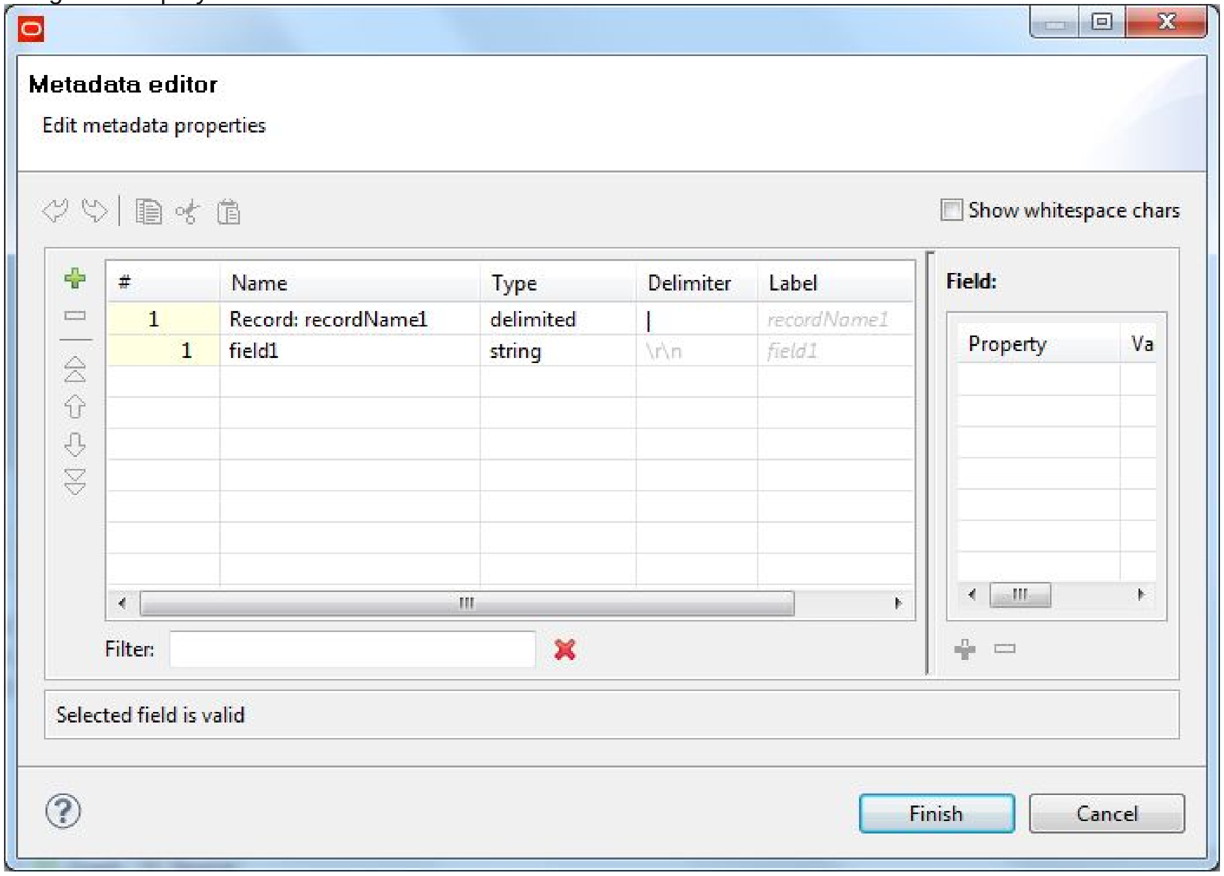
This topic describes how to configure the metadata of edges in export and import graphs.

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.



Metadata editor

Edit metadata properties

Show whitespace chars

#	Name	Type	Delimiter	Label
1	Record: recordName1	delimited	|	recordName1
1	field1	string	\r\n	field1

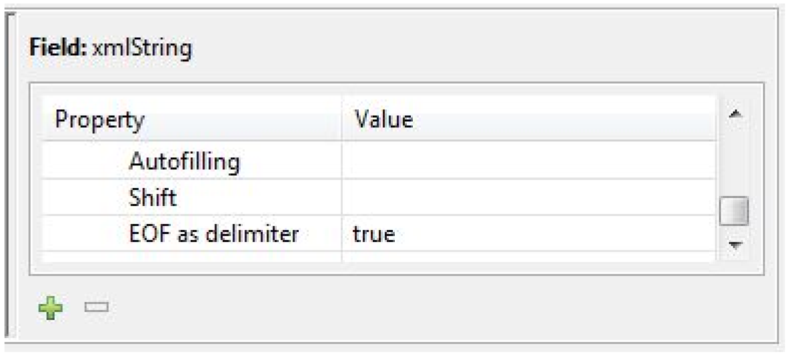
Field:

Property | Va

Filter:

Selected field is valid

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:



Property	Value
Autofilling	
Shift	
EOF as delimiter	true

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

Use Integrator to export and restore View definitions.

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

Note: In Endeca Latitude Version 2.2 and earlier, Views were called "Semantic Entities".

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 SConfig web service 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/1/0">
      <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:
   - Universal Data Reader
   - HTTP component
   - Universal Data Writer
4. In the Universal Data Reader, configure the File URL property with the path and name of the input file. You can use the `${DATAIN_DIR}` global variable. For example: `${DATA_IN}/soap.xml`.
5. Configure the HTTP Connector with the following values:

Property	**Value**
URL	`http://${ENDECA_SERVER_HOST}:${ENDECA_SERVER_PORT}/ws/sconfig/${DATA_STORE_NAME}` where • `${ENDECA_SERVER_HOST}` is the name or IP address of the Endeca server machine. You can also use the `${ENDECA_SERVER_HOST}` global variable for host portion of the URL. • `${ENDECA_SERVER_PORT}` is the port of the Endeca server. You can also use the `${ENDECA_SERVER_PORT}` global variable for this portion of the URL. • `${DATA_DOMAIN_NAME}` is the name of the data domain whose views you want to export. You can also use the `${DATA_STORE_NAME}` global variable.
Request Method	POST
Input Field	XML
Output Field	XML

6. In the Universal Data Writer 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_congif.xml`.

7. Add edges to connect the Universal Data Reader to the HTTP component, and the HTTP component to the Universal Data Writer.
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 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*. In the **Delimiter** field, leave the default value.
   (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 quickstart 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.

You can use the example provided in the Import View Configuration design pattern on the Integrator Design Patterns section of the Oracle wiki: *https://wikis.oracle.com/display/endecainformationdiscovery/Import+View+Configuration*. The following procedure illustrates how to build a View import graph using this example.

To build an import View configuration graph

1. Create a new graph.
2. Open a browser and go to *https://wikis.oracle.com/display/endecainformationdiscovery/Import+View+Configuration*
3. Copy the graph code snippet located below the illustration of the graph.
4. In your new graph, open the **Source** tab.
5. Paste the code snippet you copied from the Import View Configuration design pattern page into the **Source** tab.
6. 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 outer transactions

An ***outer transaction*** is a set of operations performed in the Oracle Endeca Server data store that is viewed as a single unit.

If an outer transaction is committed, this means that all of the data and configuration changes made during the transaction have completed successfully and are committed to the data store index.

If any of the changes made within a transaction fail to complete successfully, the outer transaction fails to commit and remains open (only one outer transaction can be open at a time). In this case, you can roll back the entire transaction, and the changes to the data store index do not occur.

In general, the best practice is to set up operations so that successful updates are automatically committed (this is the default), but failed updates can be rolled back either automatically or manually.

The Transaction Web Service of an Endeca data store is used for controlling outer transactions. For more information on this interface, see the *Oracle Endeca Server Developer's Guide*.

Note: In Version 2.3.0 the Transaction Run Graph component was introduced. Outer transactions should be implemented using this component. Older outer transaction graphs should be re-implemented using this component. Techniques used in version 2.2.0 and earlier to implement outer transactions should no longer be used.

When to use outer transactions

This topic discusses outer transactions and provides recommendations for when it is useful to run your Integrator graphs inside an outer transaction as opposed to running individual graphs for various tasks.

Typically, Integrator components load data and configuration into an Endeca data store by making web service requests or requests to an ingest interface (the Data Ingest Web Service or the Bulk Load Interface). Each web service request represents its own set of operations in the Endeca data store, and succeeds or fails on its own — it is in itself a transaction. These transactions, because they do not include any other transactions inside them, are also known as inner transactions. If some inner transactions in the Endeca data store succeed and others fail, the resulting Endeca data store may reflect only a partially updated data set (if, for example, some updates did not succeed).

Typically, however, you may want to ensure that data changes from an entire data-updating graph either complete or fail as a unit, so that the resulting set of data files represents an entirely updated data store. You may also want to make sure that end users do not access intermediate states of the data in Studio, but instead can only have access to the pre-update state of the data files (while the data-updating graph completes), and then seamlessly transition to the data store after it has been fully updated.

To guarantee that your updates either completely succeed or fail, use a graph that runs an outer transaction.

An ***outer transaction*** is a set of operations performed in the Endeca data store that is viewed as a single unit. If an outer transaction is committed, this means that all of the data and configuration changes made during this transaction have completed successfully and are committed to the Endeca data store.

To run an outer transaction, use the Endeca component **Transaction RunGraph** in the Integrator graph. This component lets you create a graph that starts and commits an outer transaction in the Endeca data store, utilizing calls to the Transaction Web Service. Using this component, you can add sub-graphs and components that will run inner transactions inside an outer transaction. Typically, a graph that runs an outer transaction is useful for running updates. Once such a graph completes, an update to your records is guaranteed to be fully committed to the Endeca data store.

Considerations for running graphs within outer transactions

When planning to run graphs within outer transactions, consider the following issues.

- Only one outer transaction can run in an Endeca data store at a time. If you run a graph that starts an outer transaction (in other words, if you run a graph built with the **Transaction RunGraph** component), do not start another graph that starts another outer transaction. If you start a second outer transaction graph, the server returns a transaction fault for that graph.
- All Integrator components can be used in graphs that run within outer transactions. In other words, all components are transaction-friendly.
- When components that update the data store (whether through the Data Ingest Web Service [DIWS]) or the Bulk Ingest Interface) run within an outer transaction, the operations reference the outer transaction ID in their calls to the server. The transaction ID is passed from the **Transaction RunGraph** component.
- For the duration of the outer transaction, the component sets the Outer Transaction ID to `transaction`.
- The `OUTER_TRANSACTION_ID` variable should be defined as a global variable in the `worskpace.prm` file for your project. When defined in `worskpace.prm` as a global variable, the variable should be empty(`OUTER_TRANSACTION_ID=`). In general the value of any "outer transaction" properties in graph components should also be set to `${OUTER_TRANSACTION_ID}` so they can use this global variable. This practice allows the same graphs to be run both independently and as part of an outer transaction without modifying either the graph components or `worskpace.prm`.
- If a graph can be run either as part of an outer transaction or independently, and that graph also updates the data store, you cannot run the graph independently at the same time as you run an outer transaction that contains it. For example, suppose you create a graph that performs an incremental update and you wrap it in an outer transaction. If you start the outer transaction, you cannot also run the incremental update independently while the outer transaction is running.

- If you use a **WebServiceClient** component to run Endeca web services that modify the data store, the **Request Structure** field for the component must include an `OuterTransactionId` element with an ID value of an outer transaction. This element must be specified first in the request. 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:configTransaction>
  ```

  In this example, the string: `<config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>` references the ID of the outer transaction listed in the `workspace.prm` file for your project.

  Note: The `createSnapshotOperation` has an `outerTransactionId` attribute rather than including an `OuterTransactionId` element in the request.

- Consider creating all your data-updating graphs inside a graph that starts and commits an outer transaction.

For more information on outer transaction behavior, see the *Oracle Endeca Server Developer's Guide*.

Wrapping existing graphs in an outer transaction

You can wrap any graphs in a graph that uses the **Transaction RunGraph** component.

To wrap graphs in an outer transaction:

1. Create an `OUTER_TRANSACTION ID` parameter in the project workspace parameters file `workspace.prm` with an empty value:

   ```
   OUTER_TRANSACTION_ID=
   ```

2. Add the following element as the first element in the outermost request element of any **WebServiceClient** or **HTTPConnector** that could be called from within an outer transaction:

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

   These configurations ensure that the component behaves like a custom component of the Endeca data store: when this value is non-empty, the component will run within an outer transaction; when this value is empty, it will be ignored and the component will run outside an outer transaction.

3. Create a graph that includes a **Transaction RunGraph** component to reference one or more graphs, and run it.

   The Integrator will start an outer transaction named `transaction` (lowercase, case-sensitive) and run all sub-graphs within it.

Creating a Transaction RunGraph graph

This section describes how to build an Integrator graph that uses the **Transaction RunGraph** component to run a series of graphs within a single outer transaction.

To run one ore more graphs within an outer transaction, create a master graph using the **Transaction RunGraph** component.

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

1. It starts an outer transaction using the Transaction Web Service.
2. It runs a series of defined sub-graphs within that transaction.
3. It commits the outer transaction when all the graphs have successfully finished.

For the duration of the outer transaction, **Transaction RunGraph** is designed to override the transaction ID specified in `workspace.prm` with the string `transaction`. Components that run within this graph automatically pick up this ID.

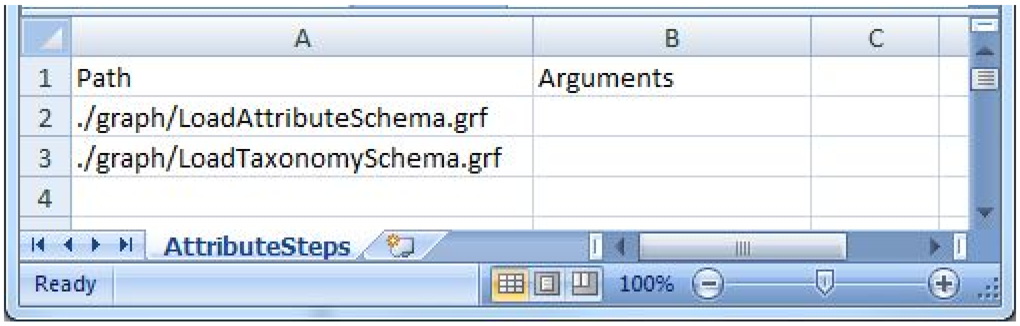
In addition, you can configure the **Transaction RunGraph** component to react to unsuccessful runs. Common practice is to roll back failed outer transactions.

In this section, a sample **Transaction RunGraph** graph will be built to run the two graphs that load the standard attribute and managed attribute schemas into the data store.

Format of the steps input file

The input file for the **Transaction RunGraph** component defines which graphs will be run by it.

The `AttributeSteps.csv` sample input file used to list the graphs resembles the following illustration:



	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 than those used in this illustration. The properties are delimited (for example, by the comma in the example CSV file). After the header row, 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 name of a graph to be run by the **Transaction RunGraph** component. The graphs are run 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.

After creating the file, copy it into the **data-in** folder.

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 79*.

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 80*.
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 recommended practice describe 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 store 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 store 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 store ("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

Using Endeca Components

This chapter explains how to use the components provided with the Integrator. For details about the functionality and configuration of Endeca components, see the Component Reference.

Adding Key-Value Pairs

This section describes how to add key-value pairs to Endeca records.

About key-value pair data

Use the **Add KVPs** component to add key-value pair data to records in an Endeca data domain.

Consider using the **Add KVPs** component in the following cases:

- You want to ingest source data that is stored in a key-value pair format instead of the more traditional rectangular data model.
- You do not know the data schema when developing the graph.

In either of these cases, you can also consider loading data in rows using the **Add/Update Records** component. That approach will be faster than loading the same data as key-value pairs.

Format of the KVP input file

The metadata schema of the **Add KVPs** 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:

```
specKey|specValue|kvpKey|kvpValue|mdexType
```

The following table provides details of these properties:

Table 7.1: Add KVP input 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.
`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 *Standard attribute default values on page 26*.
`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 *Supported data types on page 25*.

The following code illustrates a simple example of an input file for the **Add KVPs** component:

```
specKey|specValue|kvpKey|kvpValue|mdexType
ProductID|51841|Designation|Professional use|
ProductID|48191|Color|Crimson|
ProductID|48191|Color|Sea Blue|
ProductID|48197|Component|road rim|
ProductID|48197|Location|42.365615 -71.075647|mdex:geocode
```

When this example is loaded:

- The attribute "Designation" is added to Record 51841 with a value of "Professional use".
- Two values ("Crimson" and "Sea Blue") are added to the Color attribute Record 48191. This attribute is evidently a multi-assign attribute.
- The attribute "Component" is added to Record 48197 with a value of "road rim".
- A new standard attribute, "Location", is created in the data domain with a type of geocode. This attribute is added to Record 48197 with a value of "42.365615 -71.075647."

Using the Add Key-Value Pairs component in a graph

This topic describes considerations for using the **Add KVPs** component in a graph.

When you add the **Add KVPs** component to a graph, configure the following parameters:

- **Endeca Server Host**

  Enter the name or IP address of host machine of the Endeca server. Value defaults to `localhost`. You can also use the `${ENDECA_SERVER_HOST}` global variable if the value of the global variable is defined in the `workspace.prm` file for the project.
- **Endeca Server Port**

  Enter the port on which the Endeca server listens. Value defaults to `7770`. You can also use the `${ENDECA_SERVER_PORT}` global variable if the value of the global variable is defined in the `workspace.prm` file for the project.
- **Data Domain Name**

  Name of the Endeca data domain that you want to load or whose records you want to modify.
- **SSL Enabled**

  Only toggle this flag to *true* if SSL has been enabled on the Endeca Server.
- **Batch Size (Bytes)**

  Specify the size of batches to be submitted to the Endeca Server, in bytes. If necessary, a batch will exceed this size to accommodate a final record, but subsequent batches return to the specified size. Specifying 0 or a negative number disables batching. Defaults to 1000000.
- **Maximum number of failed batches**

  Specify the maximum number of batches that can fail before the ingest operation as a whole is terminated, as a positive integer. Defaults to 0, which allows no failed batches.

You can use a basic edge to connect both to and from an **Add KVPs** component. See *Connecting two components with an edge on page 15*.

Loading Taxonomies

You can load an externally managed taxonomy (EMT) into an Endeca data doamin.

For details, see *Loading managed attribute values on page 54*.

Add/Update Records

The primary use of this component is to add new records and update existing records in the Endeca Server.

For details about using this component, see *Incremental Updates on page 40*.

For information about choosing between the Add/Update Records component and the Bulk Add Replace Records component, see *Choosing a loader on page 35*.

Using the Bulk Add/Replace Records component

The primary use of the Bulk Add/ Replace Records component is to load and reload data to the Endeca data domain.

For details, see *Initial load on page 37*.

For information about choosing between the Bulk Add Replace Records component and the Add/Update Records component, see *Choosing a loader on page 35*.

Using the Delete Data component

Use the Delete Data component to delete data from a data domain selectively.

For details, see *Selectively deleting data store data on page 44*.

Using the Export Config and Import Config components

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

For details, see *Exporting and importing data doamin configurations on page 71*.

Using the Record Store Reader

Use the Record Store Reader component to read records from an Endeca Content Acquisition System (CAS) record store instance.

Endeca Content Acquisition System (CAS) provides the ability to crawl a variety of data sources to acquire data for Oracle Endeca Information Discovery applications. Data sources CAS can crawl include file systems, content management systems, Web servers, and custom data sources. CAS converts the content of the documents and files it finds into Endeca records, which it stores in a record store instance. CAS can crawl these data sources multiple times, updating the data domain with new records for the new content and data it finds as well as updating existing data.

Use the Record Store Reader component to read records from a CAS record store into an Integrator graph. The graph can then use an ingest component (such as the Add/Update Records component) to load the records into an Endeca data domain. Note 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 CAS record store:

- Set the value of the **CAS Read Type** field to `Full Extract` to read the latest version of all records currently stored in the record store. This configuration is best used for a full loads, such as an initial load or a baseline update. See *Initial load on page 37*.
- Set the value of the **CAS 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 **CAS Client ID** field. This configuration is best used for incremental updates to a data domain. See *Incremental updates on page 40*.

Prerequisites

Before using the Record Store reader, you must:

- Install Endeca Content Acquisition System (CAS) Version 3.0.x.
- Create a named record store instance.
- Configure and execute a crawl to populate the record store instance.

For full information on CAS, including how to create a record store instance and how to configure and run a crawl, see the *Endeca CAS Developer's Guide*. You can download the Endeca Content Acquisition documentation set from the Endeca Information Discovery product page on Oracle Technology Network (OTN): *http://www.oracle.com/technetwork/middleware/endeca/documentation/documentation-1544963.html*

Configuring the Record Store Reader

As the name implies, the Record Store Reader reads data into a graph.

When configuring the Record Store Reader:

- In the **CAS Service Host**, specify the name or IP address of the machine where the Content Acquisition Service (CAS) resides.
- In the **CAS Service Port** field, specify the port on which the CAS service is listening. Defaults to `8500`.
- In the **CAS Record Store Instance** field, specify the name of the Record Store whose records you want to read into the graph.
- In the **CAS Client ID** field, enter a name that identifies the Integrator client to the Record Store. Defaults to Integrator. 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. If multiple graphs access the same Record Store, you should specify a different value in this field for each graph.
- In the **CAS Read Type** field, choose the type of read you want to perform:
  - **Full Extract**

    This option reads all records current committed 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 **CAS Client ID** field.
- Only check the **SSL Enabled** box if you have enabled SSL on the Endeca server.
- You can accept the default **Multi-assign delimiter** (the Unicode DELETE character [\U007F]) or change it if you use a different character to separate values in multi-assign fields.
- The **Read Batch Size** field specifies the number of records in each batch fetched from the CAS Record Store. The default is 100 records, but you can specify a different batch size.
- The **Socket Timeout** specifies the maximum period of inactivity between two consecutive data packets when fetching data from the Record Store. The default is 300000 milliseconds (5 minutes), but you can change different time period.

Generating Edge metadata with the Record Store Wizard

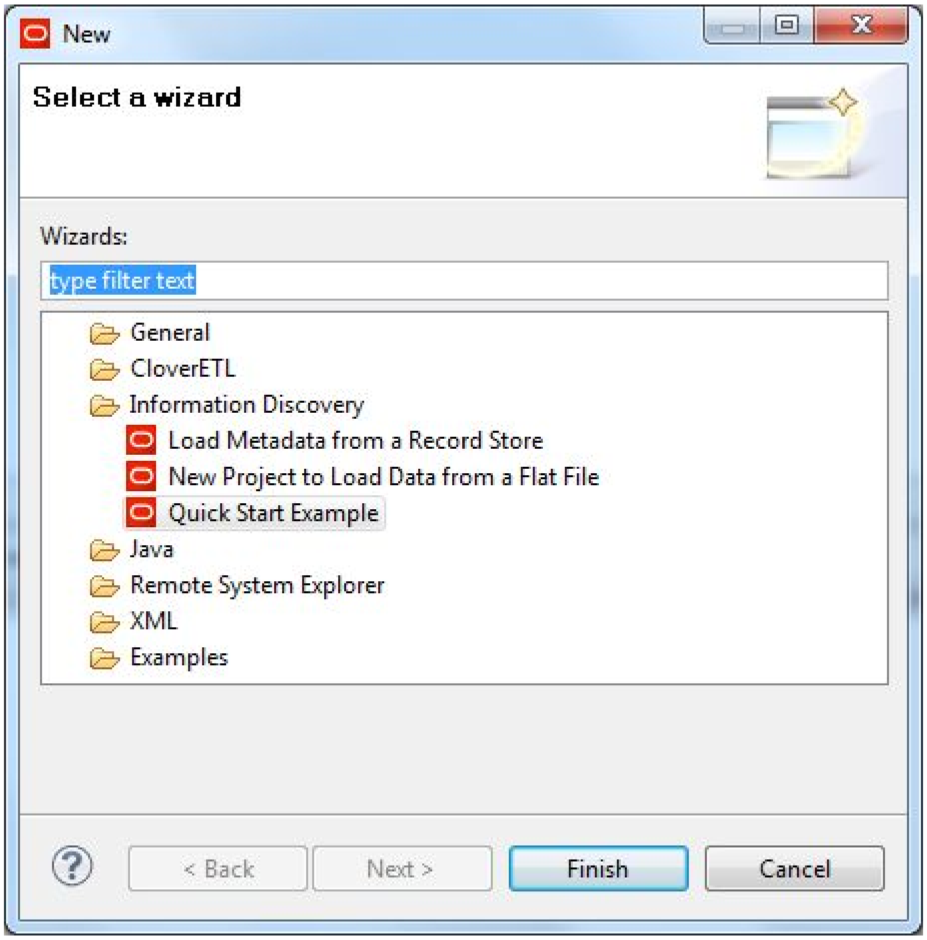
You can use the Record Store Wizard to generate an external metadata file by querying the Record Store instance for its properties.

Before you can use the Record Store Wizard:

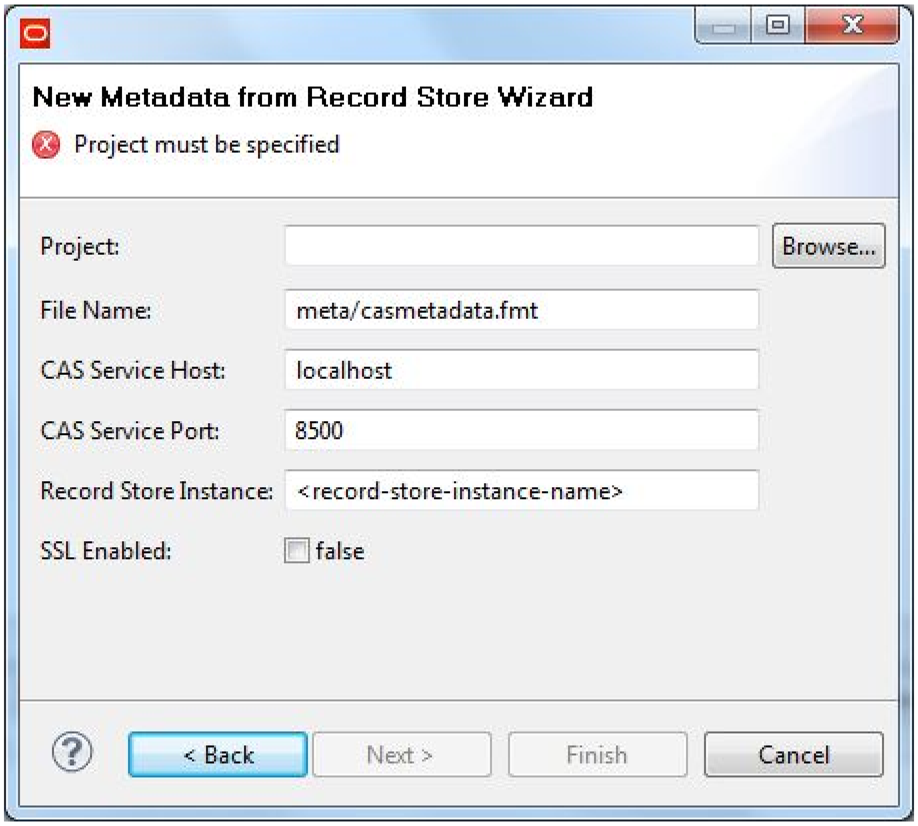
1. You must have created a Record Store instance.
2. The Record Store instance must have at least one committed generation of records. That is, you must have run a full (baseline) crawl with the output sent to the Record Store.
3. The Endeca CAS Service must be running so that the Record Store Wizard can connect to it.
4. You must have created an Integrator project and graph for the **Record Store Reader** component, as you will store the metadata file in this project.

To generate an external metadata file from the Record Store instance:

1. From your Record Store Reader project, select **File>New>Other**.
   Integrator 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 displays the Record Store Wizard.

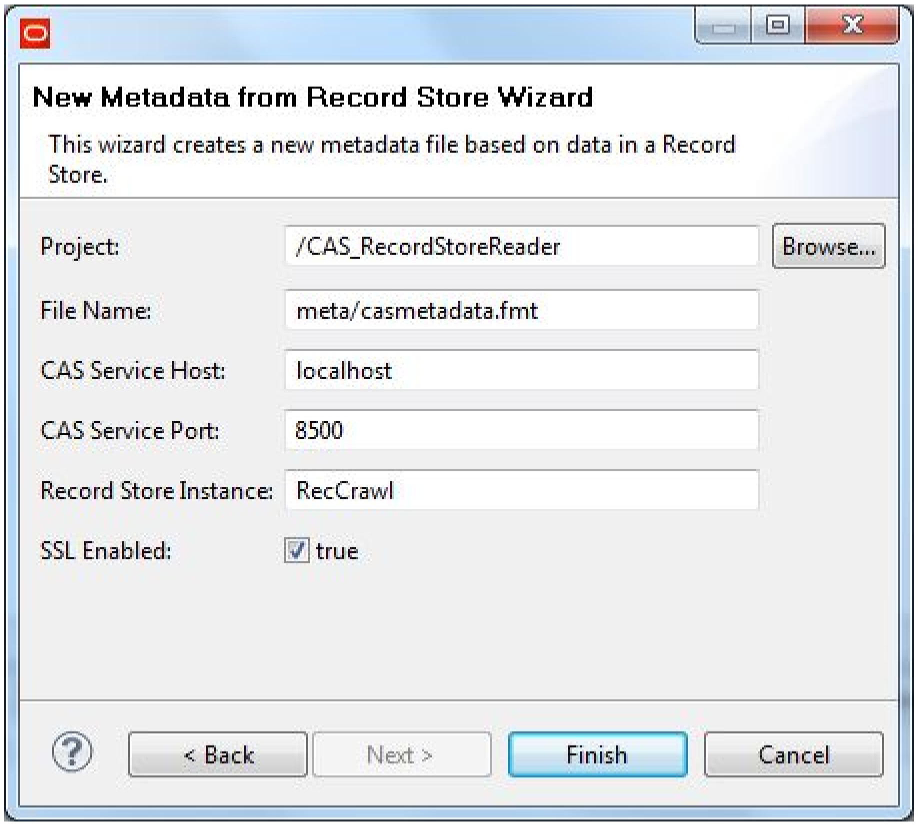

New Metadata from Record Store Wizard

Project must be specified

Project:	
File Name:	meta/casmetadata.fmt
CAS Service Host:	localhost
CAS Service Port:	8500
Record Store Instance:	<record-store-instance-name>
SSL Enabled:	false

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: `casmetadata.fmt`.
   (c) In the **CAS Service Host** field enter the name of the machine on which the Endeca CAS Service is running. This name should be the same as the name in the **CAS Service Host** configuration property of the **Record Store Reader** component.
   (d) In the **CAS Service Port** field enter the port of the Endeca CAS Service. This port should be the same as the one in the **CAS Service Port** configuration property of the **Record Store Reader** component.
   (e) 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.
   (f) Only toggle the **SSL Enabled** field to true if the Endeca CAS Service is SSL enabled.

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

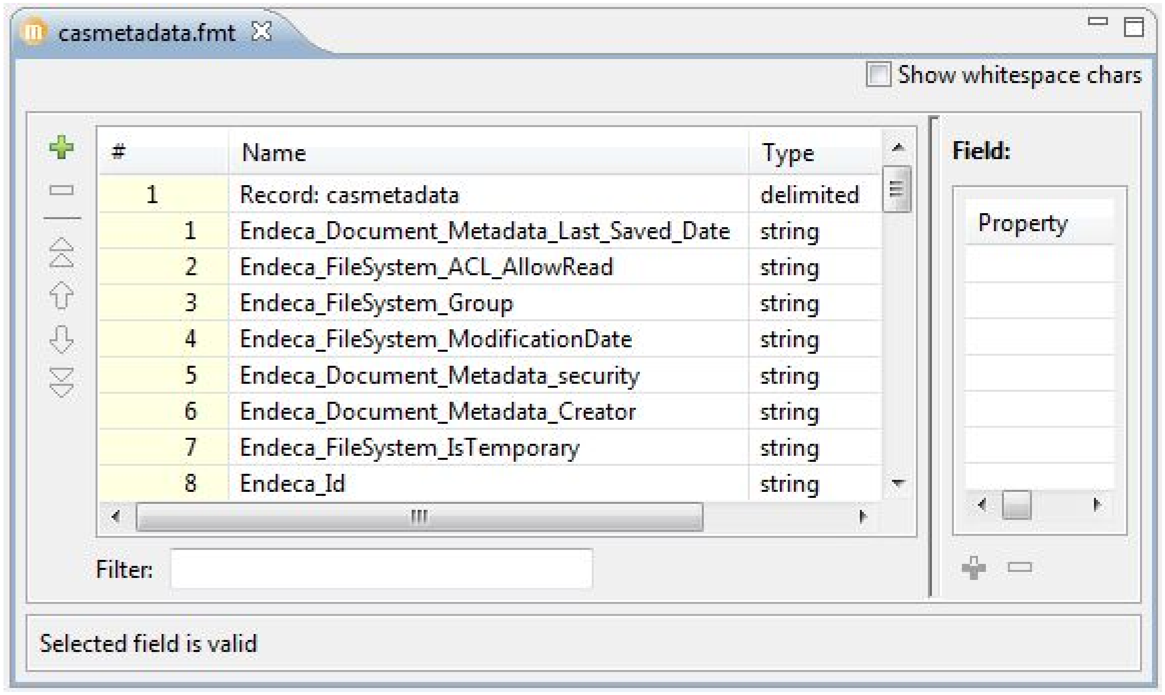


New Metadata from Record Store Wizard

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

Project:	/CAS_RecordStoreReader
File Name:	meta/casmetadata.fmt
CAS Service Host:	localhost
CAS Service Port:	8500
Record Store Instance:	RecCrawl
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 Metadata editor, as shown in this truncated example:



casmetadata.fmt

#	Name	Type
1	Record: casmetadata	delimited
1	Endeca_Document_Metadata_Last_Saved_Date	string
2	Endeca_FileSystem_ACL_AllowRead	string
3	Endeca_FileSystem_Group	string
4	Endeca_FileSystem_ModificationDate	string
5	Endeca_Document_Metadata_security	string
6	Endeca_Document_Metadata_Creator	string
7	Endeca_FileSystem_IsTemporary	string
8	Endeca_Id	string

Selected field is valid

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.

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 Salience Engine from Lexalytics. Depending on your license, the Salience 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 Salience 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 7.2: 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	A list of named entities in the current document. You can specify which types of entities to extract. Supported entity types include: • Company (i.e., businesses) • Person • Place (i.e., geographical locations) • Product • Sports • Title • List (for user-defined entities) The output record includes one column per type. Each column can contain multiple values. 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. This feature can be enabled or disabled.
Themes	A list of themes in the document. All meta-themes are added to the output record in a field you specify. For any theme that is not a meta-theme, if the theme score is higher than a user-specified threshold, then: • 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. • Regardless of whether Sentiment Analysis is enabled or disabled, the theme is added to another (i.e., not meta theme) user-specified field. This feature can be enabled or disabled.
Quotations	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. This feature can be enabled or disabled,
Document Summary	A shortened version of the input content that best represents the whole content in a limited number of words. This feature is always enabled. It cannot be disabled.

Lexalytics information sources

The Lexalytics Support Web site provides two sources of information on the Salience 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 users who are implementing the Text Enrichment feature.

Text Enrichment prerequisites

The Text Enrichment component requires the Salience Engine and a properties file, in addition to the input source text to process.

Salience Engine

The Text Enrichment component requires installation of Salience Engine Version 5.1 on the same machine as Endeca Integrator. Oracle recommends installation of build 5.1.6602 or later. See the following pages for details about installing the Salience Engine:

- Linux installation: *http://dev.lexalytics.com/wiki/pmwiki.php?n=Main.Installation#Linux*
- Windows installation: *http://dev.lexalytics.com/wiki/pmwiki.php?n=Main.Installation#Windows*

When installing the Salience Engine, write down the path to the Salience Engine `data` directory and the Salience Engine license file. You must specify these paths when configuring the Text Enrichment component.

Source Input

The source input is the text to be processed by the Salience Engine. You can use any supported input source, including files, database columns, and CAS record store data.

Input text should not be all upper case. It should be either all lower case or sentence cased normally. If submitted text is entirely upper cases, themes will not be extracted correctly.

Input text must also 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.

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 7.3: Document Sentiment Analysis properties

Document Sentiment property	Meaning
`te.document-sentiment.enabled`	If set to `true` (the default), Sentiment Analysis is enabled for documents and an overall sentiment score is computed for the current document. If set to `false`, Document Sentiment Analysis is disabled.
`te.document-sentiment.field`	Sets the target field name in the output records in which the sentiment score is written. The field name is user-variable, and `DocumentSentiment` is the default.
`te.document-sentiment.use-chains`	If set to `true` (the default), document sentiment scoring will use lexical chains in the computation of the sentiment score. The default is `true`.

Table 7.4: Named Entity processing properties

Named Entity property	Meaning
`te.entity.enabled`	If set to `true` (the default), Named Entity Extraction is enabled. If set to `false`, Named Entity Extraction is disabled.

Named Entity property	Meaning
`te.entity.types`	Sets the types of named entities to extract. Supported types are: • `Company` • `Person` • `Place` • `Product` • `Sports` • `Title` • `List` (must be used for if you have user-defined entities) The default types are `Person`, `Company`, and `Product`. Each configured entity type is written to a target field whose name is made up of the entity type (such as `Person`) prefixed by the `te.entity-sentiment.field` value.
`te.entity.field-prefix`	Sets the prefix name that is used to determine the final field names for the named entities. The field name is user-variable, and `Entities` is the default. For example, if you set this value to **Entities** and you configure `Person` and `Company` entity types to be extracted, then **EntitiesPerson** and **EntitiesCompany** will be the two target field names in the output records. If you have user-defined entities, then all the user-defined entities are put in the **EntitiesList** target field.
`te.entity-sentiment.enabled`	If set to `true` (the default), Sentiment Analysis is enabled for entities and a sentiment score is computed for the entities. If set to `false`, Entity Sentiment Analysis is disabled.
`te.entity-sentiment.cut1.label`	Sets the value for this `cut1` label. This will be the column name for the negative entities to be extracted. The field name is user-variable, and `EntitiesNegative` is the default. Negative entities are entities with a score less than the negative threshold.
`te.entity-sentiment.cut1.value`	Sets the `EntitiesNegative` threshold. The value is user-variable (for example, a value of `-0.1` can be used). Entity-sentiment scores that are less than this value are written to the `EntitiesNegative` record field.

Named Entity property	Meaning
`te.entity-sentiment.cut2.label`	Sets the value for this `cut2` 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 `EntitiesNeutral` is the default.
`te.entity-sentiment.cut2.value`	Sets the `EntitiesPositive` threshold. The value is user-variable (for example, a value of `0.1` can be used). Entity-sentiment scores that are greater than this value are written to the `EntitiesPositive` record field.
`te.entity-sentiment.cut3.label`	Sets the value for this `cut3` 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 `EntitiesPositive` is the default.

Table 7.5: Theme processing properties

Theme property	Meaning
`te.theme.enabled`	If set to `true` (the default), Theme Extraction is enabled. If set to `false`, Theme Extraction is disabled.
`te.theme.field`	Sets the target field name in the output records in which kept theme names are written. The field name is user-variable, and `Themes` is the default. Kept themes are those themes whose score is higher than the `te.theme.score.threshold` setting and have made the `te.theme.keep-max` cut-off list.
`te.theme.score.threshold`	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 `1.0` is the default.
`te.theme.keep-max`	Sets a threshold for keeping the best themes. That is, of those themes that are above the `te.theme.score.threshold` setting, only keep the themes with the best scores. The value is user-variable, and the default is `100`.
`te.theme-sentiment.enabled`	If set to `true` (the default), Sentiment Analysis is enabled for themes and a sentiment score is computed for the themes. If set to `false`, Theme Sentiment Analysis is disabled.

Theme property	Meaning
`te.theme-sentiment.cut1.label`	Sets the value for this `cut1` label. The field name is user-variable, and `ThemesNegative` is the default. Negative themes are themes with a score less than the negative threshold.
`te.theme-sentiment.cut1.value`	Sets the `ThemesNegative` threshold. The value is user-variable (for example, a value of `-0.1` can be used).
`te.theme-sentiment.cut2.label`	Sets the value for the `cut2` label. The field name is user-variable, and `ThemesNeutral` is the default. Neutral themes are themes with a sentiment score between the negative and positive thresholds.
`te.theme-sentiment.cut2.value`	Sets the `ThemesPositive` threshold. The value is user-variable (for example, a value of `0.1` can be used). ThemesPositive are themes with a score greater than the positive threshold.
`te.theme-sentiment.cut3.label`	Sets the value for this `cut3` label. The field name is user-variable, and `ThemesPositive` is the default. Positive themes are themes with a score greater than the positive threshold.
`te.meta-theme.field`	Sets the target field name in the output records in which the meta-themes are written. The field name is user-variable, and `ThemesMeta` is the default. Meta-themes are a list of themes in the document.
`te.meta-theme.frequency.threshold`	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 `1.0` is the default.

Table 7.6: Quotation processing properties

Quotation property	Meaning
`te.quotation.enabled`	If set to `true` (the default), Quoted Context Extraction is enabled. If set to `false`, Quoted Context Extraction is disabled.
`te.quotation.field`	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 `Quotes`.

Quotation property	Meaning
`te.quotation.max-length`	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 7.7: Document Summary properties

Document Summary property	Meaning
`te.summary.field`	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 `Summary`.
`te.summary.length`	Sets the document summary length in sentences. The default length is 3 sentences.

Table 7.8: Basic custom properties

Basic Custom properties	Meaning
`te.salience.userdataDirectory`	Takes an absolute path to a directory that contains a user-created data dictionary.
`te.sentiment.setSentimentDictionary`	Takes an absolute path to a user-created dictionary that will be used as the sentiment dictionary for the Salience Engine (that is, this dictionary overrides the default Salience sentiment dictionary).
`te.sentiment.addSentimentDictionary`	Takes an absolute path to a user-created dictionary that will be used in addition to the current sentiment Analysis dictionary: • If `te.sentiment.setSentimentDictionary` has been used, then the additional dictionary is added to the first user-created sentiment dictionary. • If `te.sentiment.setSentimentDictionary` has not been used, then the additional dictionary is added to the default Salience sentiment dictionary.

Advanced Custom options

In addition to the `te.*` configuration properties listed above, you can set other options provided by the Salience API. You can use custom options from the following classes:

```
Salience.Options.Base.xxx
Salience.Options.Collections.xxx
Salience.Options.Concepts.xxx
Salience.Options.Entities.xxx
Salience.Options.QueryTopics.xxx
Salience.Options.Sentiment.xxx
```

where *xxx* is the name of the specific method you want to configure, such as `Salience.Options.Base.setFailLongSentence`.

Information on these classes is available in the Lexalytics Salience 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 Salience 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.

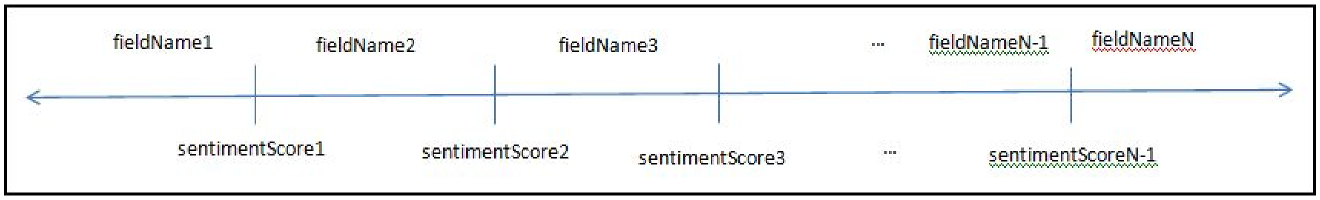
Customizing the theme and/or entity cuts

If you are using Sentiment Analysis for themes and/or entities, you can customize the number of cuts. The "Named Entity property" and "Theme property" tables above assume that you are using three cuts for positive, negative, and neutral scores, but you can use more or fewer cuts.

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
```

Note that the above explanation references only the entity-sentiment configuration options. The theme-sentiment configuration would be almost the same - only the names of the options would be different.

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 above0.1
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

# Summary is always enabled
te.summary.field = Summary
# Document summary length in sentences
te.summary.length = 2

# Set location of my user directory
te.salience.userdataDirectory=/localdisk/djones/lexalytics/salience-5.0/data/user
# Add my sentiment dictionary to the Salience default
te.sentiment.addSentimentDictionary=/localdisk/djones/lexalytics/salience-5.0/custom/custom.hsd
```

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 locations of the Salience License file and the Salience `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 file (see *Text Enrichment properties file*) 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 license file**

  The absolute path to the Salience License file

- **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, the Integrator displays a **...** browse button 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 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 uses a default of `80` (in other words, the text is only submitted to the Salience Engine for enrichment only if at least if 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 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.

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

For details on the Text Enrichment properties file, see *Text Enrichment properties file on page 93*.

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 93*. 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 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 adds a new field to the list of fields. The new field is named **fieldn** where n is the next number in the list of fields.
   (b) Chang 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.

Adding foreign language processing to Text Enrichment

The Salience Engine supports text enrichment in French, German, Portuguese, and Spanish.

To add foreign language processing to Text Enrichment:

1. Download the Text Enrichment foreign language packages from the Oracle Software Delivery Cloud.
2. Unzip the packages and place the foreign language data directories (`fr_data`, `ge_data`, `pt_data`, and `es_data`) in the same directory as the English data directory (`data`) as siblings.
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`.

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

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.

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.

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 105*.
- 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 160*.
- 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 107*.

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 comprised of 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 15* for details.

Be sure to define the output field in the edge from the Test 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 TTW component uses a regular expression (regex) both to search for matches and to render output.

The component includes two fields whose value is a regular expression:

- The **Search Pattern** field specifies a regular expression that is applied to the specified input property to search for matches.
- The **Render Pattern** field specifies a regular expression that is used to write out (render) a value into the target field.

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, or if you apply multiple regular expressions to the same field, 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/6/docs/api/java/util/regex/Pattern.html

Valid constructs you can use include:

- Escaped characters, such **\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}\.\-' ]+$
```

When entered into the **Search Pattern** field, this 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).

Adding Text Tagger Regex to a graph

The Text Tagger Regex component requires a single input, the source data to search for matches.

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** field.
- 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** field tags 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 105*.
- 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 160*.

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 Test Tagger Regex component to the next component to ensure that the data for this field is passed on for further processing.

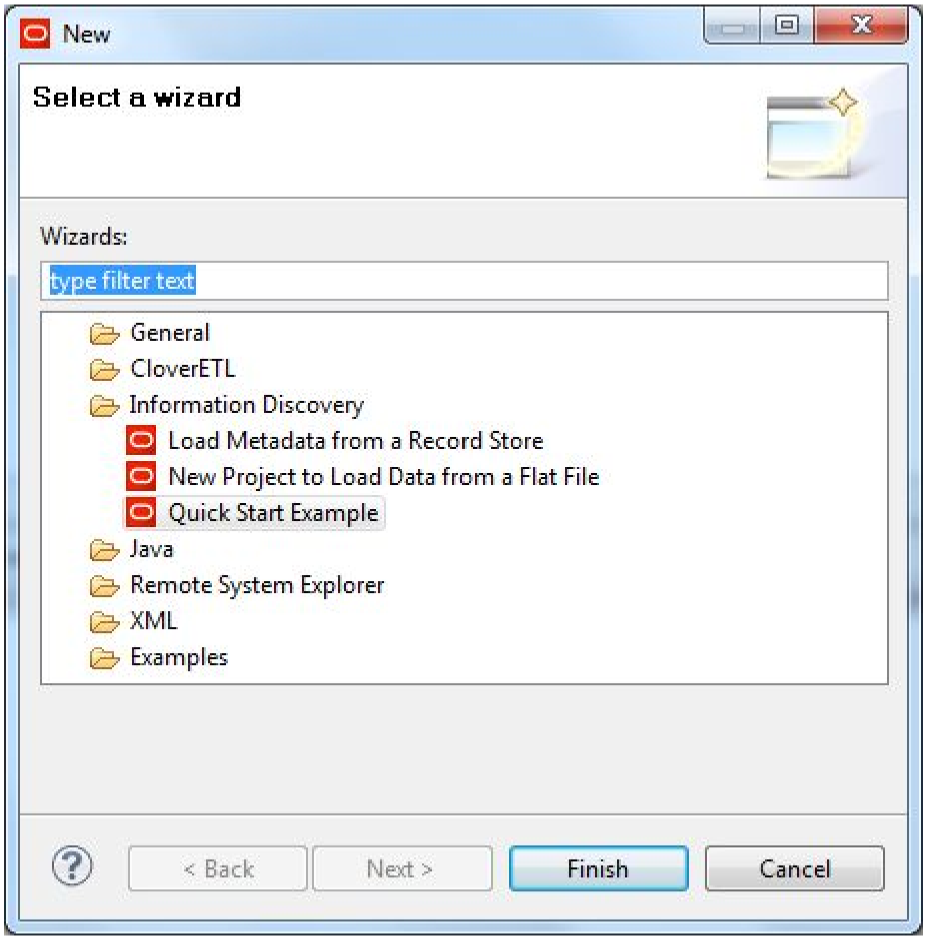
Chapter 8
Wizards

This chapter describes the Information Discovery wizards in the Integrator.

About the Information Discovery wizards

Oracle Endeca Integrator includes three Information Discovery wizards.

To access the wizards from within Integrator, select **File>New>Other**. The Select a Wizard menu is displayed:



The **Load Metadata from a Record Store** wizard generates metadata for a Record Store Reader by querying a CAS Record Store instance. This wizard is documented in the topic *Generating Edge metadata with the Record Store Wizard on page 88*.

Quick Start project

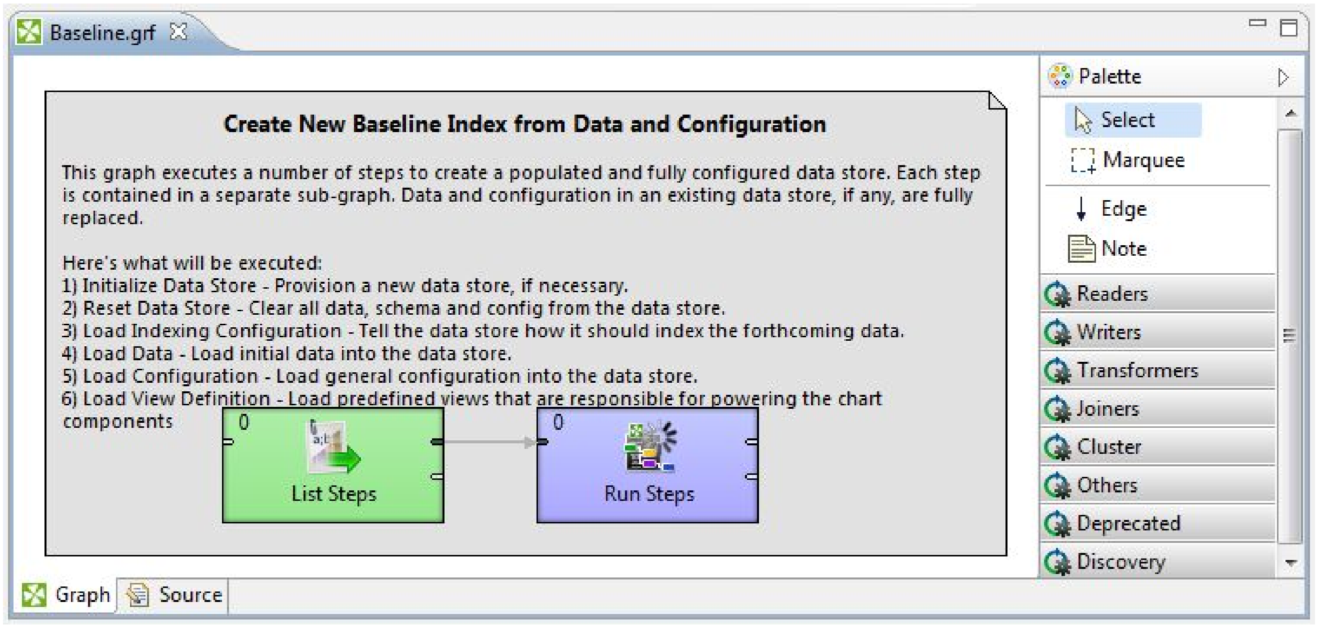
The Quick Start Wizard loads the Quick Start project.

The Integrator comes with a Quick Start project and data. The Quick Start project demonstrates the Oracle Endeca Information Discovery product in action, using sales and product data from a fictitious bicycle manufacturer.

There are three ways you can load the Quick Start project:

- In Integrator, select **File>New>Quick Start Example**.
- In Integrator, first select **File>New>Other**. Then choose **Quick Start Example** from the Select a Wizard menu.
- In the Integrator Welcome screen, click "Open and view the Quick Start project."

All three methods cause the Quick Start Wizard to immediately load the project, which displays the Baseline graph:



Create New Baseline Index from Data and Configuration

This graph executes a number of steps to create a populated and fully configured data store. Each step is contained in a separate sub-graph. Data and configuration in an existing data store, if any, are fully replaced.

Here's what will be executed:
1) Initialize Data Store - Provision a new data store, if necessary.
2) Reset Data Store - Clear all data, schema and config from the data store.
3) Load Indexing Configuration - Tell the data store how it should index the forthcoming data.
4) Load Data - Load initial data into the data store.
5) Load Configuration - Load general configuration into the data store.
6) Load View Definition - Load predefined views that are responsible for powering the chart components

Quick Start project components

The configuration files for the project are in the `config-in` folder. The source data files are in the `data-in` folder.

The `graph` folder has seven graphs that are run in this order:

1. Baseline uses a **RunGraph** component to run the other six graphs, using the `BaselineSteps.csv` file as its input.
2. InitDataStore creates an Endeca data domain (named **quickstart**) by making a `createDataDomain` call to the Endeca Server's Control Web Service. However, if an Endeca data domain named **quickstart** already exists, then it is re-used and no new data domain is created.
3. ResetDataDomain empties the **quickstart** data domain to make sure it has no existing configuration documents or data records.
4. LoadIndexingConfiguration first turns on search indexing by loading the appropriate Endeca standard attributes and then creates the search interfaces from those standard attributes.
5. LoadData first joins the input source files and then uses a **Bulk Add/Replace Records** component to load the records into the **quickstart** Endeca data domain.
6. LoadConfiguration first creates attribute groups and then loads general attribute configuration.
7. LoadViewDefinitions loads the views into the Endeca data domain.

Running the Quick Start project

You run the Quick Start project from the Baseline graph. Before doing so, make sure that the Endeca Server is running.

The graphs are run in the order indicated above. When the graphs finish, you can use the Endeca Server's `list-status` command to verify that the **quickstart** Endeca domain is running.

Load Data from OBI Server wizard

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.

Integrator can load the retrieved data into an Endeca Server data store.

Load Data from OBI Server prerequisites

The Load Data from OBI Server wizard requires a running OBI Server 11g and a running Endeca Server.

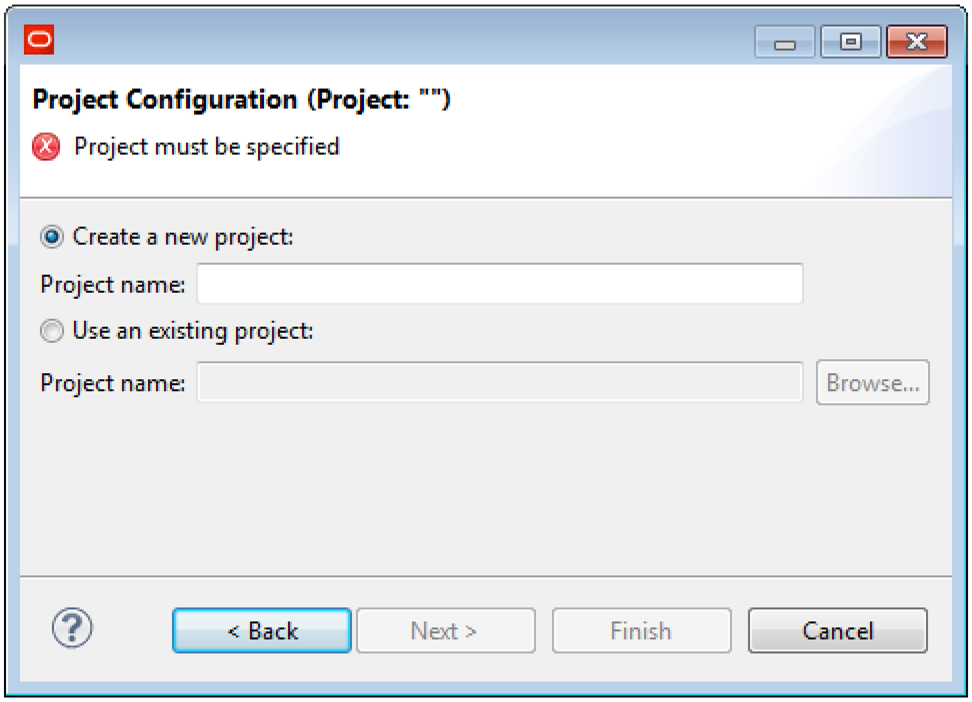
Using the Load Data from OBI Server wizard

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

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

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

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



Project Configuration (Project: "")

Project must be specified

Create a new project:

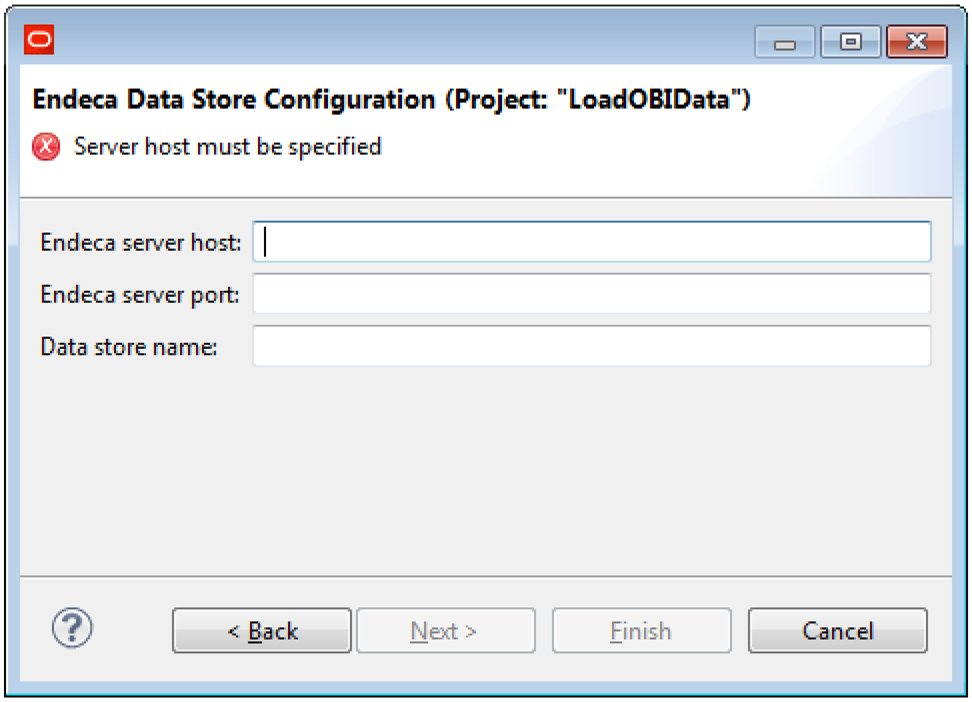
Project name:

Use an existing project:

Project name:

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

Integrator displays the **Endeca Data Store Configuration** dialog.



Endeca Data Store Configuration (Project: "LoadOBIData")

Server host must be specified

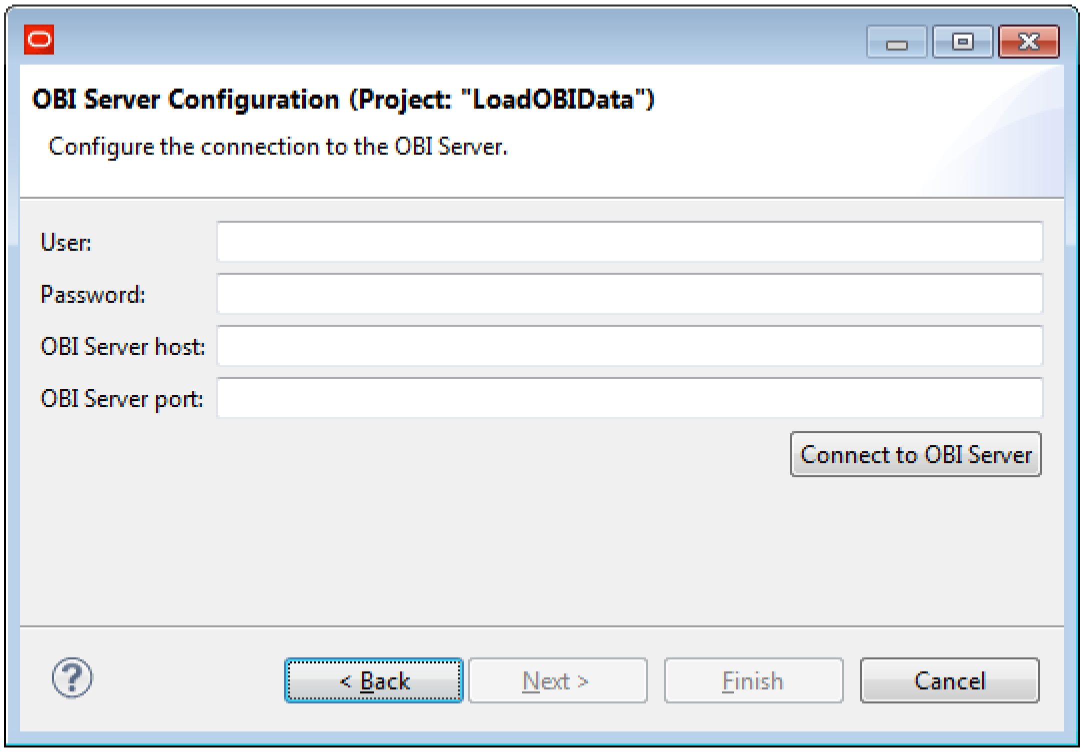
Endeca server host:

Endeca server port:

Data store name:

4. Enter the **Endeca server host** (name or IP address), and **Endeca server port**. Enter the **Data store name**. If the data store you specify does not exist, it will be created when you run the Baseline graph in the project. Click **Next**.

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



OBI Server Configuration (Project: "LoadOBIData")

Configure the connection to the OBI Server.

User:

Password:

OBI Server host:

OBI Server port:

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 **User** and **Password** of the user you want to use to log in to the OBI Server.

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

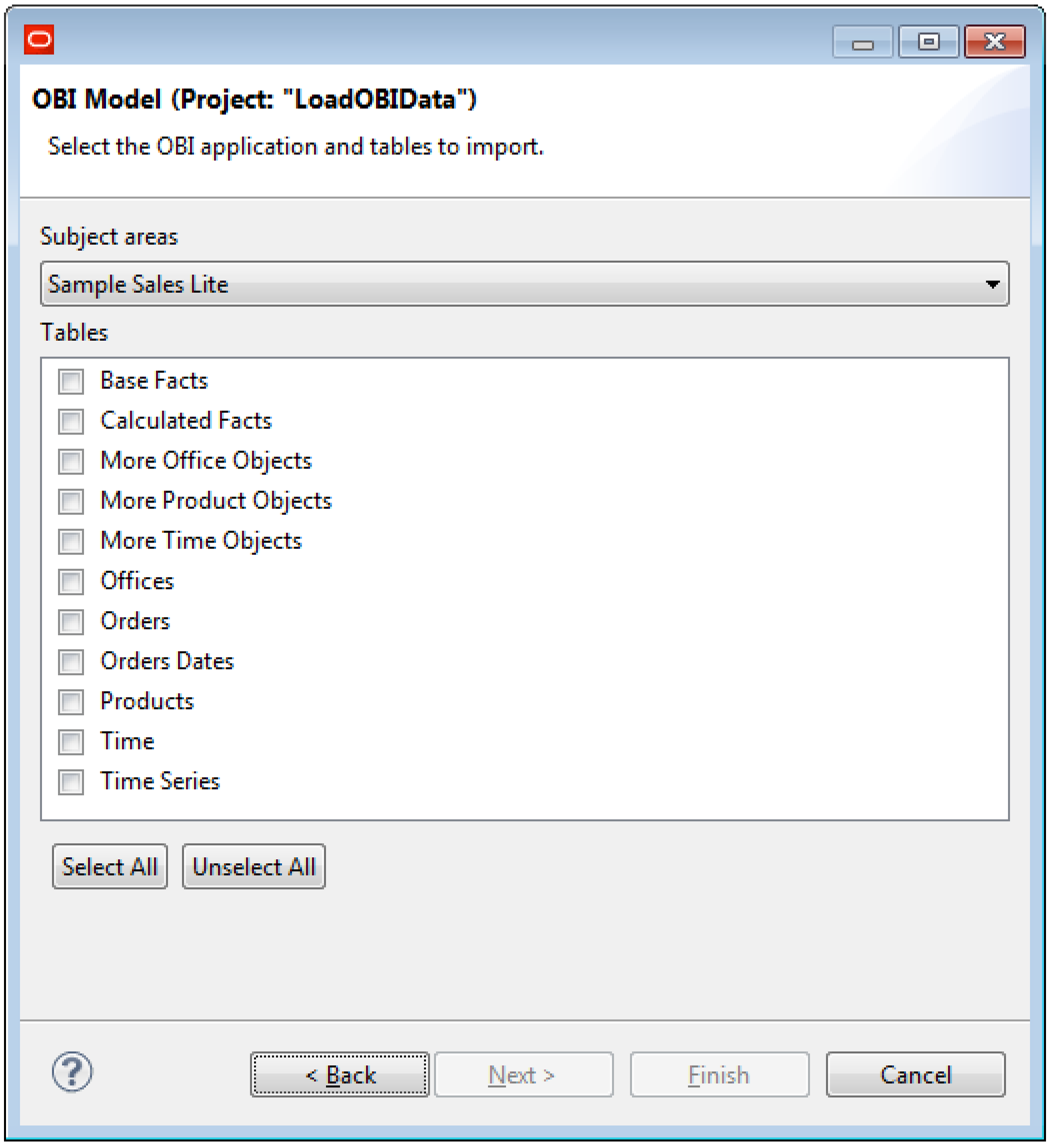
   Integrator attempts to connect to the OBI Server you specified using the connection and authentication data you entered. When you establish a connection, Integrator enables the **Next** button.

   Note:

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

7. Click **Next**.

Integrator displays the **OBI Model** dialog.


OBI Model (Project: "LoadOBIData")

Select the OBI application and tables to import.

Subject areas: Sample Sales Lite

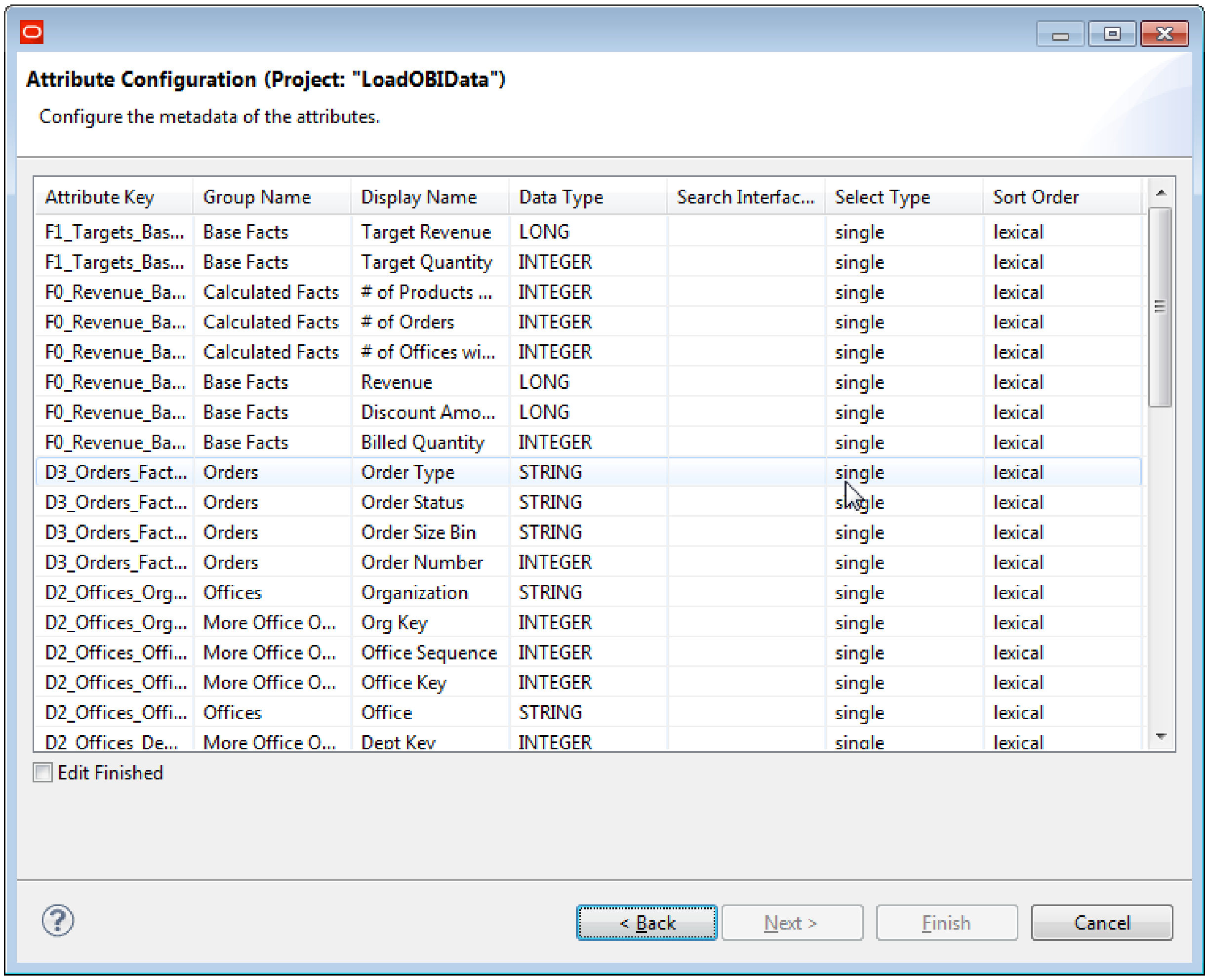
Tables

- [ ] Base Facts
- [ ] Calculated Facts
- [ ] More Office Objects
- [ ] More Product Objects
- [ ] More Time Objects
- [ ] Offices
- [ ] Orders
- [ ] Orders Dates
- [ ] Products
- [ ] Time
- [ ] Time Series

8. Click on the **Subject areas** drop list and choose the OBI subject area from which you want to retrieve data. Check the boxes next to the tables from which you want to retrieve data. To check all boxes, click **Select All**.

9. Click **Next**.

Integrator displays the **Attribute Configuration** dialog populated with guesses about the values for each column in the table.



Attribute Configuration (Project: "LoadOBIData")

Configure the metadata of the attributes.

Attribute Key	Group Name	Display Name	Data Type	Search Interfac...	Select Type	Sort Order
F1_Targets_Bas...	Base Facts	Target Revenue	LONG		single	lexical
F1_Targets_Bas...	Base Facts	Target Quantity	INTEGER		single	lexical
F0_Revenue_Ba...	Calculated Facts	# of Products ...	INTEGER		single	lexical
F0_Revenue_Ba...	Calculated Facts	# of Orders	INTEGER		single	lexical
F0_Revenue_Ba...	Calculated Facts	# of Offices wi...	INTEGER		single	lexical
F0_Revenue_Ba...	Base Facts	Revenue	LONG		single	lexical
F0_Revenue_Ba...	Base Facts	Discount Amo...	LONG		single	lexical
F0_Revenue_Ba...	Base Facts	Billed Quantity	INTEGER		single	lexical
D3_Orders_Fact...	Orders	Order Type	STRING		single	lexical
D3_Orders_Fact...	Orders	Order Status	STRING		single	lexical
D3_Orders_Fact...	Orders	Order Size Bin	STRING		single	lexical
D3_Orders_Fact...	Orders	Order Number	INTEGER		single	lexical
D2_Offices_Org...	Offices	Organization	STRING		single	lexical
D2_Offices_Org...	More Office O...	Org Key	INTEGER		single	lexical
D2_Offices_Offi...	More Office O...	Office Sequence	INTEGER		single	lexical
D2_Offices_Offi...	More Office O...	Office Key	INTEGER		single	lexical
D2_Offices_Offi...	Offices	Office	STRING		single	lexical
D2_Offices_De...	More Office O...	Dept Key	INTEGER		single	lexical

☐ Edit Finished

The following table describes the columns on the **Attribute Configuration** dialog.

Table 8.1: Attribute Configuration Properties

Column Name	Description	Editable
Attribute Key	Name of the standard attribute created from this column. Maps to the mdex-property_Key PDR property.	No

Column Name	Description	Editable
Group Name	Groups to which the attribute belongs Maps to the system-property_GroupMembership PDR property.	Yes Click in the field and modify the text.
Display Name	Name of the attribute displayed in user interfaces. Maps to the mdex-property_DisplayName PDR property.	Yes Click in the field and modify the text.
Data Type	The mdex data type of the standard attribute. Maps to the mdex-property_Type PDR property.	No
Search Interface	The Search Interface assigned to the standard attribute. The Search Interface controls the search behavior of the standard attribute.	Yes if the data type of the attribute is `mdex:string`. Otherwise, no. 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. **Note:** Define the Search Interfaces in the data store configuration before loading data from OBI Server.

Column Name	Description	Editable
Select Type	Defines the multi-select behavior of the standard attribute. Maps to the system-navigation_Select PDR property.	Yes Click in the field and choose the value from the drop list. Available values include: • Single Users can select only one refinement from this attribute. • multi-and Users can select multiple refinements from the attribute. Records are only returned if the value of the assignment matches all selected refinements. • multi-or Users can select multiple refinements from the attribute. Records are returned if they match any of the selected refinements. Defaults to `single`.
Sort Order	Specifies the order in which to display refinements in navigation. Maps to the system-navigation_Sorting PDR property.	Yes Click in the field and choose the value from the drop list. Available values include: • lexical Sorts refinements in alphabetical or numerical order. • record-count Sorts by number of records, in descending order. Defaults to `lexical`.

The following table describes mappings from OBI Server to Integrator to Endeca Server.

Table 8.2: Data Type Mappings

OBI type name	OBI data type	Integrator 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
TINYINT	-6	Integer	mdex:int
VARBINARY	-3	N/A	N/A
VARCHAR	12	String	mdex:string

10. When you are finished modifying the values in the table, check the **Edit Finished** box.
    Integrator enables the **Finish** button.
11. Click **Finish**.
    Integrator runs the wizard processing.

The wizard creates a new project with the following artifacts, listed by directory:

- config-in
  - AttributeGroups.csv
  - AttributeMetadata.csv
  - AttributeSearchability.csv
  - OBI metadata file named `host-port.xml`, where `host` is the value you entered for the **OBI Server host** on the OBI Server configuration dialog and `port` is the value you entered for the **OBI Server port** on the OBI Server configuration dialog.
- conn
  - A connection configuration file named `BIServer.cfg`
  - bijdbc.jar
- data-in
  - A query statement file: `QueryStatement.sql`
- graph
  - Baseline.grf
  - InitDataStore.grf
  - LoadConfigurations.grf
  - LoadData.grf
  - LoadIndexingConfiguration.grf
  - ResetDataStore.grf
- meta
  - DataRecord.fmt
  - DataRecord2.fmt

The Endeca data store configurations specified on the **Endeca Data Store Configuration** dialog are stored in the project's `workspace.prm` file.

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 store. If you want different configurations for the data store, modify the existing graphs (particularly the **LoadIndexingConfiguration.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.

Chapter 9

Component Reference

This chapter is a reference for the Oracle Endeca Information Discovery components available in the Integrator Palette.

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. 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 26*). 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.

Metadata schema

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:

Table 9.1: Add KVP input 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.
`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 *Standard attribute default values on page 26*.
`kvpValue`	The value of the standard attribute to be added to the record.

Schema property	Meaning
`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 *Supported data types on page 25*.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Add KVPs** component.

Table 9.2: 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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`

Name	Description	Valid Values	Example
Spec Attribute	Specifies the primary key (record spec) for the records to be added to the data domain.	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 disabled SSL for the connector. 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 for the ingest operation. Each record size is calculated in bytes. A batch consists of one or more complete records.
	
	See also *Batch size adjustments by connectors on page 160*.
		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, all records are placed into a one batch and sent to the data domain at the end of the ingest operation.
			`1000000`
			
			`0`

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 9.3: 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 9.4: 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	

Add Managed Values component

The **Add MVals** component loads managed attribute values (a taxonomy) into the Endeca data domain.

The **Add Managed Values** component loads a taxonomy (Endeca managed attribute values) into the Endeca data domain using the Data Ingest Web Service (DIWS).

- The component loads only managed values (mvals). It does not load 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 by DIWS with system default values for the Dimension Description Record (DDR) and 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 26*.
- The component also provides the option of creating synonyms for managed values.

Metadata schema

The metadata schema of the **Add 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
```

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.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Add MVals** component.

Table 9.5: Add MVals 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 7770, but it can be changed to another port.	`7770`

Name	Description	Valid Values	Example
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`
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, DIWS automatically creates the managed attribute with system default values.	Category
SSL Enabled	Enables or disabled SSL for the connector. 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 9.6: 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	23
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 9.7: 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	

Add/Update Records component

The **Add Update Records** component adds new records or updates existing records in an Endeca data domain.

The **Add/Update Records** component uses the Data Ingest Web Service (DIWS).

- You can use this component to load
  - data source records
  - Property Description Records (PDRs)
  - Dimension Description Records (DDRs)
- You cannot use this component to load
  - managed attribute values (MVals)
  - the Global Configuration Record (GCR)
  - data domain configuration documents
- A primary-key attribute (also called a record spec) is required for each record to be added or updated.
- 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 26*).
- Updates are batched on the client-side with multiple concurrent connections to the data domain.

Metadata schema

The metadata schema for the **Add/Update Records** component is not fixed. Each Integrator field represents a property on a data domain record.

The metadata type of the Integrator 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 `integer` data type translates to the `mdex:int` data type. Note that you must override this behavior to support Integrator non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`). For details, see *Creating mdexType Custom properties on page 27*.

Use cases

Use the **Add/Update Records** component to load non-bulk data when you want the data to be visible as soon as it is uploaded and when you want to allow queries to continue while you are loading the data.

Use this component in 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 by DIWS with system default values.
- Loading the record schema before an initial load. In this case, you load your PDR schema records (and, optionally, your DDR schema) before loading your data records.
- Full index initial load of records after you have loaded the record schema.
- Incremental updates that add 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 Endeca data domain are automatically created with default system values.
- Incremental updates to existing records, adding key-value pairs. If a standard attribute is configured as multi-assign, a record can have multiple assignments of that attribute. Newly added records are totally

additive. In other words, the key-value pair list of newly added records is merged into the existing record. If attribute values with the same name already exist, then the new values are appended as additional values to that standard attribute.

Note: The **Add KVPs** component provides the same functionality.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Add/Update Records** component.

Table 9.8: Add/Update 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 `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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. 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 to be added to the data domain.	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`

Name	Description	Valid Values	Example
SSL Enabled	Enables or disabled SSL for the connector. 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 for the ingest operation. Each record size is calculated in bytes. A batch consists of one or more complete records.
	
	See also *Batch size adjustments by connectors on page 160*.
		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, all records are placed into a one batch and sent to the data domain at the end of the ingest operation.
			`1000000`
			
			`0`

Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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.	
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 9.9: 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 9.10: 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)
  - the 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. Any existing key/value pair that matches a loaded record is completely replaced by the newly-loaded key/value pair.
- 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 26*).
- The component does now send records in batches. It employs a single streaming connection to the data domain.
- You can run this component in a sub-graph within a top-level graph that starts an outer transaction. For the bulk records operations to run successfully within an outer transaction, the component relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.

Valid characters

A valid character for ingest must be a character according to the *Second Edition of the XML 1.0 Specification*. If the Endeca Server detects an invalid character, it rejects the record and returns the following message to Integrator:

```
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 connector to the graph that includes this connector and use the following code:

```
//#CTL2

// Transforms input record into output record.
function integer transform() {
   string regex = "([^\\u0009\\u000a\\u000d\\u0020-\\uD7FF\\uE000-\\uFFFD]|[\\u0092\\u007F]+)";
   $0.YourDataCleanData = replace($YourDatawithInvalidPattern,regex,"");

   return ALL;
}
```

Compatibility characters are also not valid. The code above removes compatibility characters.

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. You can later update the dictionary using the `updateaspell` administrative operation. For details, see the *Oracle Endeca Server Administrator's Guide*.

Metadata schema

The 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 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 `integer` data type translates to the `mdex:int` data type. Note that you must override this behavior to support Integrator non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`). For details, see *Creating mdexType Custom properties on page 27*.

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 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Bulk Add/Replace Records** component.

Table 9.11: 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 `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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. 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 to be added to the data domain.	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`
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)	

Name	Description	Valid Values	Example
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 disabled SSL for the connector. 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
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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.	

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 9.12: 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 9.13: Port 1 metadata

Port Field Name	Data Type	Description	Example
Fault Message	String	Error message returned by the Endeca Server	

Delete Data component

The **Delete Data** component deletes specified Endeca records.

The **Delete Data** component uses the Data Ingest Web Service (DIWS). This component can delete

- an entire record
- all value assignments from a specific standard attribute on a specific record
- a specific value assignment from a specific standard attribute on a specific record

You cannot use this component to remove managed values from a taxonomy, but you can use it to remove managed attribute assignments from records.

Metadata schema

The metadata schema of the **Delete Data** 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:

```
specKey|specValue|kvpKey|kvpValue
```

where:

- *specKey* is the name of the primary key (record spec) of the record on which the delete operation will be performed.
- *specValue* is the value of the record's primary key.
- *kvpKey* is the name (key) of the Endeca standard attribute to which the assignment belongs. If *kvpValue* is blank, then all assignments of *kvpKey* are deleted. If both *kvpKey* and *kvpValue* are blank, then the entire record is deleted.
- *kvpValue* is the assigned value to be removed.

The following is a simple example of an input file for the **Delete Data** component:

```
specKey|specValue|kvpKey|kvpValue
ProductID|3000|Color|purple
ProductID|4000|Availability|
ProductID|5000||
```

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Delete Data** component.

Table 9.14: Delete 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`

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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`
SSL Enabled	Enables or disabled SSL for the connector. 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 for the ingest operation. Each record size is calculated in bytes. A batch consists of one or more complete records.
	See also *Batch size adjustments by connectors on page 160*.
		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, all records are placed into a one batch and sent to the data domain at the end of the ingest operation.
			`1000000`
			`0`

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 9.15: 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 9.16: 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 requests.

- The component exports the configuration and schema through an output port. You can connect this port to a 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.
- You can run this component in a sub-graph within a top-level graph that starts an outer transaction. To run the export operation successfully within an outer transaction, the component relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.
- The component exports all configurations and schemas but does not export 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 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 Store** 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.

Metadata schema

The metadata schema for the **Export Config** component is not fixed.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Export Config** component.

Table 9.17: 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 `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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`
SSL Enabled	Enables or disabled SSL for the connector. 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 store.

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.
- You can run this component in a sub-graph within a top-level graph that starts an outer transaction. For the import operation to run successfully within an outer transaction, the component relies on an outer transaction ID that you must specify in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.
- 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 records 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 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 Store** 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.

Metadata schema

The metadata schema for the **Import Config** component is not fixed.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Import Config** component.

Table 9.18: 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`
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	Valid ports. The default Endeca Server port is 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`
SSL Enabled	Enables or disabled SSL for the connector. SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	

Record Store Reader

The **Record Store** Reader reads records from a Content Acquisition System (CAS) Record Store instance.

The **Record Store Reader** component reads Endeca records stored in an Endeca Record Store. These records are derived by CAS 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 store via an Information Discovery component.

Metadata schema

The metadata schema 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 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Record Store Reader** component.

Table 9.19: Record Store Reader properties

Name	Description	Valid Values	Example
CAS Service Host	The name or IP address of the machine on which the CAS Services is running.	Valid host names Valid IP addresses	`localhost` (default) `255.255.255.0`
CAS Service Port	The port on which the CAS Service listens.	Valid port numbers	`8500` (default)
CAS Record Store Instance	The name of the unique Record Store instance from which to read records.	Record Store names	`productdata`
CAS Client ID	An arbitrary name that identifies the Integrator client (or a specific instance of the Record Store Reader) to the Record Store. The Record Store uses the value of the CAS Client ID to keep track of which generations if 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.	Alphanumeric characters	`Integrator` (default)

Name	Description	Valid Values	Example
CAS Read Type	Specifies the type of read operation to use to read records from the Record Store.	• `Full Extract` Retrieves all records from the most recently committed generation in the Record Store (also known as "baseline records"). This is the default option. • `Incremental` Retrieves only records that were added or changed since the last committed generation (also known as "delta records").	
SSL Enabled	Enables or disabled SSL for the connector. SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	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 delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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)

Name	Description	Valid Values	Example
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 want for a connection to be established with the Endeca CAS Service.	Integers greater than or equal to 0 A value of `0` is treated as an infinite timeout.	`300000` (default; five minutes)

Reset Data Store component

The **Reset Data Store** component reset the Endeca data store to the empty state.

The Reset Data Store component removes all of the records (including the schema and view definitions) from the Endeca data store, re-provisions the Endeca data store, and updates the spelling dictionary.

- The **Reset Data Store** component uses operations from the Data Ingest Web Service. These operations delete all records (including schema records) and configurations, and provision the Endeca data store. Then the component uses an administrative command to update the spelling dictionary (`admin?op=updateaspell`).
- You can run this component in its own graph, or as part of an outer transaction. within a graph that starts an outer transaction. For the reset operations to run successfully, the component relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.

Use cases

Use this component when you want to clear the data store. For example:

- During development, you may want to reset the data store to remove incorrect data store configurations.
- In production, you may want to refresh the data store through a baseline update.

Metadata schema

The metadata schema for the **Reset Data Store** component is not fixed.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Reset Data Store** component.

Table 9.20: Reset Data Store properties

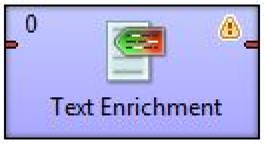
Name	Description	Valid Values	Example

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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. The data domain should be running when the graph containing the connector is run.	Valid data domain names	`quickstart`
SSL Enabled	Enables or disabled SSL for the connector. 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 Salience 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 Salience Engine also provides the ability to assess the sentiment of input text. This enhanced capability requires a different license than the basic text enrichment capability.

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 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Text Enrichment** component.

Table 9.21: 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 ${PROJECT} or a similar global variable to specify the path.	`${PROJECT}/TextEnrichments.properties`
Input field	Name of the source field in the input source record that you want to enrich (extract entities and assess sentiment)	field names	`survey_responses`

Salience license file	Absolute path to the Lexalytics Salience license file	Valid file path	`C:/Program Files (x86)/Lexalytics/license.v5`
			
			`/usr/endeca/salience/licencse.v5`

Salience data path	Absolute path to the Lexalytics `data` directory	Valid file path	`C:/Program Files (x86)/Lexalytics/data`
			
			`/usr/endeca/salience/data`

Name	Description	Valid Values	Example
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 delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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 Regex component

The **Text Tagger Regex** component uses a regular expression (regex) to match text in a specified text field on the incoming records and then tags the output records with a computed value.

You must specify both a search pattern to match the input text and a render pattern to generate the output text.

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 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Text Tagger Regex** component.

Table 9.22: 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)	
Search Pattern	The regular expression to apply to the field specified in the Source Field Name property to find matches.	Valid regular expression	

Name	Description	Valid Values	Example
Render Pattern	The regular expression to generate the output to add to the field specified in the Target Field Name property.	Valid regular expression	
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Text Tagger Regex** component.

Table 9.23: 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)	
Case Sensitive Matches	Specifies whether a match requires that the case of characters in the string match the value in the tags-rule field.	• `True` Case must match the input string for a valid match. • `False` A match is valid regardless of whether case matches.	
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiters is different from the delimiter that separates property fields on the source record. See also *Multi-assign delimiter*.	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`

Transaction RunGraph component

The **Transaction RunGraph** component runs multiple Integrator graphs within a single 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 out 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 subgraphs 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 stores.

- 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 ore 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 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 `integer` data type translates to the `mdex:int` data type. Note that you must override this behavior to support Integrator non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`). For details, see *Creating mdexType Custom properties on page 27*.

Configuration properties

Note: For details about visual properties for all connectors, see *Visual properties of components on page 158*. For details about configuration properties common to all connectors, see *Common configuration properties of components on page 159*.

The following table describes the configuration properties available for the **Transaction RunGraph** component.

Table 9.24: 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 7770, but it can be changed to another port.	`7770`
Data Store Name	Name of the data domain to which records will be added. 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 behavior when a transaction fails.	• `Rollback` Default. When a transaction fails, roll the data domain back to the state before the transaction started and commit the transaction. • `Commit` When a transaction fails, commit any successful changes to the data domain before the fail occurred and commit the transaction. • 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 Web Services client component, or you can execute the web service operation directly.	

Name	Description	Valid Values	Example
SSL Enabled	Enables or disabled SSL for the connector. SSL should only be enabled when the Endeca Server to which you are connecting has SSL enabled.	Checked (True) Unchecked (False)	

Visual properties of components

All components share that 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 Designer Guide*.

Table 9.25: Visual properties of Information Discovery components

Visual property	Purpose	Valid values
Component name	Displays the component name when the component is placed on a graph.	You can change the default name to a more descriptive one.
Location	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.
Size	Describes the dimensions (size) of the component icon within the graph.	Do not edit this field.
Click here to edit component description	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 Designer Guide*.

Table 9.26: Common properties of Information Discovery components

Common property	Purpose	Valid values
ID	Identifies the component among all of the other components within the same component type.	Do not edit this field.
Component type	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.
Specification	Describes what this component can do.	Do not edit this field.
Phase	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.
Enabled	Enables or disables the component for parsing data.	• `enabled` (the default) means the component can parse data. • `disabled` means the component does not parse data. • `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.
Pass Through Input Port	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.

Common property	Purpose	Valid values
Pass Through Output Port	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.
Allocation	If the graph is executed by a Cluster of Integrator Servers, this attribute must be specified in the graph.	For information on this property, see the *Oracle Endeca Information Discovery Integrator Designer Guide*.

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/Update Records
- Add KVPs
- Delete Data
- 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.

Chapter 10

Troubleshooting Problems

This section provides information and solutions to problems you may encounter when working with components and graphs.

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.

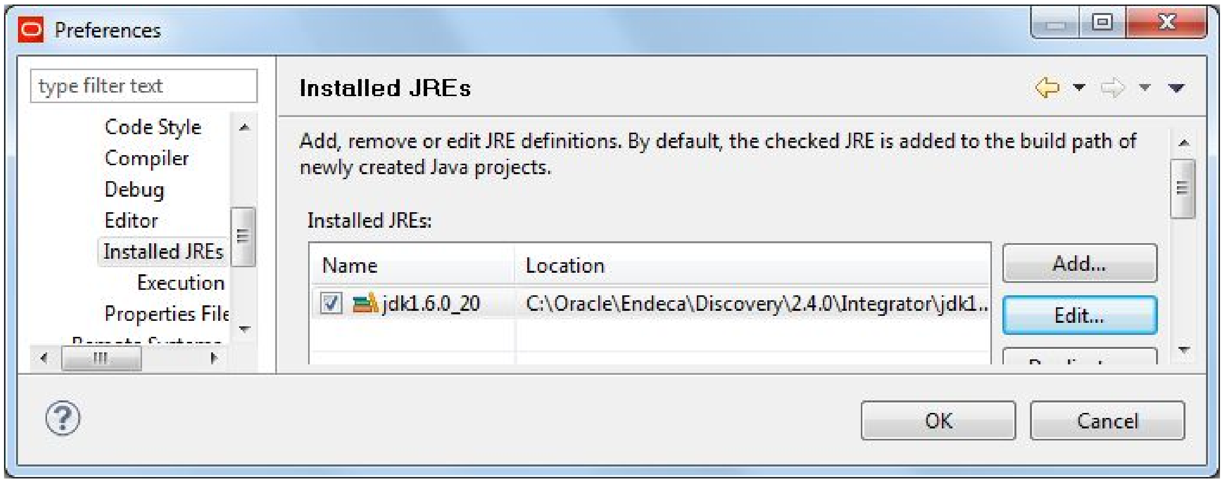
To modify memory allocations:

1. In the menu bar, choose **Window>Preferences**

   Integrator displays the **Preferences** dialog.

2. 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.



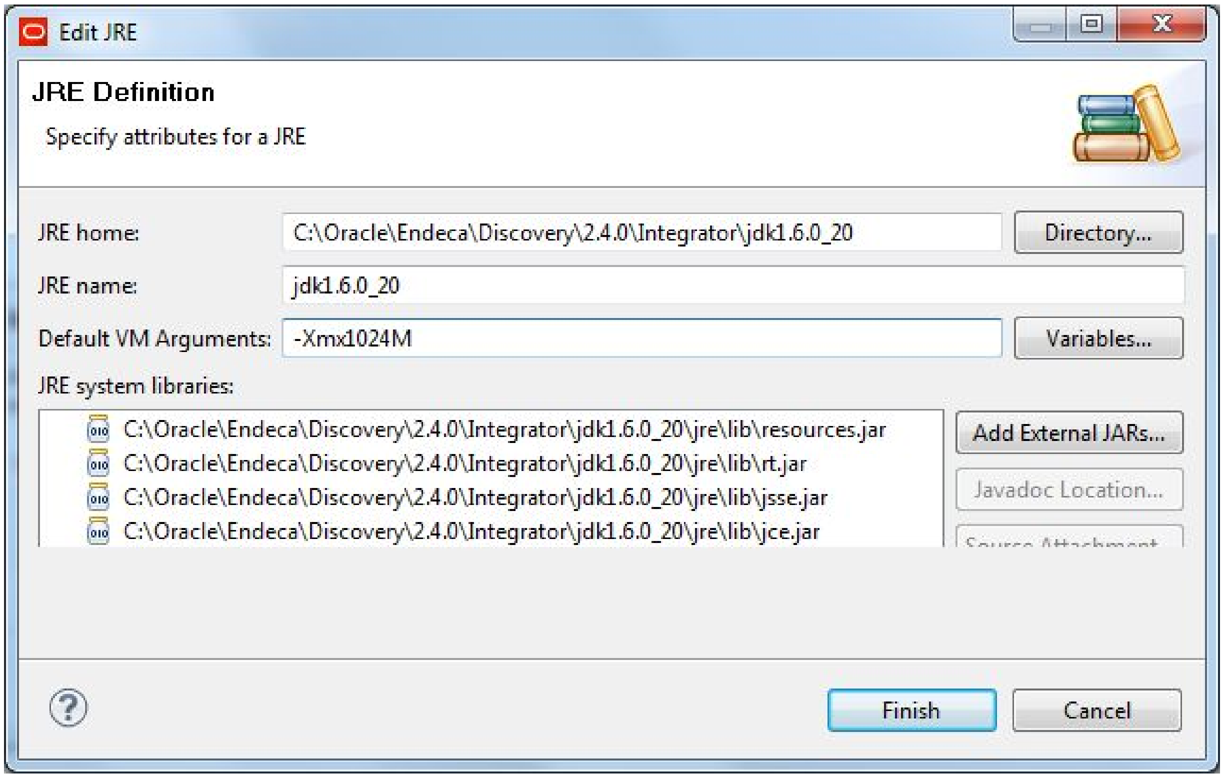
Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

Name	Location
☑ jdk1.6.0_20	C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1..

3. In the **Installed JREs** menu, click on the checked JRE and then click **Edit**.
   Integrator displays the **Edit JRE** page.
4. 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:



JRE home: C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20

JRE name: jdk1.6.0_20

Default VM Arguments: -Xmx1024M

JRE system libraries:

C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20\jre\lib\resources.jar
C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20\jre\lib\rt.jar
C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20\jre\lib\jsse.jar
C:\Oracle\Endeca\Discovery\2.4.0\Integrator\jdk1.6.0_20\jre\lib\jce.jar

5. 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 in 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 Oracle Endeca Integrator 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 semi-colon 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.

Index