Endeca® Latitude

Data Integrator Guide

Version 2.1.0 • June 2011

Contents

Copyright and disclaimer

Trademarks

Preface

Endeca® Latitude applications guide people to better decisions by combining the ease of search with the analytic power of business intelligence. Users get self-service access to the data they need without needing to specify in advance the queries or views they need. At the same time, the user experience is data driven, continuously revealing the salient relationships in the underlying data for them to explore.

The heart of Endeca's technology is the MDEX Engine.™ The MDEX Engine is a hybrid between an analytical database and a search engine that makes possible a new kind of Agile BI. It provides guided exploration, search, and analysis on any kind of information: structured or unstructured, inside the firm or from external sources.

Endeca Latitude includes data integration and content enrichment tools to load both structured and unstructured data. It also includes Latitude Studio, a set of tools to configure user experience features including search, analytics, and visualizations. This enables IT to partner with the business to gather requirements and rapidly iterate a solution.

About this guide

This guide describes the Endeca Latitude Integrator, which loads records and taxonomies into the MDEX Engine.

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.

Who should use this guide

This guide is intended for developers who are responsible for loading source data into the MDEX Engine.

Conventions used in this guide

This guide uses the following typographical conventions:

Code examples, inline references to code elements, file names, and user input are set in `monospace` font. In the case of long lines of code, or when inline monospace text occurs at the end of a line, the following symbol is used to show that the content continues on to the next line: ¬

When copying and pasting such examples, ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

Contacting Endeca Customer Support

The Endeca Support Center provides registered users with important information regarding Endeca software, implementation questions, product and solution help, training and professional services consultation as well as overall news and updates from Endeca.

You can contact Endeca Standard Customer Support through the Support section of the Endeca Developer Network (EDeN) at *http://eden.endeca.com*.

Chapter 1

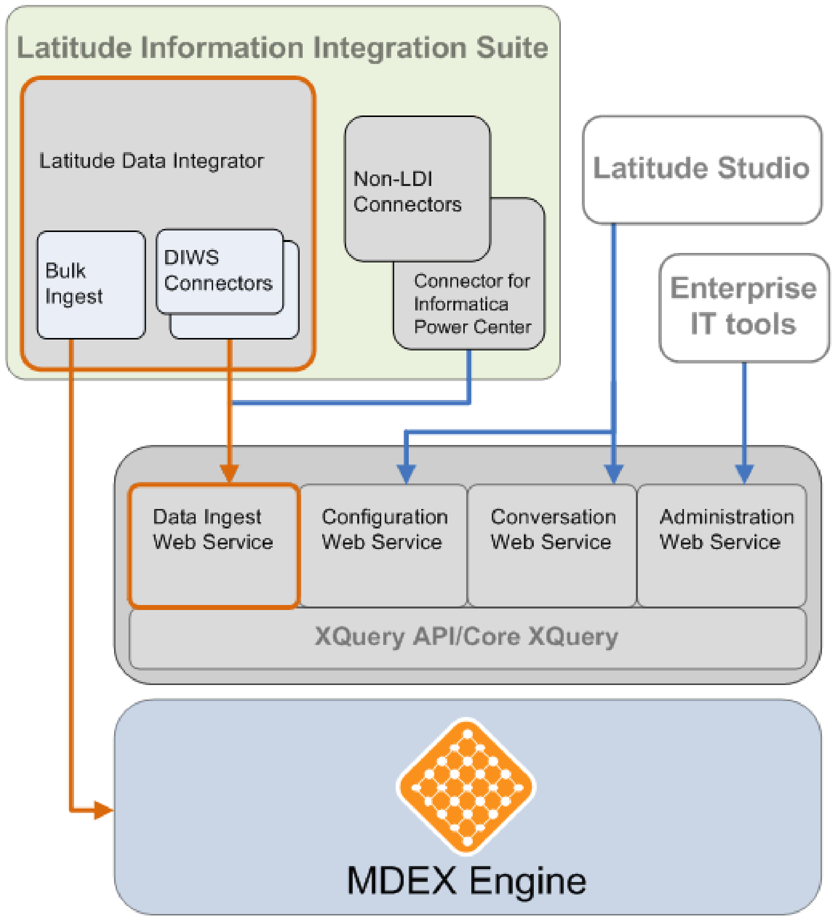
Introduction

The chapter provides an overview of the Latitude Data Integrator and the Latitude connectors.

Overview of Latitude Data Integrator

From a high level, Latitude Data Integrator (LDI) consists of the LDI Designer, the Endeca Latitude connectors, and (optionally) an LDI Server.

This diagram shows how the LDI fits into the larger picture of the Latitude architecture:

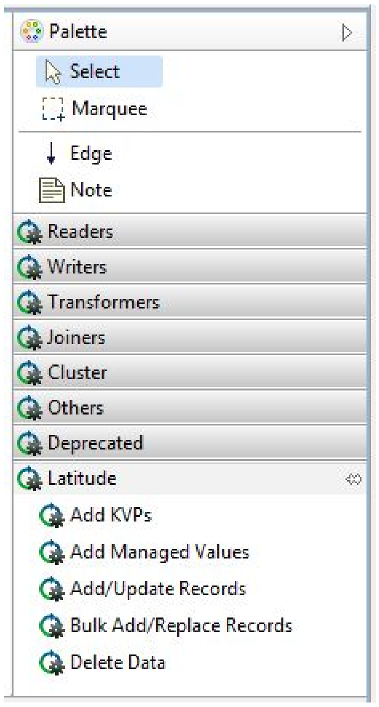


Latitude Data Integrator is a high-performance data integration platform that extracts source records from a variety of source types (from flat files to databases) and sends that data to the MDEX Engine via the Data Ingest Web Service or the Bulk Load Interface, as shown in the diagram.

Latitude connectors

The Endeca Latitude connectors are used to load data into the MDEX Engine.

The Endeca-developed Latitude connectors have been specifically engineered to work with the MDEX Engine's Data Ingest Web Service and Bulk Load Interface. The connectors are incorporated into the Designer Palette so that you can easily drag them into your graphs:



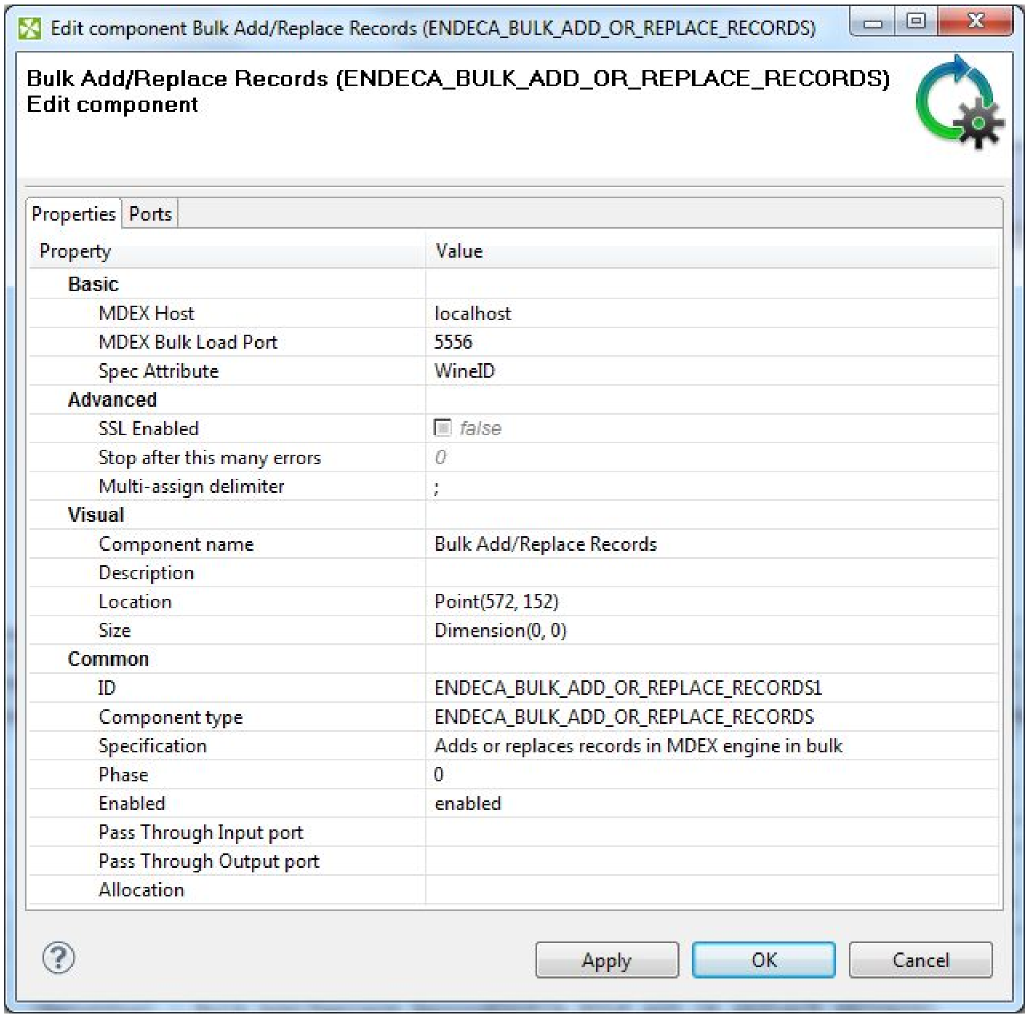
The **Bulk Add/Replace Records** and **Add/Update Records** connectors both add new records to a running MDEX Engine. The records can be added to an empty MDEX Engine (this operation is called a full index initial load) or to one that already has records. The **Add/Update Records** connector can also be used to load the application's record schema, by loading the PDRs (Property Description Records) and DDRs (Dimension Description Records). One convenient feature of both connectors that if an Endeca standard attribute to be added does not exist, it is created automatically.

The **Add KVPs** connector can update existing records by adding new key-value pair (KVP) assignments to those records. The connector can also create new records for the key-value pairs, as well as creating new standard attributes for KVP assignments for non-existent standard attributes.

The **Add Managed Values** connector adds a taxonomy (managed values) to a running MDEX Engine. If the managed values belong to a managed attribute that is currently not in the MDEX Engine, the managed attribute will be created automatically.

While the above four connectors add data to the MDEX Engine, the **Delete Data** connector can remove KVP assignments from records in the MDEX Engine or delete entire records.

All five connectors support SSL connections to an SSL-enabled MDEX Engine. They are configured via the Designer's Writer Edit components, as shown in this example for the **Bulk Add/Replace Records** connector:


Edit component Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)
Properties
Ports
Apply
OK
Cancel

Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Bulk Load Port	5556
Spec Attribute	WineID
Advanced	
SSL Enabled	☐ *false*
Stop after this many errors	*0*
Multi-assign delimiter	;
Visual	
Component name	Bulk Add/Replace Records
Description	
Location	Point(572, 152)
Size	Dimension(0, 0)
Common	
ID	ENDECA_BULK_ADD_OR_REPLACE_RECORDS1
Component type	ENDECA_BULK_ADD_OR_REPLACE_RECORDS
Specification	Adds or replaces records in MDEX engine in bulk
Phase	0
Enabled	enabled
Pass Through Input port	
Pass Through Output port	
Allocation	

Chapter 9 ("Latitude Connector Reference") provides a comprehensive reference of the connectors and their configuration properties. Other chapters in this guide describe how to build LDI Designer graphs with the connectors.

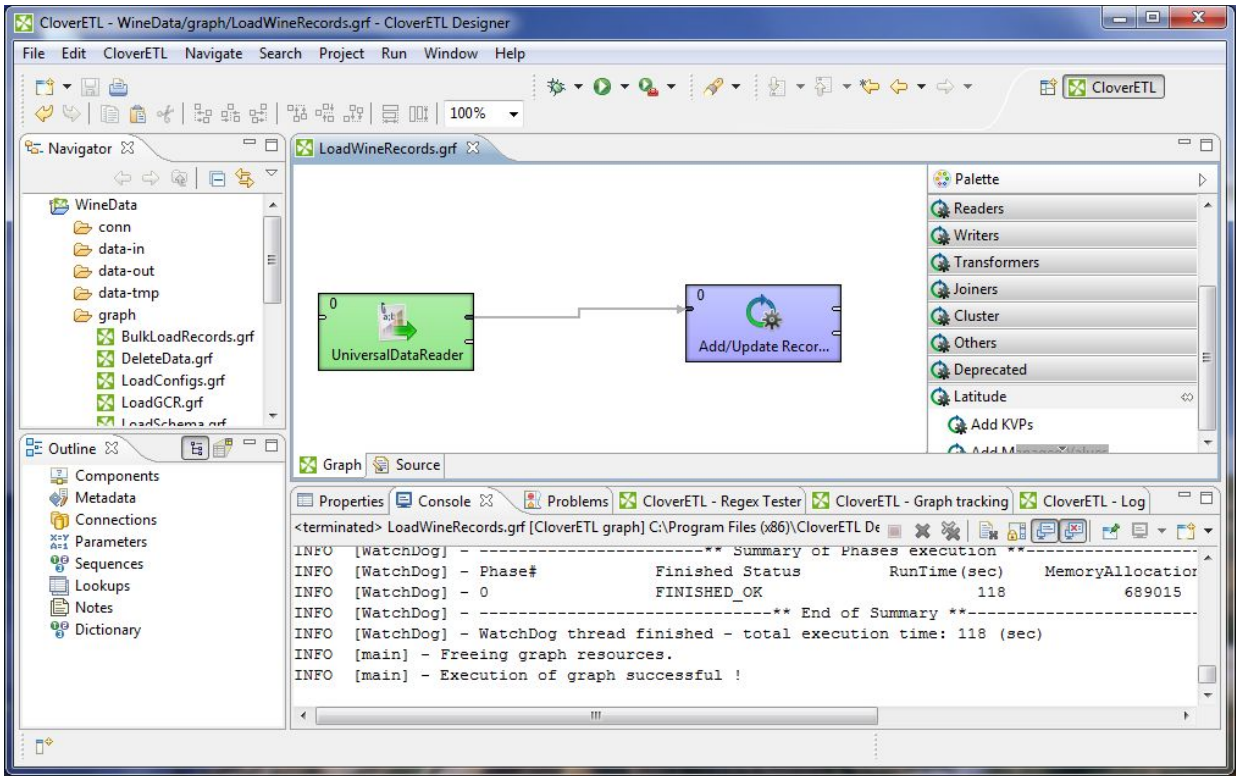
Latitude Data Integrator Designer

The Designer is where you create the graphs for loading your data.

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 Endeca components to write (send) the data to the MDEX Engine. More complex graphs will use additional components, such as Transformer and Joiner components.

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



```
<terminated> LoadWineRecords.grf [CloverETL graph] C:\Program Files (x86)\CloverETL De
INFO  [WatchDog] - ----------------------** Summary of Phases execution **-----------------
INFO  [WatchDog] - Phase#           Finished Status         RunTime(sec)    MemoryAllocatior
INFO  [WatchDog] - 0                FINISHED_OK                      118            689015
INFO  [WatchDog] - ------------------------------** End of Summary **------------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 118 (sec)
INFO  [main] - Freeing graph resources.
INFO  [main] - Execution of graph successful !
```

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

For more information on the Designer user interface, see the online help.

Latitude Data Integrator Server

The Server provides a runtime environment for the graphs.

The Latitude Data Integrator Server is not required in order to load data into the MDEX Engine. In other words, the Data Integrator Designer clients can run independently, and do not require the Server in order to do their 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 enterprise application server, such as Apache Tomcat or IBM Websphere.

Because the Server is not a mandatory component for loading data into the MDEX Engine, it is not documented in this guide. For information on the setup and use of the Latitude Data Integrator Server, see the *CloverETL Server Reference Manual*.

Supported data types

This topic lists the Designer native data types that are supported in the MDEX Engine.

The table also shows how the Designer supported data types are mapped to MDEX Engine data types during an ingest operation. You will see the data types when you create the Metadata definition for the Edge component connector.

Designer Data Types in Metadata	Maps to MDEX 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 MDEX Engine will use that type when creating the standard attribute's PDR. For details, see the "Creating mdexType Custom properties" topic in Chapter 10 of this guide.

Default values for new attributes

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

During any data ingest operation, if a non-existent Endeca standard attribute is specified for a record, the specified attribute is automatically created by the MDEX Engine. 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.

PDR property	Default setting
`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 MDEX Engine. 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)

Additional documentation

Additional Designer and Server documentation is available online and on the Web.

Documentation online

You can access online documentation from within the Designer by clicking **Help Contents** from the **Help** menu. Doing so brings up three documents:

- *CloverETL Designer User's Guide* – a comprehensive user's guide for the Designer.
- *Workbench User Guide* – describes the Eclipse Workbench development environment.
- *Java Development User Guide* – describes how to use the Java development tools.

Documentation on the Web

An extensive set of Designer and Server documentation is available at this URL:
http://www.cloveretl.com/resources

From that Web site, you can access the *Quickstart Guide*, the Designer User's Guide (which is the same as the online version listed above), and the *Server Reference Manual*.

Other resources, such as a users forum, can also be accessed from this Web page.

Chapter 2

Full Index Loads of Records

This chapter describes how to create a Data Integrator project and a graph that will perform a full index load of records into the MDEX Engine.

Overview of full index load operations

This chapter will walk you through the various tasks involved in creating a graph that can load source records into the MDEX Engine.

The task that this chapter covers is how to perform a full index load of your source records into the MDEX Engine. As the source records are ingested, they are converted into Endeca records and are indexed by the MDEX Engine.

This operation assumes that you have already loaded your attribute schema (PDRs and DDRs) into the MDEX Engine with the **Add/Update Records** connector. The load-schema procedure is documented in Chapter 4 ("Loading the Attribute Schema") of this guide. Also, the procedure assumes that the MDEX Engine is otherwise empty of user source data.

Note: Keep in mind that you can ingest your data without first loading your attribute schema. However, if you do so, you will not have control over the default values for the standard attributes that are created. For this reason, it is recommended that you first load your attribute schema data before loading your user source data.

Although a graph can contain many transformation components, the graph in this chapter is a simple one that has only two components:

- The **UniversalDataReader** component, which will read in records from your source data file.
- The **Bulk Add/Replace Records** connector. This Endeca Latitude connector will send the records to the Bulk Load Interface of the MDEX Engine. For details on this connector and some use cases for it, see Chapter 9 ("Latitude Connector Reference") in this guide.

The source data is assumed to be in a flat file, with each source record having multiple columns that are delimited by the pipe character. The format of the source data is explained in a following topic. You can, of course, use other source formats, including reading from a database. These other input formats may require other types of readers, such as the **DBInputTable** reader.

Creating a project

You must create an LDI project in which you will build your graph.

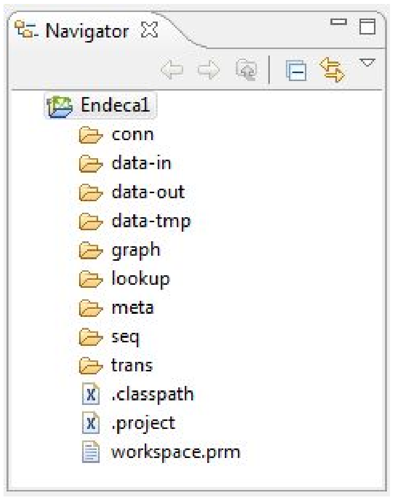
If you already have a project, you can re-use it for your graph. In other words, a project can have multiple graphs configured in it.

To create a new LDI project:

1. From the File menu, select **New** > **CloverETL Project**.
2. In the **New CloverETL project** dialog, enter a name for the project in the **Project name** field.

   You can leave the **Use default location** box checked.
3. Click **Next** and then click **Finish**.

Your new project is displayed in the Navigator pane, as in this example that shows the Endeca1 project in an expanded format:



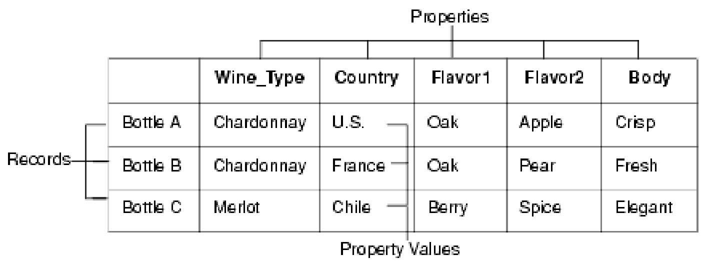
Note that the Outline pane is empty, as is the Graph Editor.

Source data format

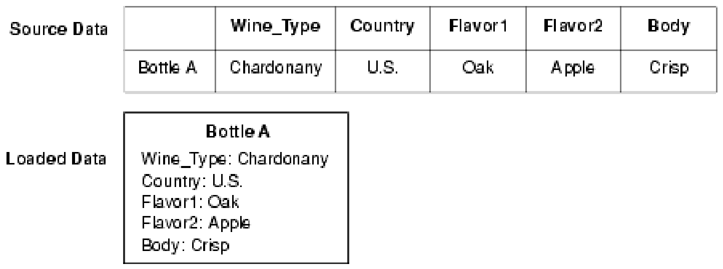
You can load source data from a variety of formats.

Your Endeca applications will most often read data directly from one or more database systems, or from database extracts. Input components load records in a variety of formats including delimited, JDBC, and XML. Each input component has its own set of configuration properties. One of the most commonly used type of input component loads data stored in delimited format.

The format used as an example in this chapter is a two-dimensional format similar to the tables found in database management systems. Database tables are organized into rows of records, with columns that represent the source properties and property values for each record. (This type of format is often called a rectangular data format.) The illustration below shows a simple example of source data in a two-dimensional format.



	Wine_Type	Country	Flavor1	Flavor2	Body
Bottle A	Chardonnay	U.S.	Oak	Apple	Crisp
Bottle B	Chardonnay	France	Oak	Pear	Fresh
Bottle C	Merlot	Chile	Berry	Spice	Elegant



	Wine_Type	Country	Flavor1	Flavor2	Body
Bottle A	Chardonany	U.S.	Oak	Apple	Crisp

Bottle A
Wine_Type: Chardonany
Country: U.S.
Flavor1: Oak
Flavor2: Apple
Body: Crisp

You specify the location and format of the source data to be loaded in the LDI reader component in the graph. The reader component passes the data to the Endeca connector, which is configured to connect to either the Data Ingest Web Service (DIWS) or the Bulk Load Interface, both of which reside on the MDEX Engine. The records are then loaded into the MDEX Engine in batches of a pre-configured size. During the ingest operation, each source row is transformed into an Endeca record with a key-value pair for each non-null source column. The MDEX Engine then indexes the records for use during search queries.

Primary key attribute

You will be using one of the Endeca standard attributes as the primary-key attribute for the records. (The primary-key property is also known as the record spec property.) The primary-key property must be a unique, single-assign property. For more information on primary keys, see the *Data Ingest API Guide*.

Although the MDEX property type of the primary key can be any supported type, in our sample data set, we will use the WineID property, which is an integer. The name of the primary-key attribute will be specified in the Metadata definition for the Edge component.

Use of hyphens in input property names

Although the MDEX Engine will accept attribute names with hyphens (because hyphens are valid NCName characters), the Designer will not accept source property names with hyphens as metadata. Therefore, if you have a source property name such as "Wine-Type", make sure you remove the hyphen from the name.

Using multi-assign data

Your source data may have multi-assign properties, that is, a property that has more than one value. For example, instead of having two properties (say, Flavor1 and Flavor2) in which each property has only one value, you can instead have one property (say, Flavor) with multiple values, as in this simple example:

```
WineID|WineType|Country|Flavor|Body
123|Chardonnay|USA|Oak;Apple|Crisp
456|Chardonnay|France|Oak;Pear|Fresh
789|Merlot|Chile|Berry;Spice|Elegant
```

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. Therefore, the Flavor property for record 123 has values of "Oak" and "Apple".

When configuring the Writer component, you can then specify (in the Writer Edit Component dialog) that the semi-colon is to be used as the delimiter for multi-assign properties.

Keep in mind that an Endeca property 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 property is enabled for multi-assign by default.

Adding the source data to the project

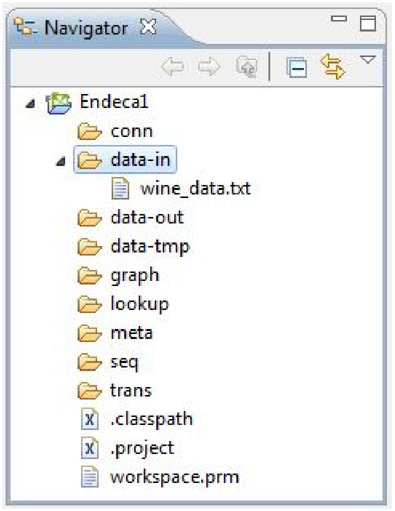
The easiest way to add your source data is to copy it into the project's `data-in` directory.

This procedure assumes that you are copying a text file named `wine_data.txt`, which is a flat file containing delimited records. The pipe character (|) is the delimiter. You can use other input file formats, such as a CSV (comma-separated value) file.

To add the source data file to your Data Integrator project:

1. Locate the project's `data-in` directory.
   To find its location, right-click on **data-in** (in the Navigator pane) and select **Properties**.
2. Copy the source file into the `data-in` directory.
   You use the Designer GUI to paste the file into the **data-in** folder in the Navigation pane.
3. In the Navigator pane, right-click on **data-in** and select **Refresh**.

After refreshing the Navigator pane, it should look like this example:



As the example shows, the `wine_data.txt` is now available to the project's graphs.

Creating a graph

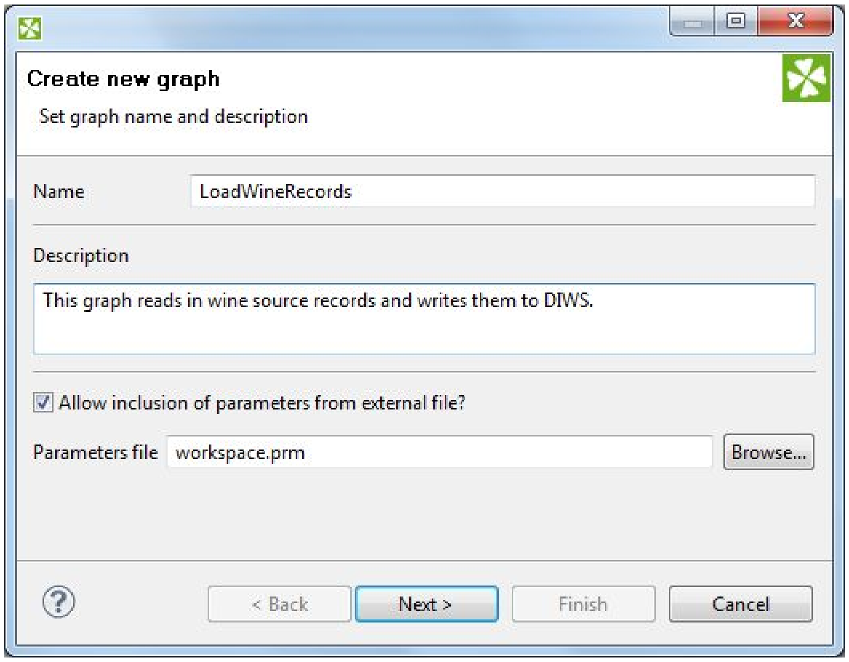
This task describes how to create an empty graph.

An empty graph is one that does not have any transformation components. The only prerequisite for this task is that you must have created a Data Integrator Designer project. A project can have multiple graphs, but only one graph will be created for the project in this chapter.

To create an empty graph:

1. In the Navigator pane, right-click the **graph** folder.
2. Select **New** > **ETL Graph**.
   The **Create new graph** dialog is displayed.
3. In the **Create new graph** dialog:
   a) Type in the name of the graph, such as **LoadWineRecords**.
   b) Optionally, type in a description.
   c) You can leave the **Allow inclusion of parameters from external file** box checked.
   d) Click **Next** when you finish.

   After this step, the **Create new graph** dialog should look like this example:



Create new graph
Set graph name and description

Name: LoadWineRecords

Description: This graph reads in wine source records and writes them to DIWS.

☑ Allow inclusion of parameters from external file?

Parameters file: workspace.prm

4. In the **Output** dialog, click **Finish**.

As a result of creating the graph, the following changes appear in the perspective:

- The Graph window will have an empty graph, with the graph name as the name of the window.
- The Properties window (below the Graph window) will show the graph properties.
- The Outline pane will show a list of items (most of them are empty).
- The Palette pane will list the available graph components, including the Endeca components.

The next task is to add reader and writer components to the graph.

Adding Reader and Writer components

You need to add components to the empty graph in order for it to process the input source data and output it to the MDEX Engine.

The two components to be added are a Reader and a Writer:

- A *Reader* is a graph component that reads in source data. In our example, the **UniversalDataReader** component is used because it can read in data from flat files.
- A *Writer* is a graph component that is responsible for outputting data from the Transformation. The **Bulk Add/Replace Records** connector is used because we are doing a bulk load of your data.

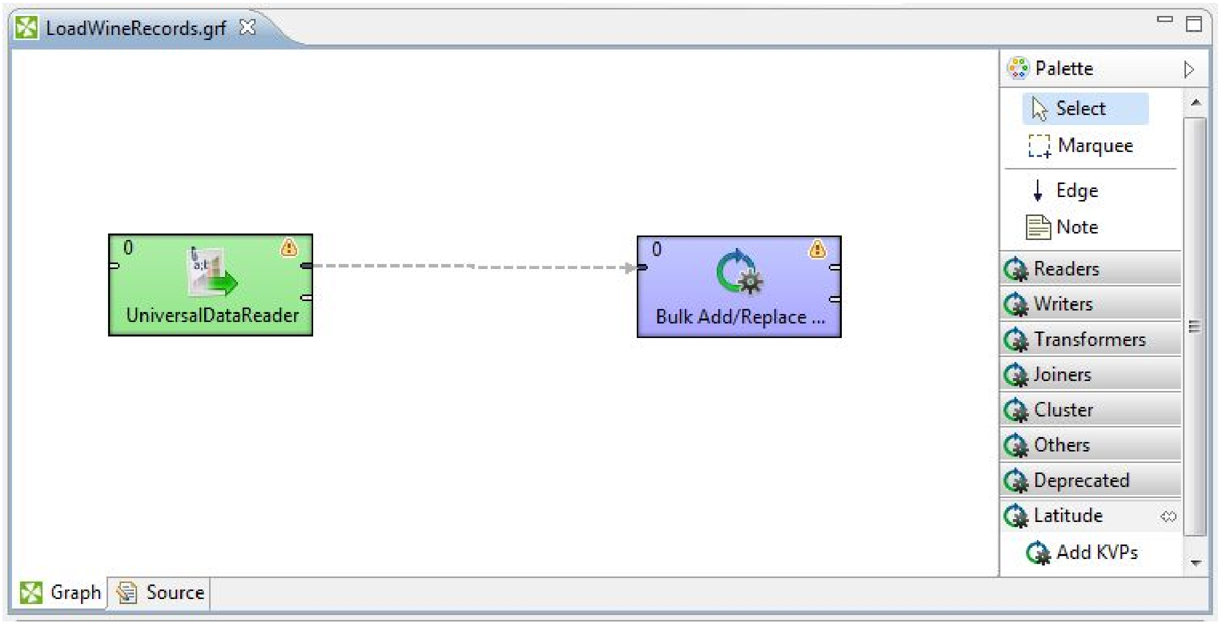
In addition, an *Edge* component will be added to connect the Reader and Writer components. The configurations for all three components are covered in this chapter.

To add components to the graph:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Palette pane, open the **Latitude** section and drag the **Bulk Add/Replace Records** component into the Graph Editor.
3. In the Palette pane, click **Edge** and use it to connect the two components.

   After connecting the components, you can get out of Edge selection mode by hitting **Escape** on your keyboard or clicking on **Select** in the Palette.
4. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the two connected components should look like this:



The next tasks are to configure these components for the source data and for a connection to the MDEX Engine.

Configuring the components

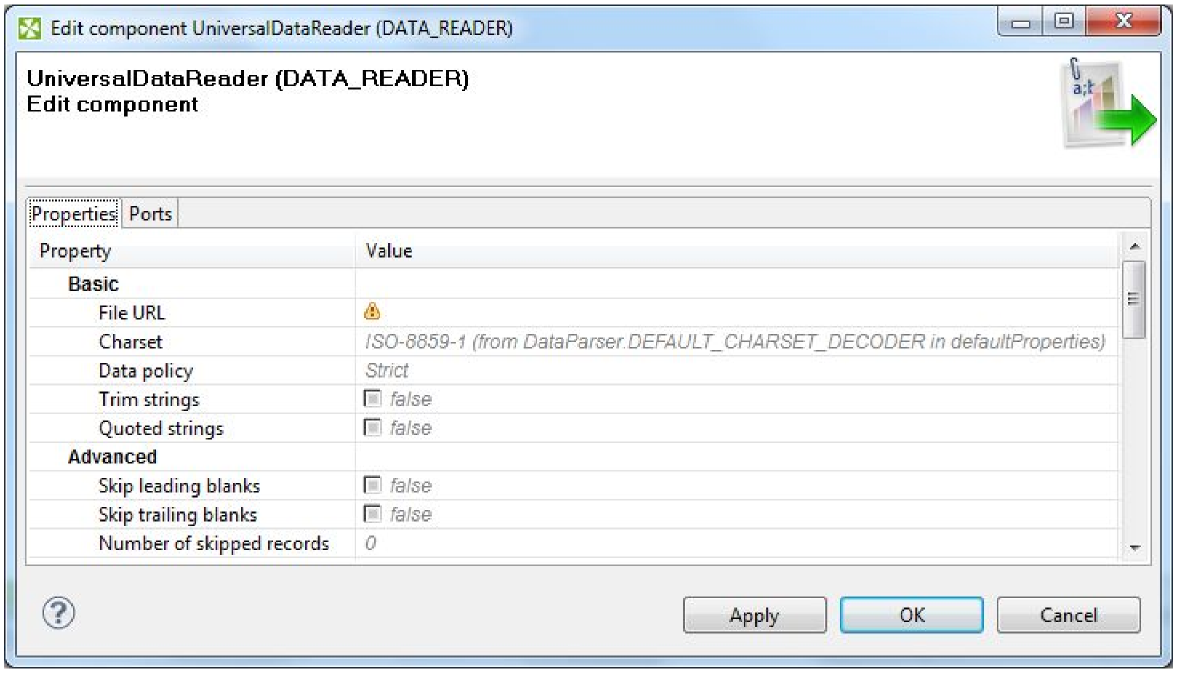
This section describes how to configure the **UniversalDataReader**, **Bulk Add/Replace Records**, and **Edge** components.

Configuring the Reader component

This task describes how to configure the Reader component to read in the source data.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the data source file to the project's **data-in** folder.

The Reader Edit Component dialog is where you configure the Reader as to how it should handle the source data:



Edit component UniversalDataReader (DATA_READER)
UniversalDataReader (DATA_READER)
Edit component
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
File URL	⚠
Charset	ISO-8859-1 (from DataParser.DEFAULT_CHARSET_DECODER in defaultProperties)
Data policy	Strict
Trim strings	☐ false
Quoted strings	☐ false
Advanced	
Skip leading blanks	☐ false
Skip trailing blanks	☐ false
Number of skipped records	0

To configure the Reader component:

1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
2. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the source data file and click **OK**.
3. Check the **Quoted strings** box so that its value changes to `true`.

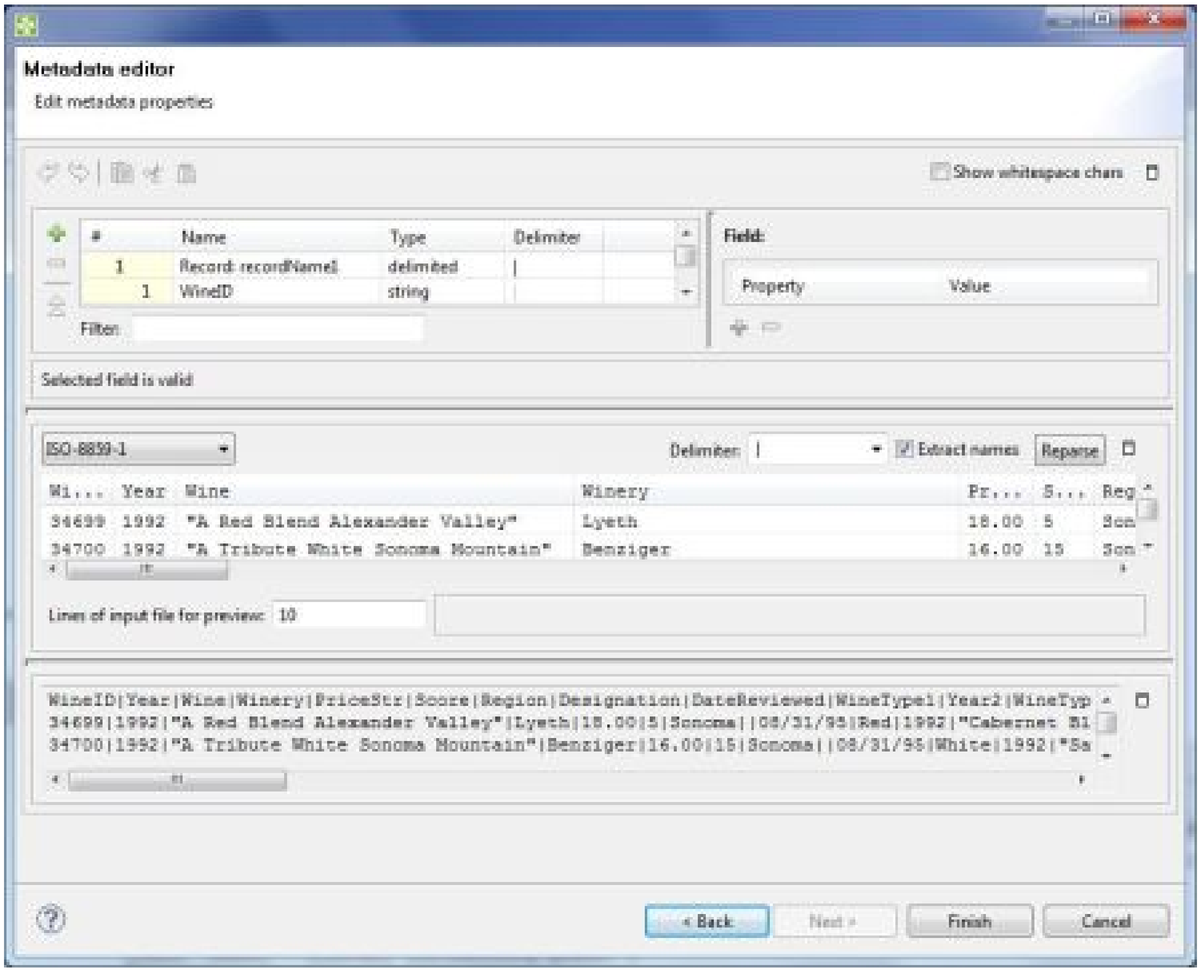
   If set to `true`, delimiter characters inside the quoted strings are ignored (not treated as delimiters) and the quotes are removed.
4. Leave the **Number of skipped records** field as 0.
5. Click **OK** to apply your configuration changes to the Reader component.
6. Save the graph.

Configuring metadata for the Edge

The **Edge** component has to be associated with a Metadata definition so it knows what fields of data are being passed from the Reader component to the Writer component.

By setting the Metadata definition, you are actually defining the properties that will be tagged on the records.

Most of the metadata configuration will be done in the Metadata editor:



#	Name	Type	Delimiter
1	Record: recordName1	delimited	|
1	WineID	string	

Property	Value

Selected field is valid

ISO-8859-1 Delimiter: | ☑ Extract names Reparse

```
Wi...  Year  Wine                                Winery     Pr...  S...  Reg
34699  1992  "A Red Blend Alexander Valley"      Lyeth      18.00  5     Son
34700  1992  "A Tribute White Sonoma Mountain"   Benziger   16.00  15    Son
```

Lines of input file for preview: 10

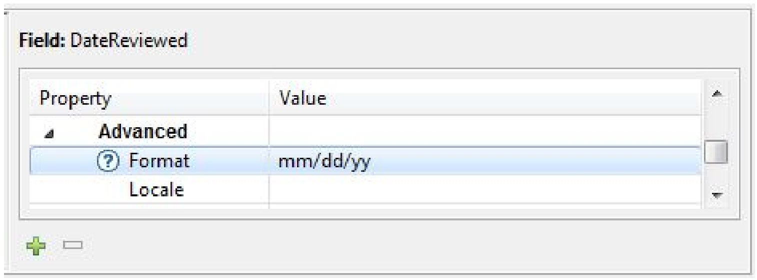
```
WineID|Year|Wine|Winery|PriceStr|Score|Region|Designation|DateReviewed|WineType1|Year2|WineTyp
34699|1992|"A Red Blend Alexander Valley"|Lyeth|18.00|5|Sonoma||08/31/95|Red|1992|"Cabernet Bl
34700|1992|"A Tribute White Sonoma Mountain"|Benziger|16.00|15|Sonoma||08/31/95|White|1992|"Sa
```

You will be using this editor in step 6 of this procedure.

To configure the Metadata definition for the **Edge** component:

1. In the **Outline** pane, right-click on **Metadata**.
2. Select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
3. In the Flat File dialog, click the **Browse** button, which brings up the **URL Dialog**.
4. In the URL Dialog, double-click the **data-in** folder, select the source data file, and click **OK**.
   As a result, the Flat File dialog is populated with source data from the data file.
5. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**.
   The Metadata Editor is displayed, as in the example above.
6. In the middle pane of the Metadata editor:
   a) Check the **Extract names** box.
   b) Click **Reparse**.
   c) Click **Yes** in the Warning message.
7. 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 name that is appropriate for your data, such as **Wine** for the wine data set. In this example, **Record:Wine** will be the resulting Name value.
b) Make sure that the **Type** field of the primary-key property is set to the correct type. In our wine data, we change the WineID property to type **long**.
c) If the source data has date properties, you should their type to **date** (the type may be set to **string** by the Designer). In our example, change the DateReviewed property to a **date** type and then enter **mm/dd/yy** in the **Format** field in the Field pane on the right.



Field: DateReviewed

Property	Value
Advanced	
Format	mm/dd/yy
Locale	

d) Verify that the other properties have their property type set correctly.
e) Verify that all properties have the correct delimiter character set (which is the pipe character for the wine source data). The final property should have a new-line as the delimiter (\n on Linux and \r\n on Windows).
f) When you have input all your changes, click **Finish**.

8. In the Graph Editor window, right-click on the **Edge** between the **UniversalDataReader** component and the **Bulk Add/Replace Records** connector. Choose **Select Metadata** and the metadata you have just configured, for example, **Wine (id:Metadata0)**.
9. Save the graph.

The Metadata definition for the **Edge** component is now set.

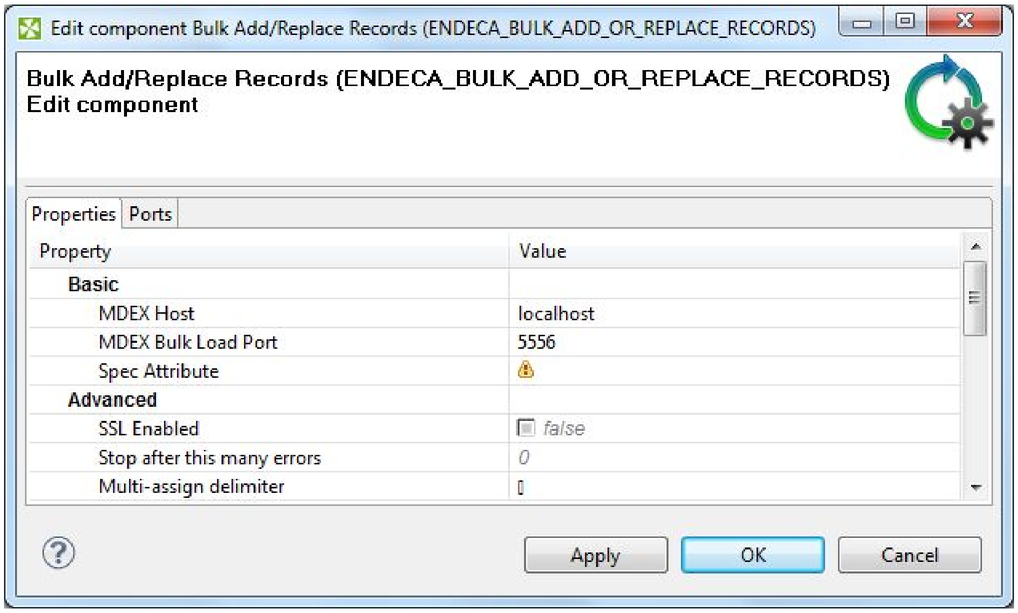
Configuring the Bulk Add/Replace Records connector

This topic describes how to configure the **Bulk Add/Replace Records** connector for the bulk loading of records.

This procedure assumes that you have created a graph and added the **Bulk Add/Replace Records** connector.

Note: When using the **Bulk Add/Replace Records** connector, it is a good idea to use the Dgraph `--bulk_load_port` flag when starting the MDEX Engine.

The Writer Edit Component dialog is where you configure the **Bulk Add/Replace Records** connector:


Edit component Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)

Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Bulk Load Port	5556
Spec Attribute	⚠
Advanced	
SSL Enabled	☐ *false*
Stop after this many errors	*0*
Multi-assign delimiter	▯

To configure the **Bulk Add/Replace Records** connector:

1. In the Graph window, double-click the **Bulk Add/Replace Records** component.
   The Writer Edit Component dialog is displayed.
2. In the Writer Edit Component dialog, enter these settings:
   - **MDEX Host**: Enter the host name of the machine on which the MDEX Engine is running.
   - **MDEX Bulk Load Port**: Enter the bulk load port on which the MDEX Engine is listening. Note that the MDEX Engine opens a bulk load port on 5556 by default. However, you can change the port number with the Dgraph `--bulk_load_port` flag.
   - **Spec Attribute**: Enter the name of the standard attribute that is the primary key (record spec) for the records.
   - **SSL Enabled**: Toggle this field to `true` only if the MDEX Engine is SSL enabled.
   - **Stop after this many errors**: Optionally, you can specify the maximum number of ingest errors that can occur before the load operation is terminated.
   - **Multi-assign delimiter**: Optionally, you can specify the character that separates multi-assign values in an input property. Keep in mind that this delimiter is different from the delimiter that separates properties.
3. When you have input all your changes, click **OK**.
4. Save the graph.

Running the graph to load records

After creating the graph and configuring the components, you can run the graph to send the records to the MDEX Engine.

You can run a graph in one of three ways:

- You can select **Run** > **Run As** > **CloverETL graph** from the main menu.
- You can right-click in the Graph editor and select **Run As** > **CloverETL graph** from the context menu.

- You can click the green circle with white triangle icon in the Tool bar:

To run the graph:

1. Make sure that you have an MDEX Engine running on the host and port that are configured in the **Bulk Add/Replace Records** connector.
2. Run the graph using of the methods listed above.

As the graph runs, the process of the graph execution is listed in the Console Tab. The execution is completed successfully when you see final output similar to this example:

```
INFO  [WatchDog] - ----------------------** Final tracking Log for phase
[0] **---------------------
INFO  [WatchDog] - Time: 03/06/11 17:32:45
INFO  [WatchDog] - Node                   ID          Port      #Records
      #KB aRec/s   aKB/s
INFO  [WatchDog] - -------------------------------------------------------
--------------------------
INFO  [WatchDog] - UniversalDataReader    DATA_READER0
             FINISHED_OK
INFO  [WatchDog] -  %cpu:0.31                         Out:0        57076
    44707    570     447
INFO  [WatchDog] - Bulk Add/Replace RecordENDECA_BULK_ADD_OR_REPLACE_RECORDS1
          FINISHED_OK
INFO  [WatchDog] -  %cpu:..                            In:0        57076
    44707    570     447
INFO  [WatchDog] - ---------------------------------** End of Log **------
--------------------------
INFO  [WatchDog] - Execution of phase [0] successfully finished - elapsed
time(sec): 100
INFO  [WatchDog] - -----------------------** Summary of Phases execution
**---------------------
INFO  [WatchDog] - Phase#            Finished Status         RunTime(sec)
   MemoryAllocation(KB)
INFO  [WatchDog] - 0                 FINISHED_OK                      100
            43793
INFO  [WatchDog] - ------------------------------** End of Summary **-----
----------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 100
(sec)
INFO  [main] - Freeing graph resources.
INFO  [main] - Execution of graph successful !
```

As the example shows, the Final Tracking Log lists the number of records that were read in by the **UniversalDataReader** component and the number of records that were sent to the MDEX Engine by the **Bulk Add/Replace Records** connector.

Chapter 3

Incremental Updates

This chapter describes how to create a Data Integrator graph that will perform an incremental update of records into the MDEX Engine.

Overview of incremental updates

You can incrementally update the data set in the MDEX Engine, including adding new records.

Using the **Add/Update Records** connector, you can perform these types of incremental updates:

- Add a brand-new record to the data set in the MDEX Engine.
- Update an existing record by adding key-value pairs.

Note that the **Add/Update Records** connector cannot load managed attribute values, nor can it delete records or record data.

Format of the incremental source input file

Because the assumption is that you are adding (or updating) records that are similar in format to what is already in the MDEX Engine, the format of the input will be very similar to the format of the input file for the full index load. For more information, see the topic titled "Source data format" in Chapter 2 ("Full Index Loads of Records") of this guide.

How updates are applied

The records to be added are considered totally additive. That is, if a record with the same primary key already exists in the MDEX Engine, the key-value pairs list of the added record will be merged into the existing record.

If an Endeca attribute with the same name already exists (but has a different assigned value), then the added key-value pair will be an additional value for the same property (multi-assign). 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.

Keep in mind, however, that you cannot add a second value to a single-assign attribute. (That is, an attribute whose PDR has the `mdex-property_IsSingleAssign` set to `true`.) In the **Color** example, if **Color** were a single-assign attribute and the record already had one **Color** assignment, then an attempt to add a second **Color** assignment would fail.

When adding standard attributes, the operation works as follows for the new attribute:

- If the new attribute already exists in the MDEX Engine but with a different type, an error is thrown and the new attribute is not added.
- If the new attribute already exists in the MDEX Engine 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.

Adding components to the incremental updates graph

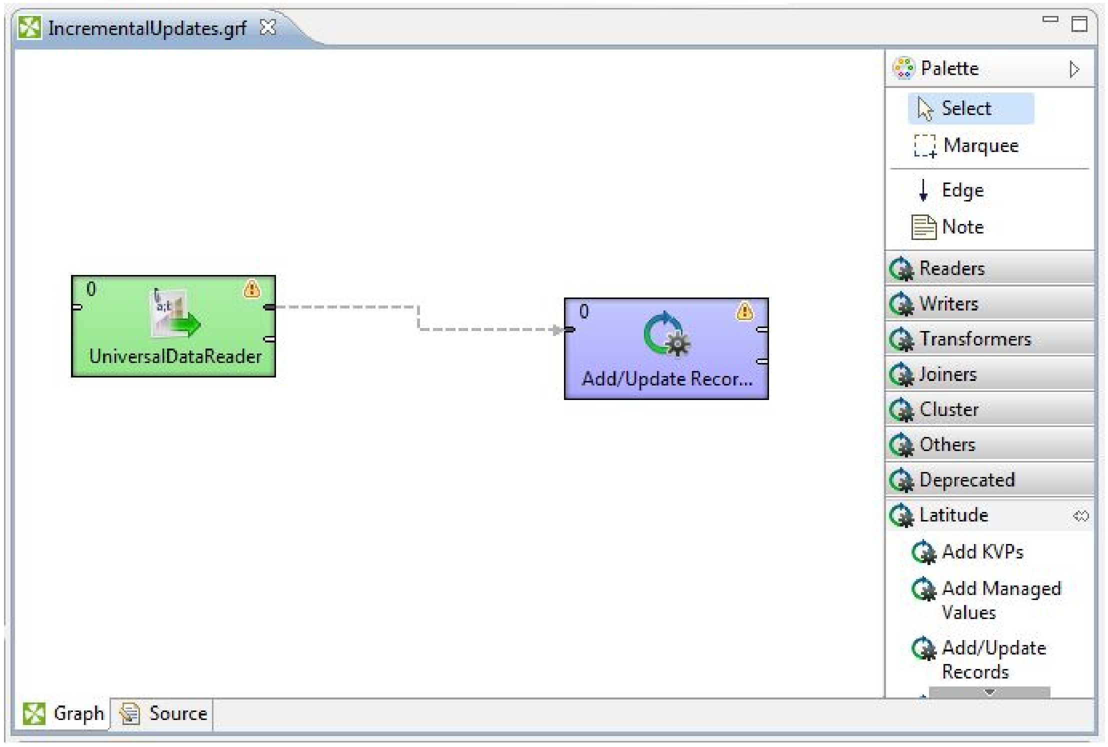
The graph for performing incremental updates requires a reader and the **Add/Update Records** connector.

This procedure assumes that you have created an empty graph.

To add components to a graph for incremental updates:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Palette pane, open the **Latitude** section and drag the **Add/Update Records** connector into the Graph Editor.
3. In the Palette pane, click **Edge** and use it to connect the two components.
4. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the two connected components should look like this:



The next tasks are to configure all three components.

Configuring the Reader and the Edge for incremental updates

The configuration of the incremental updates reader and edge components is almost identical to that of fresh index load graph.

This procedure assumes that you have added the incremental updates source file to the project's **data-in** folder.

To configure the **UniversalDataReader** and Edge components for incremental updates:

1. To configure the **UniversalDataReader** component for the incremental updates input file, use the same procedure as described in the topic titled "Configuring the Reader component" in Chapter 2 ("Full Index Loads of Records") of this guide.

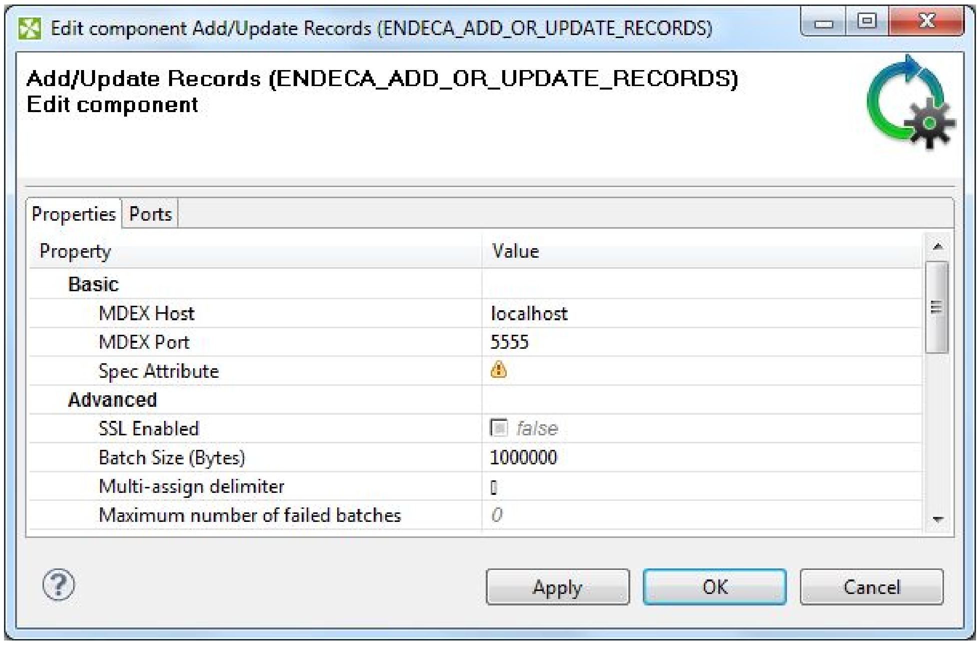
   The only difference is that you will be using your incremental updates file as the input file.
2. To configure the Edge component, use the same procedure as described in the topic titled "Configuring metadata for the Edge" in Chapter 2 ("Full Index Loads of Records") of this guide.
3. When you have finished your configuration, save the graph.

Configuring the Add/Update Records connector

You must configure the **Add/Update Records** connector with the location and port of the MDEX Engine, as well as the primary key for the records.

This procedure assumes that you have created a graph and added the **Add/Update Records** connector.

The Writer Edit Component dialog is where you configure the **Add/Update Records** connector:



Edit component Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)

Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Spec Attribute	⚠
Advanced	
SSL Enabled	☐ false
Batch Size (Bytes)	1000000
Multi-assign delimiter	▯
Maximum number of failed batches	0

To configure the **Add/Update Records** connector:

1. In the Graph window, double-click the **Add/Update Records** component.
   The Writer Edit Component dialog is displayed.
2. In the Writer Edit Component dialog, enter these mandatory settings in the Basic section:
   - **MDEX Host**: Enter the host name of the machine on which the MDEX Engine is running.
   - **MDEX Port**: Enter the port on which the MDEX Engine is listening for requests.
   - **Spec Attribute**: Enter the name of the property that is the primary key (record spec) for the records.
3. Still in the Writer Edit Component dialog, you can make these optional settings in the Advanced section:
   - **SSL Enabled**: Toggle this field to `true` if the MDEX Engine is SSL-enabled.
   - **Batch Size (Bytes)** : To change the default batch size (which is in bytes), enter a positive integer. Specifying 0 or a negative number will disable batching.
   - **Multi-assign delimiter**: Specify the character that separates multi-assign values in an input property. Keep in mind that this delimiter is different from the delimiter that separates properties.
   - **Maximum number of failed batches**: Enter a positive integer that sets the maximum number of batches that can fail before the ingest operation is ended. Entering 0 allows no failed batches.
4. When you have input all your changes, click **OK**.
5. Save the graph.

Running the incremental updates graph

After creating the graph and configuring the components, you can run the graph to load the incremental update records into the MDEX Engine.

To run the graph to load incremental updates:

1. Make sure that you have an MDEX Engine running on the host and port that are configured in the **Add/Update Records** connector.
2. Run the graph using one of the run methods.

   For example, you can click the green circle with white triangle icon in the Tool bar:

As the graph runs, the process of the graph execution is listed in the Console Tab. The execution is completed successfully when you see final output similar to this example of adding five new records:

```
INFO  [WatchDog] - ----------------------** Final tracking Log for phase
[0] **---------------------
INFO  [WatchDog] - Time: 06/06/11 10:53:21
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           5
        3      5      3
INFO  [WatchDog] - Incrementals           ENDECA_ADD_OR_UPDATE_RECORDS0
             FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           In:0           5
        3      5      3
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
             4927
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 !
```

As the example shows, the Final Tracking Log lists the number of records that were read in by the **UniversalDataReader** component and the number of records (5 in this example) that were sent to the MDEX Engine by the **Add/Update Records** connector.

Chapter 4

Loading the Attribute Schema

This chapter describes how to load your PDR and DDR configuration files into the MDEX Engine.

About attribute schema files

The attribute schema for your application is defined by the PDR and DDR files in the MDEX Engine.

Each Endeca standard attribute is defined by its PDR (Property Description Record). Each Endeca managed attribute is defined by its own PDR and also by a DDR (Dimension Description Record).

If you are loading your source records without first loading your attribute schema, the MDEX Engine will automatically create the PDRs for your standard attributes, using the system default settings. The values of these default settings are described in Chapter 1 of this guide.

However, it is recommended that you create your own PDR and DDR input records and then use the Latitude Data Integrator Designer to load that schema into the MDEX Engine. This process uses the **UniversalDataReader** component to read in the schema files and the **Add/Update Records** connector to load them into the MDEX Engine.

Loading PDRs

This topic provides an overview of the PDR load process.

From a high-level view, the steps you will follow to load your PDR schema into the MDEX Engine are:

1. Create the PDR input file. (Described in this chapter in the "Creating the PDR input file" topic.)
2. Either create a new project or re-use an existing one. (Not described in this chapter, as we will use the same project that was created in the "Creating a project" topic in Chapter 2.)
3. Create a graph and add the **UniversalDataReader** component and the **Add/Update Records** connector. (Not described in this chapter, as this procedure is the same as described in the "Adding Reader and Writer components" topic in Chapter 2.)
4. Configure the components. (Described in this chapter.)
5. Run the graph. (Not described in this chapter, as this procedure is the same as described in the "Running the graph" topic in Chapter 2.)

Keep in mind that if you are also loading DDR records, you should first load the PDRs that will be associated with the DDRs (unless the appropriate PDRs have already been loaded into the MDEX Engine).

Format of the PDR input file

The PDR input file defines one or more Endeca standard attributes, with the specific settings of some PDR properties.

The format of the PDR input file is:

- The first line of the file must be a delimited list of PDR properties that are to be set. Except for the `mdex-property_Key`, any PDR property that is not specified is set to its system default.
- The second and following lines are delimited lists of values for the PDR properties.

Note that although the real PDR properties use hyphens in their names, those in the input file cannot have hyphens. For example, the real `mdex-property_Key` property must be listed as `mdexproperty_Key`. The hyphens in the names are added by the **Add/Update Records** connector before the names are sent to the MDEX Engine.

Besides the `mdex-property_` prefix, the other property prefixes that cannot use hyphens in PDR or DDR input files are:

- `mdex-dimension_`
- `mdex-config_`
- `system-navigation_`

In addition, property names also cannot use hyphens in their names. Although the MDEX Engine will accept property names with hyphens, the Designer will not. Therefore, if you have a property name such as "Wine-Type", make sure you remove the hyphen from the name.

The following is an example of a PDR input file:

```
mdexproperty_Key|mdexproperty_DisplayName|mdexproperty_Type|mdexproper¬
ty_IsUnique|mdexproperty_IsSingleAssign|systemnavigation_ShowRecordCounts
WineID|Wine ID|mdex:int|true|true|true
WineName|Wine Name|mdex:string|false|false|true
WineType|Wine Type|mdex:string|false|false|true
Year|Year Grown|mdex:int|false|false|true
Flavors|Flavors|mdex:string|false|false|true
Price|Price|mdex:double|false|false|true
Designation|Designation|mdex:string|false|false|true
```

The example creates seven PDRs (such as WineID and WineType) and specifically sets the values of five of their PDR properties. As mentioned, the `mdexproperty_Key` is mandatory in order to set the name of the new PDRs. The `mdexproperty_Type` is used because the property type would otherwise default to `mdex:string`, and three of the new properties are numeric.

See Chapter 1 of this guide for the system default values used by the MDEX Engine when creating attributes.

Configuring the Reader for the PDR input file

This task describes how to configure the **UniversalDataReader** component to read in the PDR source data.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the PDR source file to the project's **data-in** folder.

To configure the Reader component for the PDR input file:

1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.

2. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the source data file and click **OK**.
3. Check the **Quoted strings** box so that its value changes to `true`.
4. Leave the **Number of skipped records** field set to the default of 0.

   The reason is that we want the first row (the PDR property names) to be read in and sent to the Writer component.
5. Click **OK** to apply your configuration changes to the Reader component.
6. Save the graph.

Configuring the Add/Update Records connector for PDR output

This topic describes how to configure the **Add/Update Records** connector for loading PDR data.

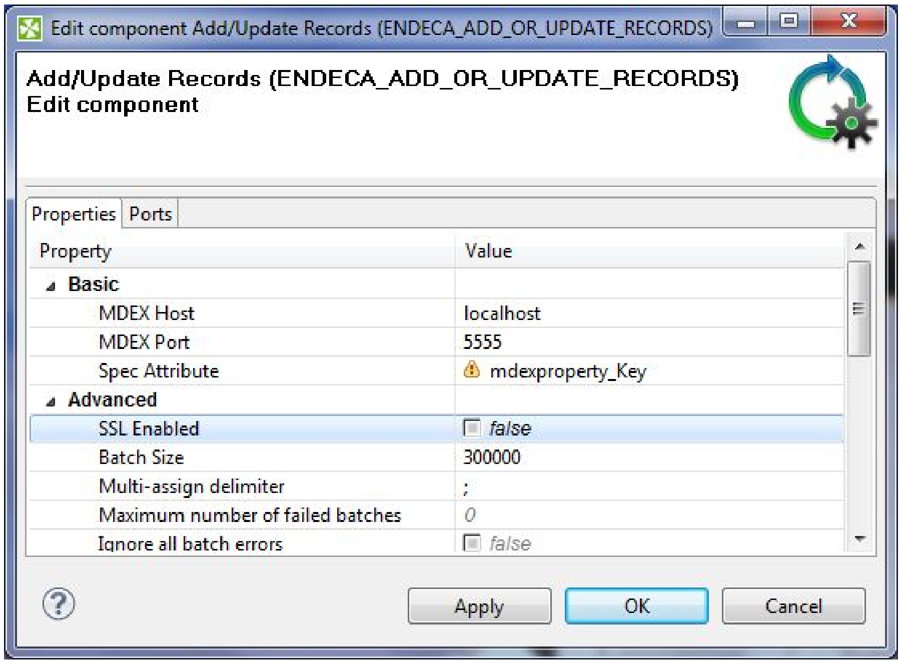
This procedure assumes that you have created a graph and added the **Add/Update Records** component.

To configure the **Add/Update Records** connector for PDR output:

1. In the Graph window, double-click the **Add/Update Records** component.
2. In the Writer Edit Component dialog, enter these settings:

   - **MDEX Host**: The host name of the machine on which the MDEX Engine is running.
   - **MDEX Port**: The port on which the MDEX Engine is listening for requests.
   - **Spec Attribute**: Set this value to **mdexproperty_Key** (this will be primary key for the PDR records).
   - You can leave the other settings at their default values.

   At this point, the Basic and Advanced sections of the dialog should look like this example:


Edit component Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Edit component
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Spec Attribute	mdexproperty_Key
Advanced	
SSL Enabled	false
Batch Size	300000
Multi-assign delimiter	;
Maximum number of failed batches	0
Ignore all batch errors	false

3. When you have input all your changes, click **OK**.
4. Save the project.

The next task is to configure the Edge component in the graph.

Configuring PDR metadata

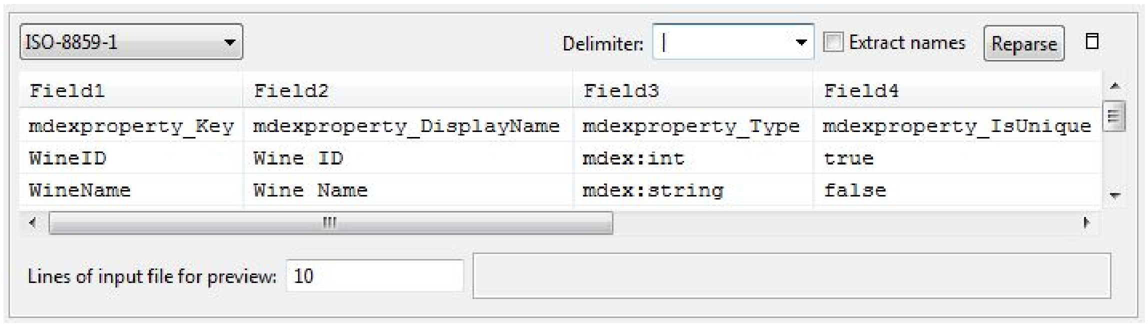
The Edge component must be configured with a Metadata definition.

This Metadata definition task will use the Metadata Editor. In the procedure, the column names will be extracted from the input file via a reparsing operation.

To configure the Metadata definition for the PDR Edge:

1. In the **Outline** pane, right-click on **Metadata**.
2. Select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
3. In the Flat File dialog, browse for the PDR input file, select it, and click **OK**.
4. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**.

   The PDR data is loaded into the Metadata Editor, with the properties named Field1, Field2, and so forth. For example, the middle pane of the Metadata Editor should look like this:



Field1	Field2	Field3	Field4
mdexproperty_Key	mdexproperty_DisplayName	mdexproperty_Type	mdexproperty_IsUnique
WineID	Wine ID	mdex:int	true
WineName	Wine Name	mdex:string	false

5. In the middle pane of the Metadata Editor:
   a) Check the **Extract names** box.
   b) Click **Reparse**.
   c) Click **Yes** in the Warning message.

   The correct property names are now displayed in the upper and middle panes of the Metadata Editor.
6. In the upper pane of the Metadata Editor:
   a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a name such as `PDR`.
   b) Make sure that the **Type** field of ***all*** the properties is set to type **string**.
   c) Verify that all properties have the correct delimiter character set (which is the pipe character for our example). The final property should have a new-line as the delimiter (\n on Linux and \r\n on Windows).
   d) When you have input all your changes, click **Finish**.
7. In the Graph Editor window, right-click on the Edge between the **UniversalDataReader** and **Add/Update Records** components. Choose **Select Metadata** and the metadata you have just configured, for example, **PDR (id:Metadata0)**.
8. Save the graph.

After creating the graph and configuring the components, you can run the graph to send the PDR data to the MDEX Engine. You can run the graph by clicking the green circle with white triangle icon in the Tool bar:

Loading DDRs

This topic provides an overview of the DDR load process.

From a high-level view, the steps you take to load your DDR schema into the MDEX Engine are as follows:

1. Create the DDR input file. (Described in this chapter in the "Creating the DDR input file" topic.)
2. Either create a new project or re-use an existing one. (Not described in this chapter, as we will use the same project that was created in the "Creating a project" topic in Chapter 2.)
3. Create a graph and add the **UniversalDataReader** component and the **Add/Update Records** connector. (Not described in this chapter, as this procedure is the same as described in the "Adding Reader and Writer components" topic in Chapter 2.)
4. Configure the components. (Described in this chapter.)
5. Run the graph. (Not described in this chapter, as this procedure is the same as described in the "Running the graph" topic in Chapter 2.)

Keep in mind that before loading DDR records, you should first load the PDR records that are associated with the DDRs (unless the appropriate PDRs have already been loaded into the MDEX Engine).

Format of the DDR input file

The DDR input file defines the Endeca managed attributes, with the specific settings of some DDR properties.

Similar to a PDR input file, the format of the DDR input file is:

- The first line must be a delimited list of DDR properties that are to be set. Except for the `mdex-dimension_Key`, any DDR property that is not specified is set to its system default.
- The second and following lines are delimited lists of values for the DDR properties. Each value must correspond to a DDR property.

Note that although the real DDR properties use hyphens in their names, those in the input file cannot have hyphens. For example, the real `mdex-dimension_Key` property must be listed as `mdexdimension_Key`. The hyphens in the names are added by the **Add/Update Records** connector before the names are sent to the MDEX Engine.

The following is an example of a DDR input file:

```
mdexdimension_Key|mdexdimension_EnableRefinements|mdexdimension_IsDimension¬
SearchHierarchical|mdexdimension_IsRecordSearchHierarchical
WineType|true|false|false
Designation|true|false|false
```

The example creates two DDRs (WineType and Designation) and sets the values of its DDR properties. As mentioned, the `mdexdimension_Key` is mandatory in order to set the name of the new DDRs.

Configuring the Reader and the Edge for DDRs

These two configurations are very similar to those for PDR loads.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component and the **Add/Update Records** connector. It also assumes that you have added an Edge and also added the DDR source file to the project's **data-in** folder.

To configure the **UniversalDataReader** and Edge components:

1. To configure the **UniversalDataReader** component for the DDR input file, use the same procedure as described in the topic titled "Configuring the Reader for the PDR input file" in this chapter.

   The only difference is that you will be using your DDR file as the input file.
2. To configure the Edge component, use the same procedure as described in the topic titled "Configuring PDR metadata" in this chapter.

   Be sure to use the **Extract names** and **Reparse** options on the Metadata Editor.
3. When you have finished your configuration, save the graph.

Configuring the Add/Update Records connector for DDR loads

This topic describes how to configure the **Add/Update Records** connector for loading DDR data.

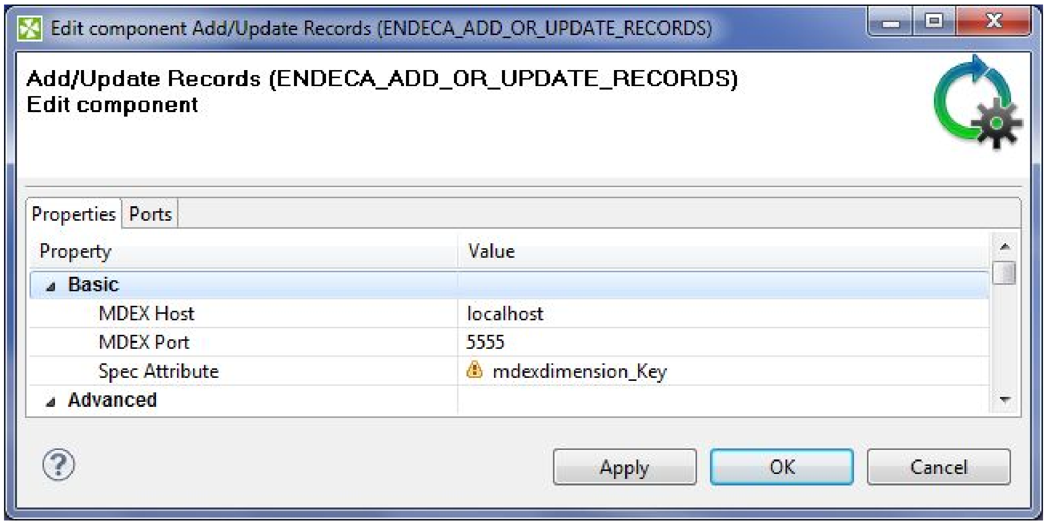
This procedure assumes that you have created a graph and added the **Add/Update Records** connector.

To configure the **Add/Update Records** connector for DDR output:

1. In the Graph window, double-click the **Add/Update Records** component.
2. In the Writer Edit Component dialog, enter these settings:

   - **MDEX Host**: The host name of the machine on which the MDEX Engine is running.
   - **MDEX Port**: The port on which the MDEX Engine is listening for requests.
   - **Spec Attribute**: Set this value to **mdexdimension_Key** (this will be primary key for the DDR records).
   - You can leave the other settings at their default values.

At this point, the Basic section of the dialog should look like this example:


Edit component Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Edit component
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Spec Attribute	mdexdimension_Key
Advanced	

3. When you have input all your changes, click **OK**.
4. Save the project.

After creating the graph and configuring the components, you can run the graph to send the DDR data to the MDEX Engine. You can run the graph by clicking the green circle with white triangle icon in the Tool bar:

Chapter 5

Loading Configuration Files

This chapter describes how to load the Global Configuration Record and the index configuration documents for the MDEX Engine.

Types of MDEX Engine configuration documents

The MDEX Engine offers a rich set of index configuration documents that allow you to customize your Endeca implementation.

The index configuration is the mechanism for implementing a number of Endeca features such as search and ranking. The index configuration documents are created automatically by the `mkmdex` utility with a set of defaults that are described in the following topics. The index configuration documents are stored in the MDEX indexes database and loaded into the MDEX Engine at startup.

The documents are as follows:

Index Configuration Document	Purpose
`dimsearch_config`	Configures attributes (both Standard Attributes and Managed Attributes) for value search.
`precedence_rules`	Sets precedence rules, which provide a way to delay the display of attributes until they offer a useful refinement of the navigation state. Precedence rules allow your Endeca implementation to delay the display of a refinement until the user triggers it, making navigation through the data easier and avoiding information overload.
`recsearch_config`	Configures record search, including search interfaces which control record search behavior for groups of attributes. Some of the features that can be specified for a search interface include relevance ranking, matching across multiple attributes, and partial matching.
`relrank_strategies`	Sets relevance ranking, which is used to control the order of results that are returned in response to a record search.
`stop_words`	Sets stop words, which are words that are set to be ignored by the MDEX Engine.
`thesaurus`	The thesaurus allows the system to return matches for related concepts to words or phrases contained in user queries.

Recommended order for loading the index configuration

The recommended order of loading is:

1. Load the record schema (PDRs and DDRs) first. It does not matter if the actual data records are loaded, but the PDRs and DDRs are important because they create the properties that should be referenced by the configuration files.
2. `relrank_strategies` document (necessary if a relevance ranking strategy is referenced by the next two documents)
3. `recsearch_config` document
4. `dimsearch_config` document
5. `precedence_rules` document
6. `stop_words` document
7. `thesaurus` document

Global Configuration Record

The Global Configuration Record (GCR) stores global configuration settings for the MDEX Engine.

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 *Latitude Developer's Guide*.

When loading your changes for the GCR , keep these requirements in mind:

- The `mdex-config_Key` property must be unique and single-assign. The value must be `global` for the property.
- The GCR must contain valid values for all of its properties. None of its properties can be omitted.
- The GCR cannot have any arbitrary, user-defined properties.

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 is a sample GCR:

```
<mdex:record>
  <mdex-config_Key>global</mdex-config_Key>
  <mdex-config_EnableValueSearchWildcard>true</mdex-config_EnableValueSearch¬
Wildcard>
  <mdex-config_MergePolicy>aggressive</mdex-config_MergePolicy>
  <mdex-config_SearchChars>+_</mdex-config_SearchChars>
  <mdex-config_SpellingRecordMinWordOccur>2</mdex-config_SpellingRecordMin¬
WordOccur>
  <mdex-config_SpellingRecordMinWordLength>4</mdex-config_SpellingRecordMin¬
WordLength>
  <mdex-config_SpellingRecordMaxWordLength>24</mdex-config_SpellingRecord¬
MaxWordLength>
  <mdex-config_SpellingDValMinWordOccur>5</mdex-config_SpellingDValMinWor¬
dOccur>
  <mdex-config_SpellingDValMinWordLength>3</mdex-config_SpellingDValMin¬
WordLength>
  <mdex-config_SpellingDValMaxWordLength>20</mdex-config_SpellingDValMax¬
WordLength>
</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.

dimsearch_config document

This document sets the configuration for value search.

The default `dimsearch_config` document contains an empty configuration:

```
<DIMSEARCH_CONFIG/>
```

In the configuration document, you can use the `RELRANK_STRATEGY` attribute to specify a relevance ranking strategy to use on the results. If you do so, you must first use the `relrank_strategies` document to configure the relevance ranking strategy in the MDEX Engine.

Sample dimsearch_config document

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

```
<DIMSEARCH_CONFIG FILTER_FOR_ANCESTORS="FALSE" RELRANK_STRATEGY="WineRel¬
Rank"/>
```

As mentioned above, the WineRelRank strategy must have been configured previously with the `rel¬
rank_strategies` document.

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="dimsearch_config">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="dimsearch_config"` attribute references the `dimsearch_config` document.

Run-time error

If the `RELRANK_STRATEGY` attribute in the document references a non-existent relevance ranking strategy, the load operation will fail with an error similar to this example:

```
ERROR [WatchDog] - Graph execution finished with error
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT3 finished with
status: ERROR caused by: Error applying updates:
Invalid Relevance ranking strategy "WineRelRank" in DIMSEARCH_CONFIG element.
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT3 error details:
org.apache.axis2.AxisFault: Error applying updates: Invalid Relevance
ranking strategy
"WineRelRank" in DIMSEARCH_CONFIG element.
```

To correct this error, first use the `relrank_strategies` document to create the relevance ranking strategy in the MDEX Engine before you attempt to load your `dimsearch_config` document.

precedence_rules document

This document sets your application's precedence rules, which provide a way to delay the display of attributes until they offer a useful refinement of the navigation state.

Precedence rules allow your Endeca implementation to delay the display of a refinement until the user triggers it, making navigation through the data easier and avoiding information overload.

The default `precedence_rules` document does not define any rules:

```
<PRECEDENCE_RULES/>
```

The attributes referenced in the `precedence_rules` document do not have to exist in the MDEX Engine at ingest time. That is, no error checking is done for the existence of the attributes (this allows the rules to be created even before the data they reference is loaded). For this reason, you must make sure that the attributes are spelled correctly in the input file and will exist in the MDEX Engine.

Note that if the source attribute in a precedence rule does not exist but its destination attribute does exist, then the precedence rule will never be triggered. This behavior effectively hides the destination attribute from refinements. To correct this behavior, either remove the rule or create the source attribute in the MDEX Engine.

Note also that the type of the source attribute value should be specified correctly. That is, if a source attribute value for the precedence rule is from a standard attribute, PROPERTY must be specified for the type. If a source attribute value is from a managed attribute, STANDARD or LEAF must be specified for the type.

Sample precedence_rules document

To define precedence rules, you need to create a text input file similar to this example which shows a STANDARD-type precedence rule.

```
<PRECEDENCE_RULES>
  <PRECEDENCE_RULE
    DEST_DIMENSION="Winery" DEST_DVAL_SPEC="/"
    SRC_DIMENSION="Region" SRC_DVAL_SPEC="/" TYPE="STANDARD"/>
</PRECEDENCE_RULES>
```

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="precedence_rules">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="precedence_rules"` attribute references the `precedence_rules` document.

recsearch_config document

This document configures record search, including search interfaces which control record search behavior for groups of attributes.

Some of the features that can be specified for a search interface include relevance ranking, matching across multiple Endeca attributes, partial matching, and enabling snippeting for one or more Endeca attributes.

The default `recsearch_config` document contains an empty configuration:

```
<RECSEARCH_CONFIG/>
```

In the configuration document, you can use the `RELRANK_STRATEGY` attribute to specify a relevance ranking strategy to use on the results. If you do so, you must first use the `relrank_strategies` document to configure the relevance ranking strategy in the MDEX Engine.

Sample recsearch_config document

To configure record search, you need to create a text input file similar to this example:

```
<RECSEARCH_CONFIG>
  <SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER" CROSS_FIELD_RELE¬
VANCE_RANK="0"
     DEFAULT_RELRANK_STRATEGY="WineRelRank" NAME="WineSearch">
    <MEMBER_NAME RELEVANCE_RANK="4">WineType</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="3">Wine</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="2">Winery</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
    <MEMBER_NAME SNIPPET_SIZE=10>Description</MEMBER_NAME>
    <PARTIAL_MATCH MAX_WORDS_OMITTED="1" MIN_WORDS_INCLUDED="2"/>
  </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

The example creates a search interface named WineSearch that uses the WineRelRank strategy as its default relevance ranking strategy. The example also configures partial matching and enables snippeting for the Description attribute.

As mentioned above, the WineRelRank strategy must have been configured previously with the `rel¬ rank_strategies` document, and the four referenced Endeca attributes must exist in the MDEX Engine.

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="recsearch_config">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="recsearch_config"` attribute references the `recsearch_config` document.

Run-time errors

If the `RELRANK_STRATEGY` attribute in the document references a non-existent relevance ranking strategy, the load operation will fail with an error similar to this example:

```
ERROR [WatchDog] - Graph execution finished with error
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT0 finished with
status: ERROR caused by: Error applying updates:
Invalid Relevance Ranking Strategy "WineRelRank" referenced
in SEARCH_INTERFACE "WineSearch"
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT0 error details:
org.apache.axis2.AxisFault: Error applying updates: Invalid
Relevance Ranking Strategy "WineRelRank" referenced
in SEARCH_INTERFACE "WineSearch"
```

To correct this error, first use the `relrank_strategies` document to create the relevance ranking strategy in the MDEX Engine before you attempt to load your `recsearch_config` document.

In addition, the Endeca attributes referenced in the search interface must also exist in the MDEX Engine. Otherwise, the load operation will fail with an error similar to this example:

```
Error applying updates: No property with the name "WineType" exists for
search interface "WineSearch"
```

To correct this error, first load your PDRs before loading the configuration documents.

relrank_strategies document

This document configures the relevance ranking strategies for a Latitude application.

Relevance ranking is used to control the order of results that are returned in response to a record search. An individual relevance ranking strategy is expressed in a RELRANK_STRATEGY element, which in turn is made of individual relevance ranking modules such as RELRANK_EXACT, RELRANK_FIELD, and so on.

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

```
<RELRANK_STRATEGIES/>
```

Sample relrank_strategies document

This example creates a relevance ranking strategy named WineRelRank that consists of the RELRANK_INTERP and RELRANK_FIELD relevance ranking modules.

```
<RELRANK_STRATEGIES>
  <RELRANK_STRATEGY NAME="WineRelRank">
    <RELRANK_INTERP/>
    <RELRANK_FIELD/>
  </RELRANK_STRATEGY>
</RELRANK_STRATEGIES>
```

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="relrank_strategies">
$XMLString
```

```
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="relrank_strategies"` attribute references the `relrank_strategies` document.

stop_words document

This document sets the stop words for queries.

Stop words are words that should be eliminated from a query before it is processed by the MDEX Engine.

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

```
<STOP_WORDS/>
```

Sample stop_words document

This example sets the stop words for a wine application.

```
<STOP_WORDS>
  <STOP_WORD>wine</STOP_WORD>
  <STOP_WORD>bottle</STOP_WORD>
  <STOP_WORD>an</STOP_WORD>
  <STOP_WORD>of</STOP_WORD>
  <STOP_WORD>the</STOP_WORD>
</STOP_WORDS>
```

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="stop_words">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="stop_words"` attribute references the `stop_words` document.

thesaurus document

This document configures the thesaurus for your application.

The thesaurus allows the system to return matches for related concepts to words or phrases contained in user queries.

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

```
<THESAURUS/>
```

Sample thesaurus document

This example sets the thesaurus entries for a wine application.

```
<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_ENTRY>
    <THESAURUS_FORM>zin</THESAURUS_FORM>
    <THESAURUS_FORM>zinfandel</THESAURUS_FORM>
  </THESAURUS_ENTRY>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>cab</THESAURUS_FORM>
    <THESAURUS_FORM>cabernet</THESAURUS_FORM>
  </THESAURUS_ENTRY>
</THESAURUS>
```

Request structure text

When you configure the **WebServiceClient** component, you add the following request text in the **Edit request structure** dialog:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="thesaurus">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

The `name="thesaurus"` attribute references the `thesaurus` document.

Loading the configuration documents

This section describes how to create and configure a graph for loading the index configuration documents.

The procedure is basically the same for all the index configuration documents. The only exceptions are the format of the input file and the document name used in this element in the **Edit request structure** dialog:

```
<mdex:configDocument name="relrank_strategies">
```

The individual topics for the configuration documents in this chapter describe these exceptions.

Graph components

The graphs use the **UniversalDataReader** component to read in the configuration document. The strategy is to configure the reader to read the entire flat file as one field of string data type. The string (named XMLString) is then passed to the writer component.

The **WebServiceClient** writer component uses the Configuration Web Service's `config-service:putConfigDocuments` operation to send the configuration document to the MDEX Engine.

Creating a graph

This task describes how to create an empty graph for loading a configuration document.

The only prerequisite for this task is that you must have created a Data Integrator Designer project. Keep in mind that a project can have multiple graphs, which means that you can create this graph in an existing project.

To create an empty graph for your configuration documents:

1. In the Navigator pane, right-click the **graph** folder.
2. Select **New** > **ETL Graph**.
3. In the **Create new graph** dialog:
   a) Type in the name of the graph, such as **LoadConfig**.
   b) Optionally, type in a description.
   c) You can leave the **Allow inclusion of parameters from external file** box checked.
   d) Click **Next** when you finish.
4. In the **Output** dialog, click **Finish**.

Adding components to the graph

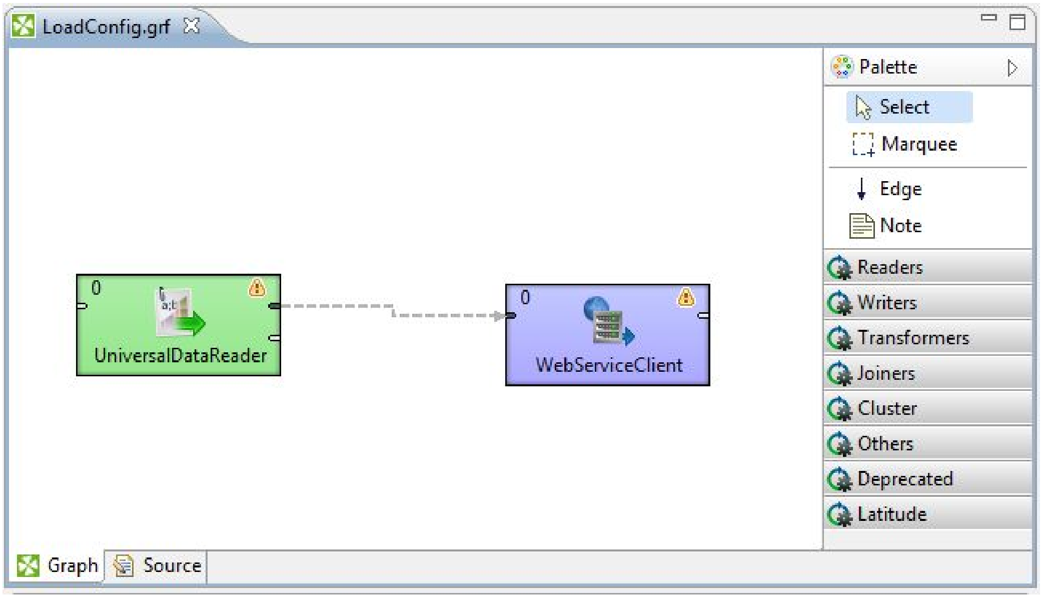
This tasks describes how to add the **UniversalDataReader** and **WebServiceClient** components to the graph.

In addition, an Edge component will be added to connect the two components.

To add components to the graph:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Palette pane, open the **Others** section and drag the **WebServiceClient** component into the Graph Editor.
3. In the Palette pane, click **Edge** and use it to connect the two components.
4. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the two connected components should look like this:



The next tasks are to configure these components.

Configuring the Reader for the configuration document

This task describes how to configure the **UniversalDataReader** component to read in the configuration document.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the configuration document source file to the project's **data-in** folder.

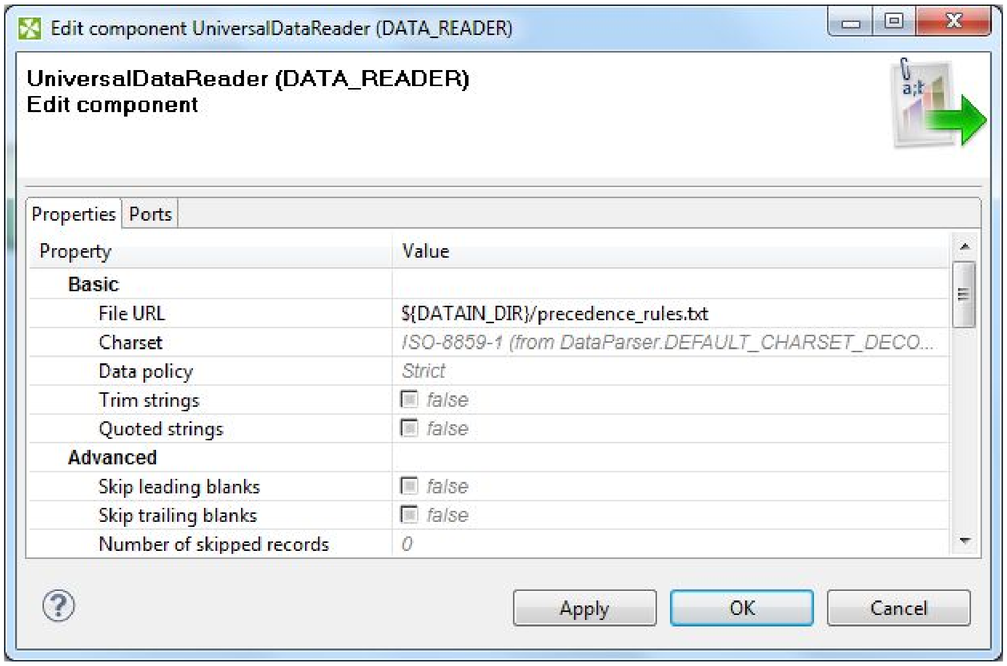
Important: The procedure also assumes that you have loaded your attribute schema (PDRs and DDRs) into the MDEX Engine. This is because if the configuration document specifies an attribute to use, that attribute should already exist in the MDEX Engine; if it does not exist, the MDEX Engine may reject the configuration document and LDI will display a load error.

To configure the **UniversalDataReader** component for the configuration input document:

1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
2. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the source data file and click **OK**.
3. Leave the **Quoted strings** box to its default value of `false` and the **Number of skipped records** field to its default of 0.

   You can also leave the other settings to their default values.
4. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
5. Save the graph.

After step 3, the Reader Edit Component dialog should look like this example:


Edit component UniversalDataReader (DATA_READER)

UniversalDataReader (DATA_READER)
Edit component

Property	Value
Basic	
File URL	${DATAIN_DIR}/precedence_rules.txt
Charset	ISO-8859-1 (from DataParser.DEFAULT_CHARSET_DECO...
Data policy	Strict
Trim strings	false
Quoted strings	false
Advanced	
Skip leading blanks	false
Skip trailing blanks	false
Number of skipped records	0

The next task is to configure the Edge.

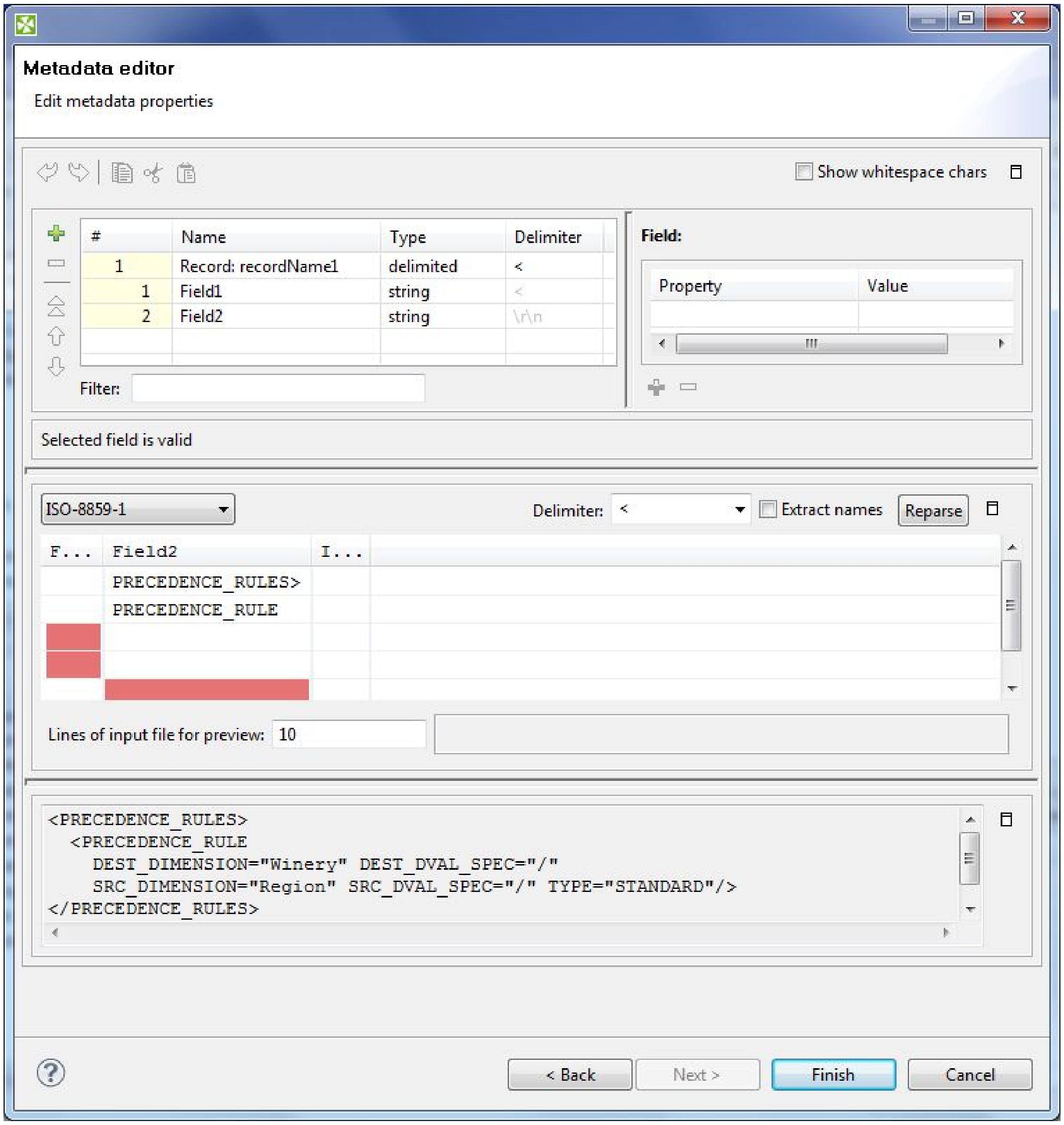
Configuring metadata for configuration documents

The Edge component must be configured with a Metadata definition for loading a configuration document.

The prerequisite for this task is that an Edge component must exist in the graph.

Note: This procedure will configure metadata for loading the `precedence_rules` configuration document. The procedure for loading the other configuration documents is identical, with the exception that at Step 3 you select the name of the appropriate file.

Most of the metadata configuration will be done in the Metadata Editor. This example shows the editor just after the source `precedence_rules` document was input.



#	Name	Type	Delimiter
1	Record: recordName1	delimited	<
1	Field1	string	<
2	Field2	string	\r\n

Property	Value

Selected field is valid

ISO-8859-1

F...	Field2	I...
	PRECEDENCE_RULES>	
	PRECEDENCE_RULE	

Lines of input file for preview: 10

```
<PRECEDENCE_RULES>
  <PRECEDENCE_RULE
    DEST_DIMENSION="Winery" DEST_DVAL_SPEC="/"
    SRC_DIMENSION="Region" SRC_DVAL_SPEC="/" TYPE="STANDARD"/>
</PRECEDENCE_RULES>
```

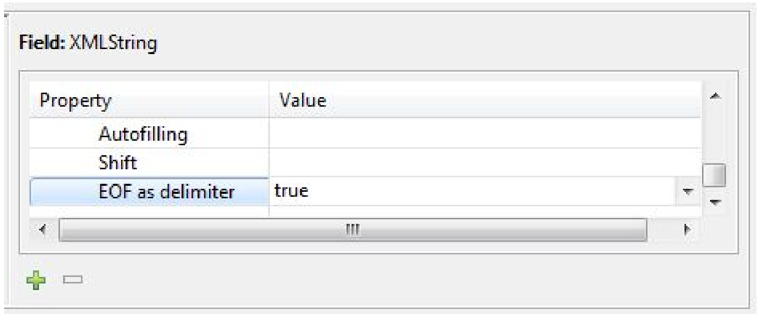
To configure the Metadata definition for configuration documents:

1. Right-click on the Edge and select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
2. In the Flat File dialog, click the **Browse** button, which brings up the **URL Dialog**.
3. In the URL Dialog, double-click the **data-in** folder, select the source data file, and click **OK**.
   As a result, the Flat File dialog is populated with source data from the input file.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed, looking like the example above.
5. In the Record pane, make these changes to the **Record:recordName1** entity:

a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a name that is appropriate for your data, such as **PrecRules** for the `precedence_rules` configuration document.
b) Leave the **Type** field set to `delimited`.
c) Leave the **Delimiter** field as-is for now. (You will delete it in Step 9.)

6. In the Record pane, delete all record fields except for the **Field1** record field (that is, delete **Field2**, **Field3**, and so on.)
7. In the Record pane, make these changes to the **Field1** record field:

a) Although optional, you should change the **Field1** default name to a more descriptive name, such as **XMLString**.
b) Leave the **Type** field set to **String**.
c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as in this example:


Field: XMLString

Property	Value
Autofilling	
Shift	
EOF as delimiter	true

8. When you have input all your changes in the Metadata Editor, click **Finish**. (For now, ignore the red (invalid metadata) warning messages.)
9. Now you must remove the record delimiter from the metadata. To do so:

a) In the graph, click the **Source** icon (which is next to the **Graph** icon).
b) In the Record element (which is a child of the Metadata element), find the **fieldDelimiter** and **recordDelimiter** attributes. For example, on Windows, these attributes look like this:

```
<Metadata id="Metadata0"
   previewAttachment="${DATAIN_DIR}/precedence_rules.txt"
   previewAttachmentCharset="ISO-8859-1">
<Record fieldDelimiter="&lt;" name="PrecRules"
   previewAttachment="${DATAIN_DIR}/precedence_rules.txt"
   previewAttachmentCharset="ISO-8859-1"
   recordDelimiter="\r\n" skipSourceRows="0" type="delimited">
<Field eofAsDelimiter="true" name="XMLString" type="string"/>
</Record>
</Metadata>
```

c) Delete the **fieldDelimiter** and **recordDelimiter** attributes.
d) While still within the Source view, right-click and select **Save** to save the graph.

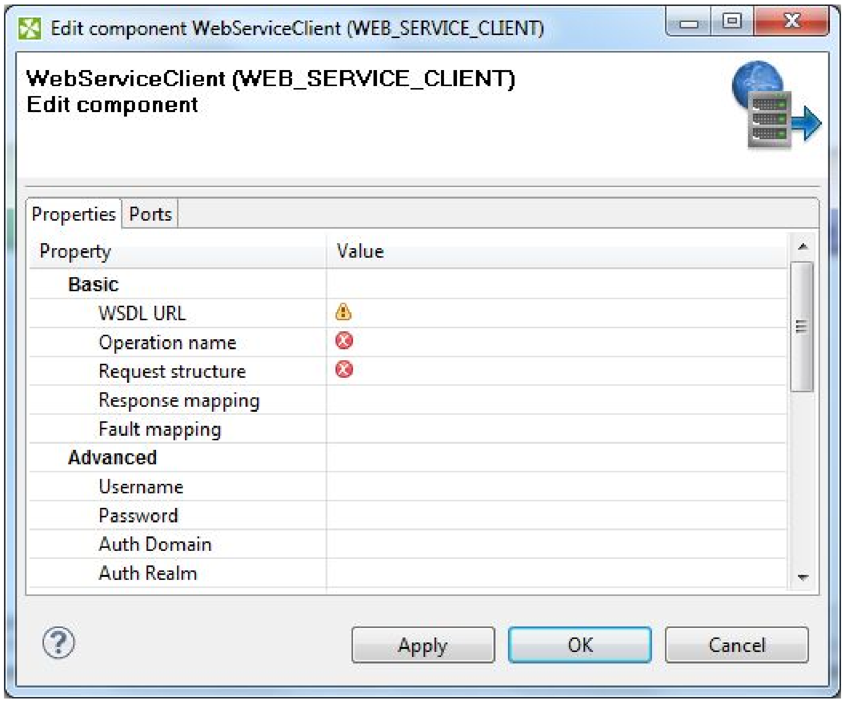
Configuring the WebServiceClient component

You must configure the **WebServiceClient** component to communicate with the Endeca Configuration Web service.

This procedure will configure metadata for loading the `precedence_rules` configuration document, and therefore assumes that you have added the configuration document source file to the project's **data-in** folder. The procedure for loading the other configuration documents with the **WebServiceClient** component is identical, with the exception that at Step 7 you specify the name of the appropriate configuration document in the `mdex:configDocument` element:

```
<mdex:configDocument name="precedence_rules">
```

The Writer Edit Component dialog is where you configure the **WebServiceClient** component:


Edit component WebServiceClient (WEB_SERVICE_CLIENT)

WebServiceClient (WEB_SERVICE_CLIENT)
Edit component

Properties | Ports

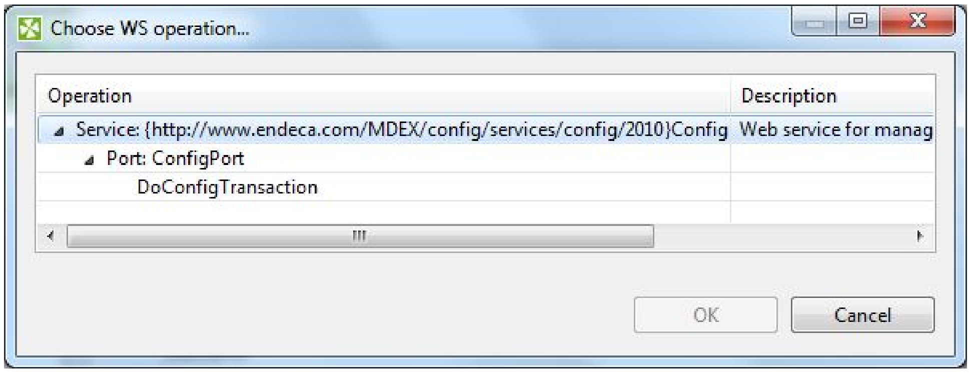
Property	Value
Basic	
WSDL URL	⚠
Operation name	⊗
Request structure	⊗
Response mapping	
Fault mapping	
Advanced	
Username	
Password	
Auth Domain	
Auth Realm	

To configure the **WebServiceClient** component:

1. Make sure that the MDEX Engine is running and the Configuration Web service is available by issuing this URL command from your browser (be sure to use the correct port number for your MDEX Engine):

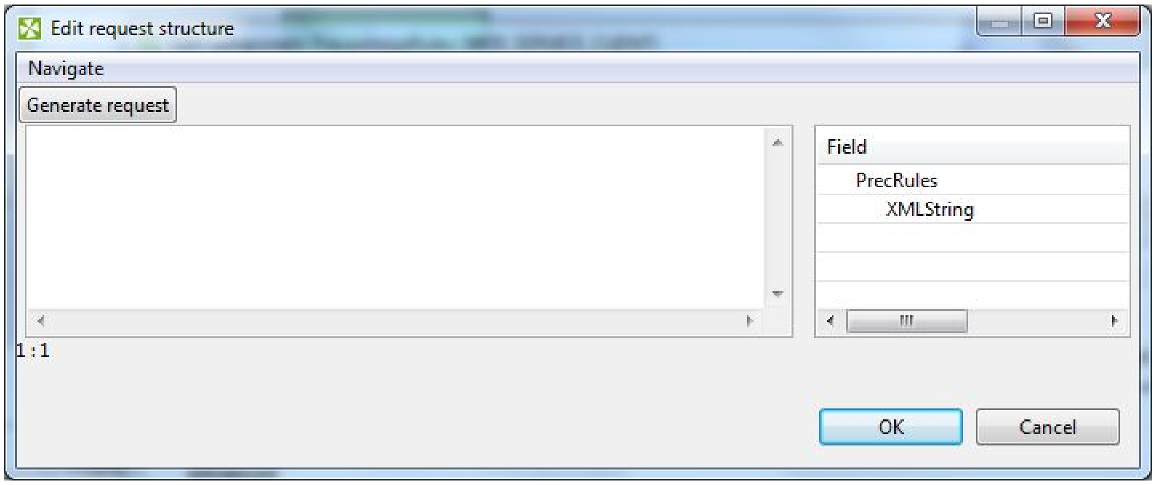
   ```
   http://localhost:5555/ws/config?wsdl
   ```

   The URL command show return the WSDL of the Web service.
2. In the Graph window, double-click the **WebServiceClient** component.
   The Writer Edit Component dialog is displayed.
3. In the **WSDL URL** field, enter the same URL as in Step 1.
4. In the **Operation name** field, click the **...** browse button, which displays the **Choose WS operation** dialog:


Choose WS operation...

Operation	Description
Service: {http://www.endeca.com/MDEX/config/services/config/2010}Config	Web service for manag
Port: ConfigPort	
DoConfigTransaction	

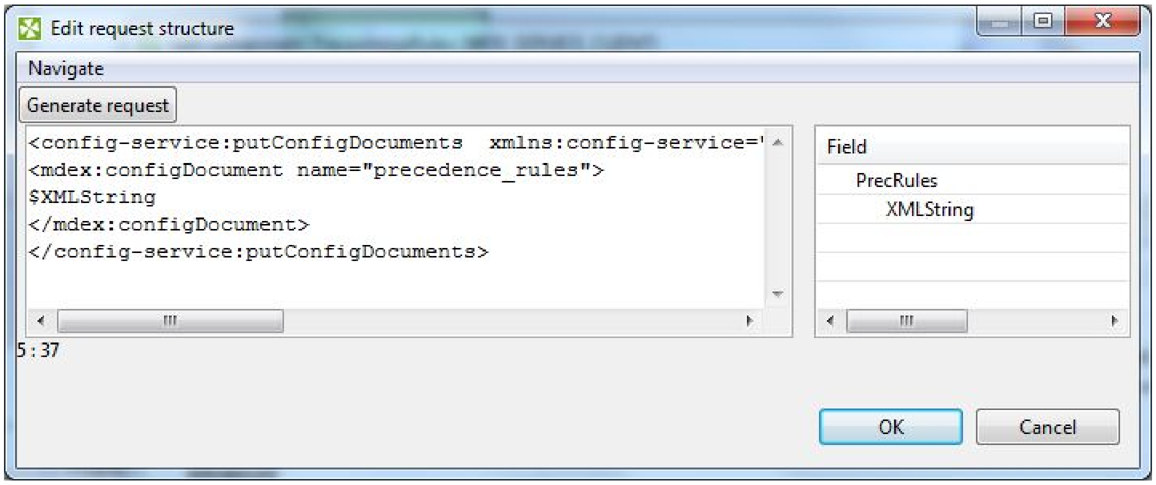
5. In the **Choose WS operation** dialog, select **DoConfigTransaction** and then click **OK**.
   The name of the Web service operation is entered in the **Operation name** field.
6. Click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the browse button to display the **Edit request structure** dialog:



7. Add this text to the **Generate request** field:

```
<config-service:putConfigDocuments
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="precedence_rules">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

   At this point, the **Edit request structure** dialog should look like this example:



```
<config-service:putConfigDocuments  xmlns:config-service='
<mdex:configDocument name="precedence_rules">
$XMLString
</mdex:configDocument>
</config-service:putConfigDocuments>
```

8. After adding the request text in the **Edit request structure** dialog, click **OK**.
9. When you have input all your changes in the Edit Component dialog, click **OK**.
10. Save the graph.

After creating the graph and configuring the components, you can run the graph to send the configuration data to the MDEX Engine. You can run the graph by clicking the green circle with white triangle icon in the Tool bar:

Loading the GCR

This topic provides an overview of how to load the GCR into the MDEX Engine.

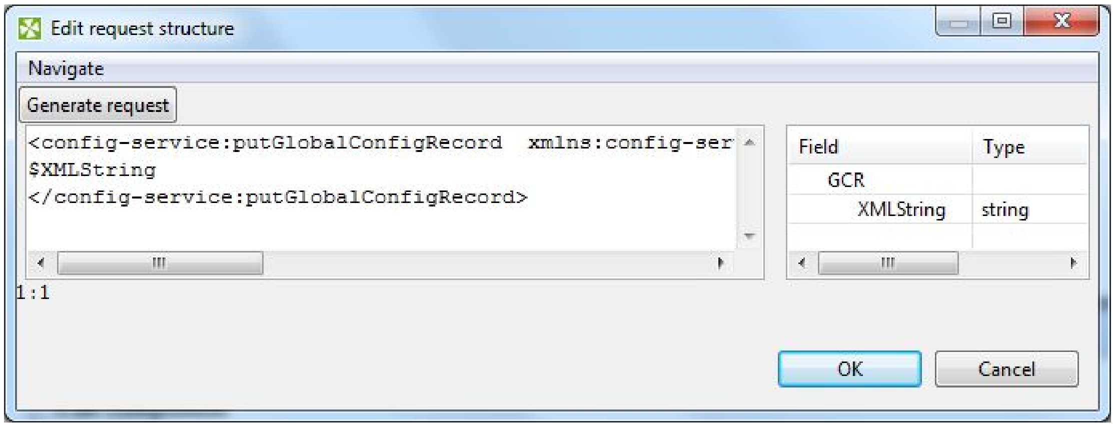
Loading the Global Configuration Record (GCR) into the MDEX Engine is very similar to loading the index configuration documents. The only difference is the format of the request text that you add to the **Edit request structure** dialog. This GCR-specific request text is shown in Step 6 below.

To load the GCR:

1. Create a graph, as described in the "Creating a graph" topic in this chapter.
2. Add your GCR input file to the project's **data-in** folder.
3. Add the **UniversalDataReader** and **WebServiceClient** components to the graph, as described in the "Adding components to the graph" topic in this chapter.
4. Configure the **UniversalDataReader** component, as described in the "Configuring the Reader for the configuration document" topic in this chapter.
5. Configure the metadata, as described in the "Configuring metadata for configuration documents" topic in this chapter.
6. Configure the **WebServiceClient** component, as described in the "Configuring the WebServiceClient component" topic in this chapter. The only difference is that you add this text to the **Generate request** field:

```
<config-service:putGlobalConfigRecord
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/con¬
fig/2010"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
$XMLString
</config-service:putGlobalConfigRecord>
```

   At this point, the **Edit request structure** dialog should look like this example:



```
<config-service:putGlobalConfigRecord  xmlns:config-ser
$XMLString
</config-service:putGlobalConfigRecord>
```

1:1

Field	Type
GCR	
XMLString	string

7. Make sure you save the graph.

You can run the graph by clicking the green circle with white triangle icon in the Tool bar:

Note that if you changed the spelling settings, you should rebuild the aspell dictionary by running the `admin?op=updateaspell` administrative operation.

Chapter 6

Adding Key-Value Pairs

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

About key-value pair data

The **Add KVPs** connector can add key-value pair data to MDEX Engine records.

The two main use cases for the **Add KVPs** connector are:

- To ingest source data that is stored in a key-value pair format instead of the more traditional rectangular data model.
- When you do not what the schema is ahead of time.

With either case, you have the option of loading data in rows (with the **Add/Update Records** connector) that 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** connector 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 meanings of these schema properties are as follows:

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 MDEX Engine, it is automatically created by DIWS with system default values.
`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

Schema property	Meaning
	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 will be `mdex:string`. If the standard attribute already exists, you can specify an empty value for `mdexType`.

The following is a simple example of an input file for the **Add KVPs** connector:

```
specKey|specValue|kvpKey|kvpValue|mdexType
WineID|51841|Designation|Best buy|
WineID|48191|Flavors|Cherry|
WineID|48191|Flavors|Blueberry|
WineID|48197|Drinkability|Drink now|
WineID|48197|Location|42.365615 -71.075647|mdex:geocode
```

The example adds a Designation assignment to Record 51841, two Flavors assignments to Record 48191 (Flavors is a multi-assign attribute), and a Drinkability assignment to Record 48197. In addition, a new geocode standard attribute named Location is created in the MDEX Engine and added to Record 48197.

Configuring the Reader for the KVP input file

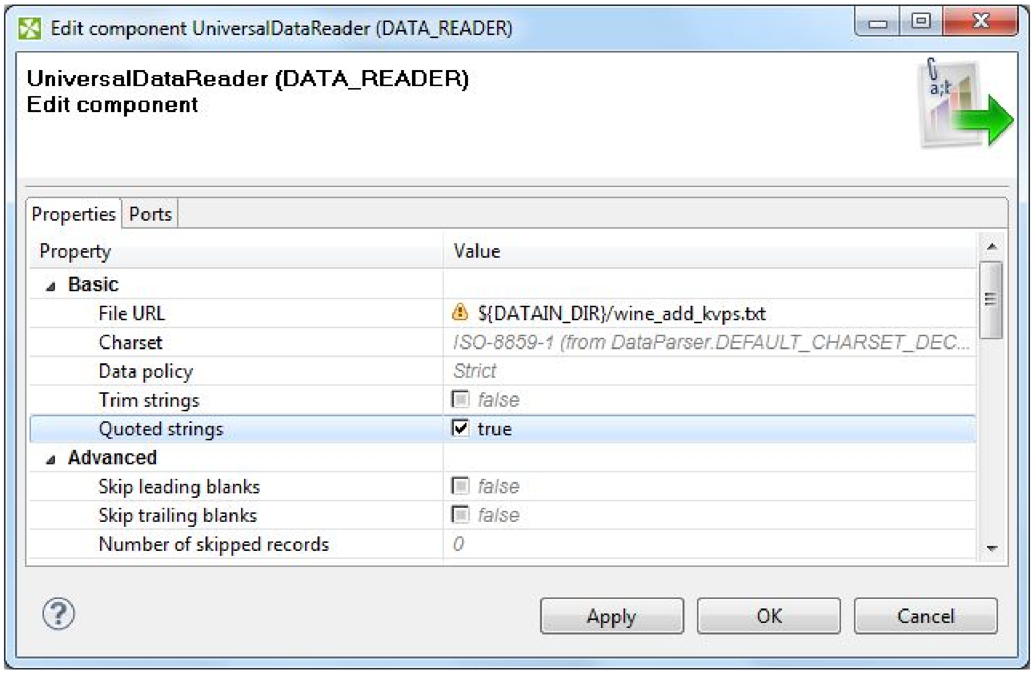
This task describes how to configure the **UniversalDataReader** component to read in the KVP data.

This procedure assumes that you have created a graph for the KVP components and that you have copied the input file into the **data-in** folder in the Navigation pane of the project. The procedure also assumes that you will be using the **UniversalDataReader** component to read in the KVP input data.

To configure the **UniversalDataReader** component for the KVP input file:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
3. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the KVP input file and click **OK**.
4. Check the **Quoted strings** box so that its value changes to `true`.
5. Leave the **Number of skipped records** field set to the default of 0.
6. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
7. Save the graph.

After the component is configured, the Reader Edit Component dialog should look like this example:


Edit component UniversalDataReader (DATA_READER)

UniversalDataReader (DATA_READER)
Edit component

Property	Value
Basic	
File URL	${DATAIN_DIR}/wine_add_kvps.txt
Charset	ISO-8859-1 (from DataParser.DEFAULT_CHARSET_DEC...
Data policy	Strict
Trim strings	false
Quoted strings	✓ true
Advanced	
Skip leading blanks	false
Skip trailing blanks	false
Number of skipped records	0

Configuring the Add KVPs connector

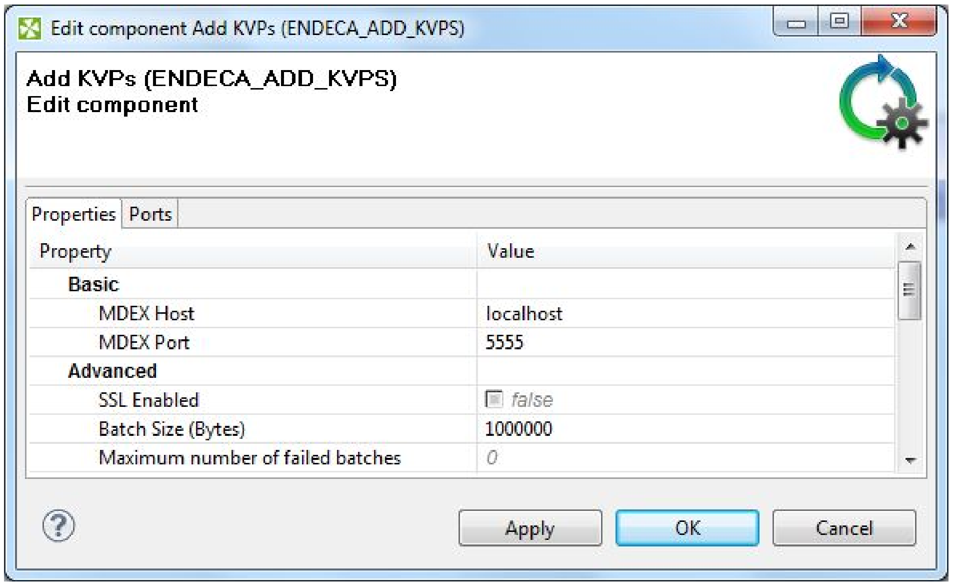
You must configure the **Add KVPs** connector to properly connect to your MDEX Engine.

This procedure assumes that you have created a graph for the **Add KVPs** connector.

To configure the **Add KVPs** connector:

1. In the Palette pane, open the **Latitude** section and drag the **Add KVPs** connector into the Graph Editor.
2. In the Graph window, double-click the **Add KVPs** connector.
   The Writer Edit Component dialog is displayed.
3. In the Writer Edit Component dialog, enter these settings:
   a) **MDEX Host**: The host name of the machine on which the MDEX Engine is running.
   b) **MDEX Port**: The port on which the MDEX Engine is listening for requests.
   c) **SSL Enabled**: Toggle this field to `true` if the MDEX Engine is SSL-enabled.
   d) **Batch Size (Bytes)**: To change the default batch size, enter a positive integer. Specifying 0 or a negative number will disable batching.
   e) **Maximum number of failed batches**: Enter a positive integer that sets the maximum number of batches that can fail before the ingest operation is ended. Entering 0 allows no failed batches.
4. When you have input all your changes, click **OK**.
5. Save the graph.

After configuration, the Writer Edit Component dialog should look like this example:


Edit component Add KVPs (ENDECA_ADD_KVPS)

Add KVPs (ENDECA_ADD_KVPS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Advanced	
SSL Enabled	false
Batch Size (Bytes)	1000000
Maximum number of failed batches	0

Configuring KVP metadata

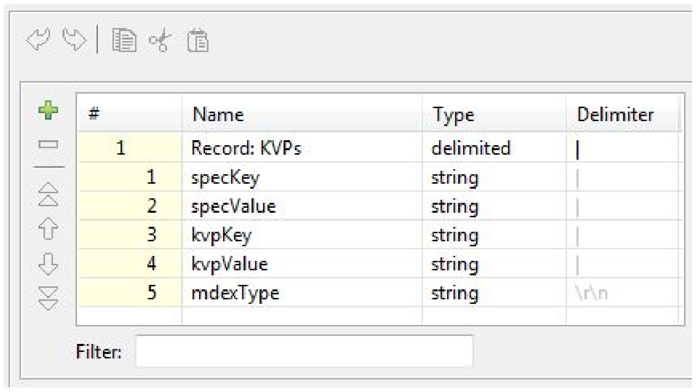
The Edge component must be configured with a Metadata definition for loading the key-value pair data.

This procedure assumes that you have created a graph and added a reader component and the **Add KVPs** connector to it. It also assumes that you have added the key-value pair source file to the project's **data-in** folder.

To configure the Metadata definition for the KVP Edge:

1. In the Palette pane, click **Edge** and use it to connect the reader and the **Add KVPs** connector.
2. Right-click on the Edge and select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
3. In the Flat File dialog, click the **Browse** button, which brings up the **URL Dialog**.
4. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the data delete input file and click **OK**.
5. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**.
6. In the upper pane of the Metadata Editor:
   a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a name such as `KVPs`.
   b) Make sure that the **Type** field of ***all*** properties is set to type **string**. For example if the `specKey` property is set to **integer**, change it to **string**.
   c) Verify that all properties have the correct delimiter character set (which is the pipe character in our example). The final property should have a new-line as the delimiter (\n on Linux and \r\n on Windows).

   At this point, the pane should look like this example:



#	Name	Type	Delimiter
1	Record: KVPs	delimited	|
1	specKey	string	|
2	specValue	string	|
3	kvpKey	string	|
4	kvpValue	string	|
5	mdexType	string	\r\n

Filter:

d) When you have input all your changes, click **Finish**.

7. In the Graph Editor window, right-click on the **Edge** and choose **Select Metadata** and the metadata you have just configured, for example, **KVPs (id:Metadata0)**.
8. Save the graph.

The Metadata definition for the Edge component is now set.

Running the KVPs graph

After creating the graph and configuring the components, you can run the graph to add the key-value pair record assignments to the MDEX Engine.

To run the graph to add key-value pairs to the MDEX Engine:

1. Make sure that you have an MDEX Engine running on the host and port that are configured in the **Add KVPs** connector.
2. Run the graph using one of the run methods.

   For example, you can click the green circle with white triangle icon in the Tool bar:

As the graph runs, the process of the graph execution is listed in the Console Tab. The execution is completed successfully when you see final output similar to this example that adds five key-value pair assignments to the MDEX Engine:

```
INFO  [WatchDog] - Successfully started all nodes in phase!
WARN  [Consumer-0] - Unrecognized assignment type "".  Using "mdex:string"
 instead.
WARN  [Consumer-0] - Unrecognized assignment type "".  Using "mdex:string"
 instead.
WARN  [Consumer-0] - Unrecognized assignment type "".  Using "mdex:string"
 instead.
WARN  [Consumer-0] - Unrecognized assignment type "".  Using "mdex:string"
 instead.
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: 08/06/11 14:30:39
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           5
        0      5       0
INFO  [WatchDog] - Add KVPs               ENDECA_ADD_KVPS0
             FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           In:0           5
        0      5       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
             7146
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 !
```

The example also shows four occurrences of this benign message:

```
Unrecognized assignment type "".  Using "mdex:string" instead.
```

The message is simply informing you that the fifth input schema field (the `mdexType` field) is empty on four of the KVP entries and that the connector will use the `mdex:string` property type when ingesting the data.

Chapter 7

Loading Taxonomies

This chapter describes how to load an externally managed taxonomy (EMT) into the MDEX Engine.

Overview of loading a taxonomy

This chapter will walk you through the various tasks in creating a graph that can load a taxonomy into the MDEX Engine.

The **Add Managed Values** connector allows you to load an externally managed taxonomy (EMT) into the MDEX Engine. When loaded, externally managed taxonomies are added as managed values to a managed attribute. You must create a graph and add the **Add Managed Values** connector and the **UniversalDataReader** component to it.

Keep the following two items in mind when adding a taxonomy:

- Managed values can be added to only one managed attribute in a taxonomy load operation. That is, you can specify the name of only one managed attribute in the **Add Managed Values** connector. This means that all the managed values in the taxonomy input file will be added to the same managed attribute.
- If the managed attribute (to which the taxonomy is being added) does not exist in the MDEX Engine, it will be created automatically by the Data Ingest Web Service. That is, the appropriate PDR and DDR for the managed attribute will be created with system default values. For these default values, see Chapter 1 in this guide.

For the procedure documented in this chapter, the definitions of the managed values to be added are in a flat file. However, the definitions can use other formats that are supported by the LDI reader components. The format of the source data is explained in a following topic.

Format of the taxonomy input file

The input must contain four mandatory configuration properties and a corresponding set of managed value data.

The first line of a taxonomy input file must have these managed value header properties, and in this order:

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

The meanings of these header properties are as follows:

Property	Purpose
`spec`	A unique string identifier for the managed value. This is the managed value spec.
`displayname`	The name for the managed value.
`parent`	Specifies 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. You can add synonyms to a managed value so that users can search for other text strings and still get the same records as a search for the original managed value name. Synonyms can be added to both root and child managed values. If you add multiple synonyms for a managed value, the synonyms are separated by a delimiter that you specify in the configuration of the **Add Managed Values** connector.

The second and following lines contain managed value data for the managed value properties, as illustrated by this sample file:

```
spec|parent|displayname|synonym
White|White Wines|/|Blanc,Blanco,Weisse
Red|Red Wines|/|Rouge,Tinto,Rotwein
Merlot|Merlot Wines|Red
```

In this simple example, the Red managed value (with a display name of "Red Wines") and the White managed value (with a display name of "White Wines") are at the root of the WineType managed attribute, while the Merlot managed value is a child of the Red managed value.

After creating the input file, you can copy it into the **data-in** folder in the Navigation pane of the project.

Creating a graph for the taxonomy

This task describes how to create an empty graph for loading a taxonomy.

The only prerequisite for this task is that you must have created a Data Integrator Designer project. Keep in mind that a project can have multiple graphs, which means that you can create this graph in an existing project.

To create an empty graph for your taxonomy:

1. In the Navigator pane, right-click the **graph** folder.
2. Select **New** > **ETL Graph**.
   The **Create new graph** dialog is displayed.
3. In the **Create new graph** dialog:
   a) Type in the name of the graph, such as **LoadTaxonomy**.
   b) Optionally, type in a description.
   c) You can leave the **Allow inclusion of parameters from external file** box checked.
   d) Click **Next** when you finish.
4. In the **Output** dialog, click **Finish**.

Adding components to the taxonomy graph

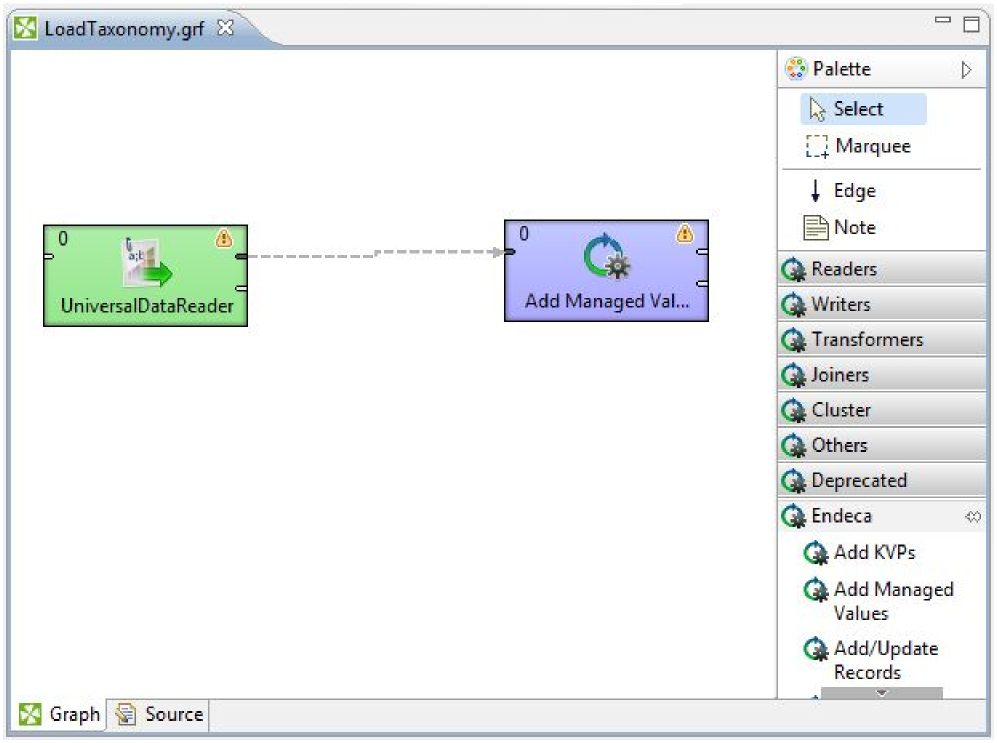
The process requires that you add the **UniversalDataReader** component and the **Add Managed Values** connector to the graph.

In addition, an Edge component will be added to connect the two components.

To add components to the graph:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Palette pane, open the **Latitude** section and drag the **Add Managed Values** connector into the Graph Editor.
3. In the Palette pane, click **Edge** and use it to connect the two components.
4. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the two connected components should look like this:



The next tasks are to configure these components.

Configuring the Reader for the taxonomy input file

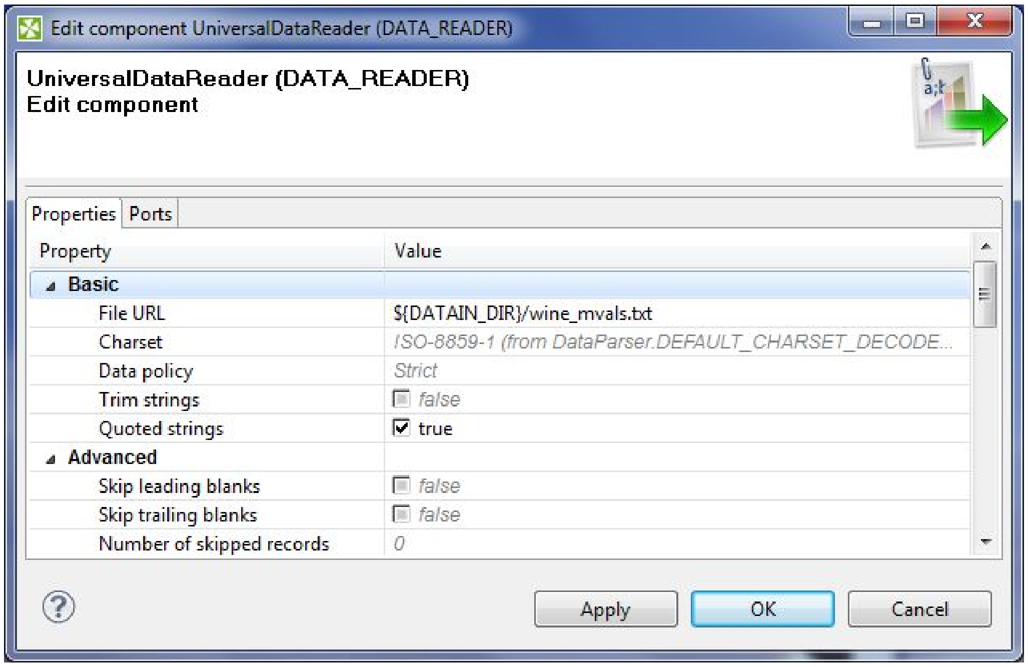
This task describes how to configure the **UniversalDataReader** component to read in the taxonomy data.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the taxonomy source file to the project's **data-in** folder.

To configure the **UniversalDataReader** component for the taxonomy input file:

1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
2. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the taxonomy input file and click **OK**.
3. Check the **Quoted strings** box so that its value changes to `true`.
4. Leave the **Number of skipped records** field set to the default of 0.
5. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
6. Save the graph.

After the component is configured, the Reader Edit Component dialog should look like this example:



Edit component UniversalDataReader (DATA_READER)

UniversalDataReader (DATA_READER)
Edit component

Property	Value
Basic	
File URL	${DATAIN_DIR}/wine_mvals.txt
Charset	ISO-8859-1 (from DataParser.DEFAULT_CHARSET_DECODE...
Data policy	Strict
Trim strings	false
Quoted strings	✓ true
Advanced	
Skip leading blanks	false
Skip trailing blanks	false
Number of skipped records	0

The next task is to configure the **Add Managed Values** connector.

Configuring the Add Managed Values connector

You must configure the **Add Managed Values** component with the location and port of the MDEX Engine, as well as the managed attribute name.

This procedure assumes that you have created a graph and added the **Add Managed Values** connector.

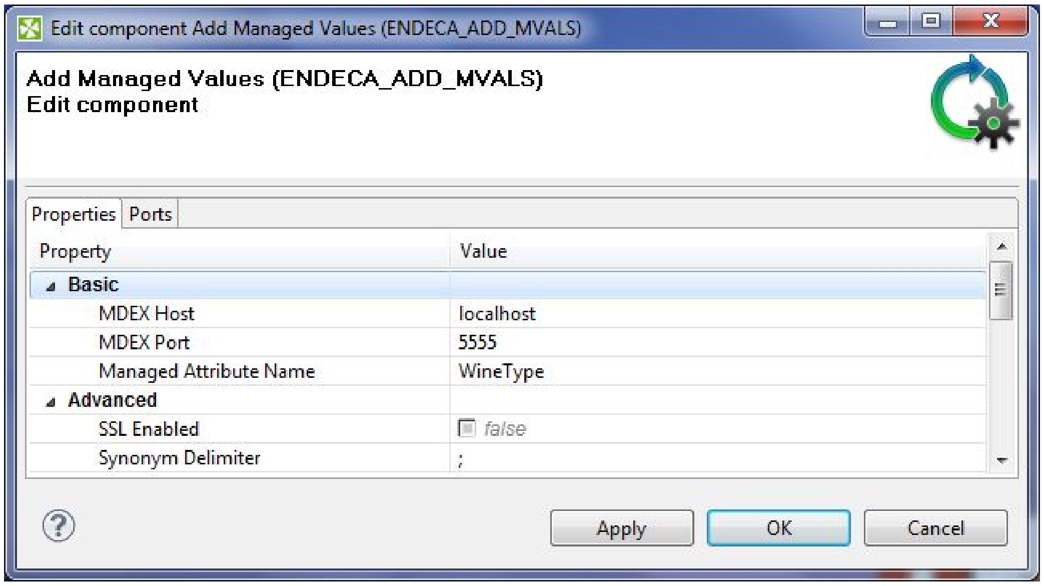
To configure the **Add Managed Values** connector:

1. In the Graph window, double-click the **Add Managed Values** connector.
   The Writer Edit Component dialog is displayed.
2. In the Writer Edit Component dialog, enter these settings:

a) **MDEX Host**: The host name of the machine on which the MDEX Engine is running.
b) **MDEX Port**: The port on which the MDEX Engine is listening for requests.
c) **Managed Attribute Name**: The name of the dimension to which the dimension values will be added.
d) **SSL Enabled**: Toggle this field to `true` if the MDEX Engine is SSL-enabled.
e) **Synonym Delimiter**: Optionally, you can specify the character that separates multiple synonyms for a managed value. Keep in mind that this delimiter is different from the delimiter that separates the property fields.

3. When you have input all your changes, click **OK**.
4. Save the graph.

After configuration, the Writer Edit Component dialog should look like this example:


Edit component Add Managed Values (ENDECA_ADD_MVALS)
Add Managed Values (ENDECA_ADD_MVALS)
Edit component
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Managed Attribute Name	WineType
Advanced	
SSL Enabled	false
Synonym Delimiter	;

In this sample **Add Managed Values** connector, the managed values will be added to the WineType managed attribute.

Configuring taxonomy metadata

The Edge component must be configured with a Metadata definition for loading the taxonomy.

The prerequisite for this task is that an Edge component must exist in the graph.

To configure the Metadata definition for the taxonomy Edge:

1. In the **Outline** pane, right-click on **Metadata**.
2. Select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
3. In the Flat File dialog, click the **Browse** button, which brings up the **URL Dialog**.
4. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.

   d) Select the taxonomy source file and click **OK**.

5. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**. The taxonomy data is loaded into the Metadata Editor.
6. In the middle pane of the Metadata Editor:
   a) Check the **Extract names** box.
   b) Click **Reparse**.
   c) Click **Yes** in the Warning message.
7. In the upper pane of the Metadata Editor:
   a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a name such as `Mvals`.
   b) Make sure that the **Type** field of ***all*** the properties is set to type **string**. For example if the spec property is set to **long**, change it to **string**.
   c) Verify that all properties have the correct delimiter character set (which is the pipe character in our example). The final property should have a new-line as the delimiter (\n on Linux and \r\n on Windows).
   d) When you have input all your changes, click **Finish**.
8. In the Graph Editor window, right-click on the Edge between the **UniversalDataReader** component and the **Add Managed Values** connector. Choose **Select Metadata** and the metadata you have just configured, for example, **Mvals (id:Metadata0)**.
9. Save the graph.

The Metadata definition for the Edge component is now set.

Running the taxonomy graph

After creating the graph and configuring the components, you can run the graph to send the taxonomy to the MDEX Engine.

To run the graph to load a taxonomy:

1. Make sure that you have an MDEX Engine running on the host and port that are configured in the **Add Managed Values** connector.
2. Run the graph using one of the run methods.

   For example, you can click the green circle with white triangle icon in the Tool bar:

As the graph runs, the process of the graph execution is listed in the Console Tab. The execution is completed successfully when you see final output similar to this example of adding the three managed values:

```
INFO  [WatchDog] - ----------------------** Final tracking Log for phase
[0] **---------------------
INFO  [WatchDog] - Time: 24/05/11 17:58:57
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          69
        3     69       3
INFO  [WatchDog] - Add Managed Values     ENDECA_ADD_MVALS0
```

```
             FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           In:0          69
        3     69       3
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
             6126
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 !
```

As the example shows, the Final Tracking Log lists the number of records that were read in by the **UniversalDataReader** component and the number of records (managed values) that were sent to the MDEX Engine by the **Add Managed Values** connector.

Chapter 8

Deleting Data

This chapter describes how to delete records from the MDEX Engine data set. It also describes how to key/value pairs from individual records.

Format of the delete input file

The format of the delete input file uses a fixed schema and a specific ordering of the input fields.

The **Delete Data** connector can perform the following deletions of data in the MDEX Engine:

- Delete a full record.
- Delete a specific key/value pair from a record. All other key/value pairs on the record are not affected.
- Delete all key/value pairs (from the same standard attribute) from a record. This is a wildcard delete of the values from a specific standard attribute on the record. All other key/value pairs (on the record) from other standard attributes are not affected.

You can specify all three types of delete operations in the same input file.

The two restrictions of this connector are:

- It cannot delete managed values on the record.
- When deleting records, it cannot do wildcard deletes (for example, delete Records 50*) and it cannot delete ranges of records (for example, delete Records 5000 to 5100). You must specify each record explicitly by its primary key.

The format of the input file is fixed and uses a specific ordering:

- The first row of the input file is the record header row and must use a fixed schema.
- The second and following lines specify information about the records and/or record data to be deleted.

The schema of the record header row is:

```
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 attribute to which the assignment belongs. If both *kvpKey* and *kvpValue* are blank, the entire record is deleted.

- *kvpValue* is the assigned value to be removed. If this field is blank but *kvpKey* is not, then all assignments of *kvpKey* are deleted.

An example of a text input file for the **Delete Data** connector is:

```
specKey|specValue|kvpKey|kvpValue
WineID|3000|Flavors|peach
WineID|4000|Drinkability|
WineID|5000||
```

When the connector is run with this input file:

- The assignment "peach" from the Flavors standard attribute is removed from Record 3000.
- All assignments from the Drinkability standard attribute are removed from Record 4000.
- Record 5000 is deleted from the MDEX Engine.

After creating the input file, you should add it to the project's **data-in** folder.

Adding components to the delete data graph

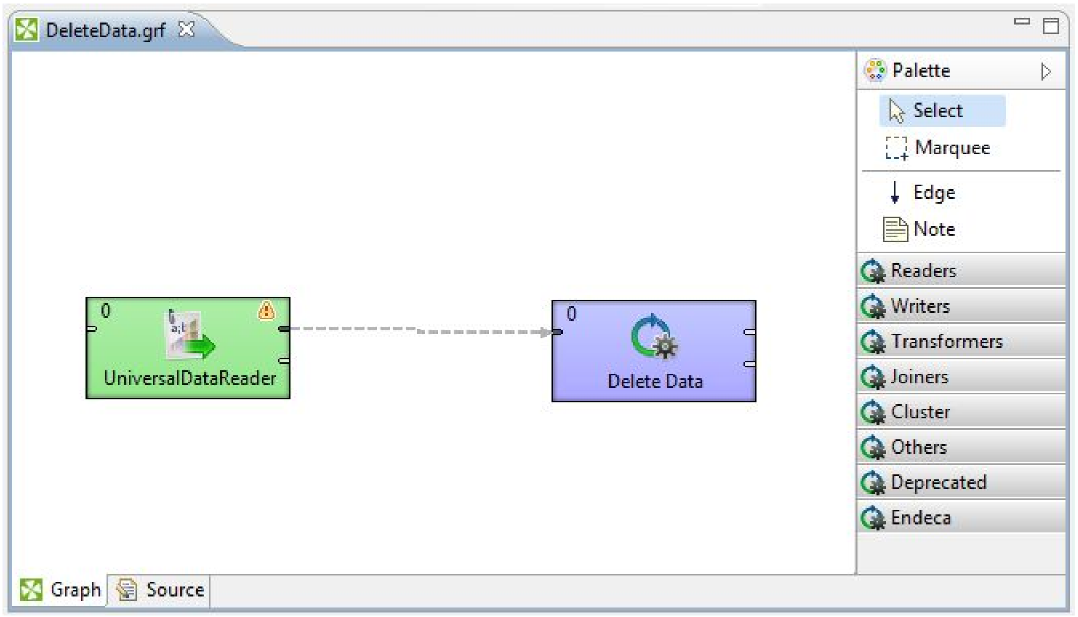
Building a graph to delete data requires that you add the **Delete Data** connector to the graph.

This procedure assumes that you have added the delete input file to the **data-in** folder of the project and have also created an empty graph.

To add components to a graph for deleting records:

1. In the Palette pane, open the **Readers** section and drag the **UniversalDataReader** component into the Graph Editor.
2. In the Palette pane, open the **Latitude** section and drag the **Delete Data** connector into the Graph Editor.
3. In the Palette pane, click **Edge** and use it to connect the two components.
4. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the two connected components should look like this:



The next tasks are to configure the components.

Configuring the Reader for the delete input file

This task describes how to configure the **UniversalDataReader** component to read in the file that specifies what record data to delete.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the delete input file to the project's **data-in** folder.

To configure the **UniversalDataReader** component for the data delete input file:

1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
2. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the data delete input file and click **OK**.
3. Check the **Quoted strings** box so that its value changes to `true`.
4. Leave the **Number of skipped records** field set to the default of 0.
5. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
6. Save the graph.

Configuring the metadata for data deletes

The Edge component must be configured with a Metadata definition.

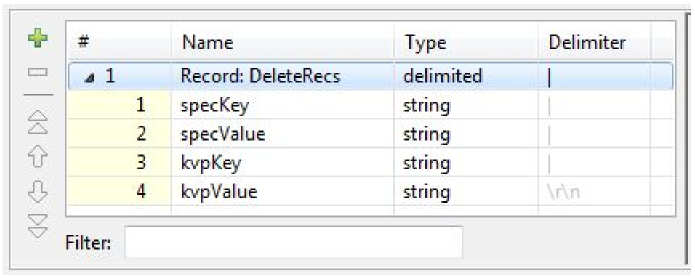
The prerequisite for this task is that an Edge component must exist in the graph.

To configure the Metadata definition for the data delete Edge:

1. Right-click on the Edge and select **New metadata** > **Extract from flat file**.
   The Flat File dialog is displayed.
2. In the Flat File dialog, click the **Browse** button, which brings up the **URL Dialog**.
3. For the **File URL** property:
   a) Click inside its Value field, which displays a **...** browse button.
   b) Click the browse button.
   c) Click the **Workspace view** tab and then double-click the **data-in** folder.
   d) Select the data delete input file and click **OK**.
4. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**.
5. In the middle pane of the Metadata Editor:
   a) Check the **Extract names** box.
   b) Click **Reparse**.
   c) Click **Yes** in the Warning message.
6. In the upper pane of the Metadata Editor:

a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a name such as `DeleteRecs`.
b) Make sure that the **Type** field of ***all*** properties is set to type **string**. For example if the `specKey` property is set to **integer**, change it to **string**.
c) Verify that all properties have the correct delimiter character set (which is the pipe character in our example). The final property should have a new-line as the delimiter (\n on Linux and \r\n on Windows).

At this point, the pane should look like this example:



#	Name	Type	Delimiter
1	Record: DeleteRecs	delimited	|
1	specKey	string	|
2	specValue	string	|
3	kvpKey	string	|
4	kvpValue	string	\r\n

Filter:

d) When you have input all your changes, click **Finish**.

7. Save the graph.

The Metadata definition for the Edge component is now set.

Configuring the Delete Data connector

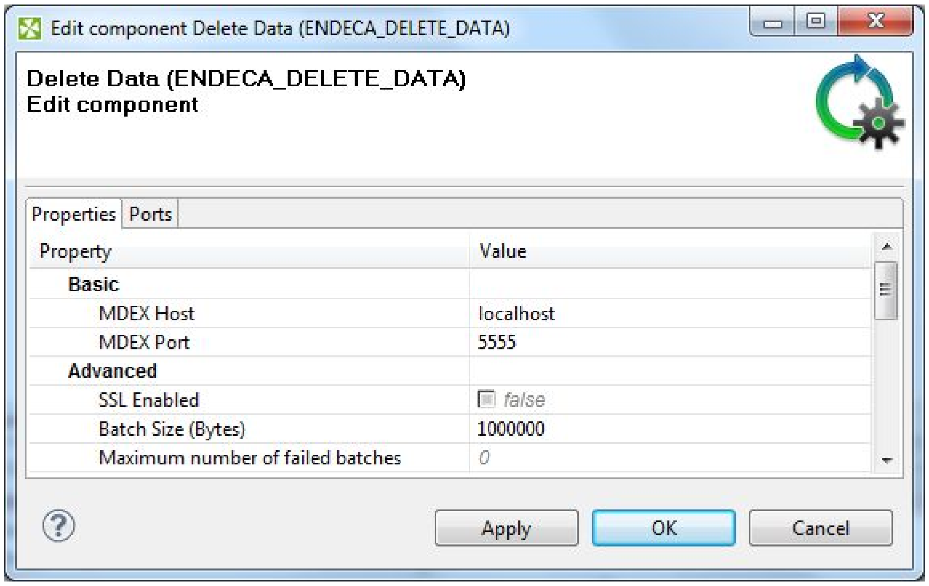
You must configure the **Delete Data** component with the location and port of the MDEX Engine.

This procedure assumes that you have created a graph and added the **Delete Data** connector.

To configure the **Delete Data** connector:

1. In the Graph window, double-click the **Delete Data** component.
   The Writer Edit Component dialog is displayed.
2. In the Writer Edit Component dialog, enter these settings:
   a) **MDEX Host**: The host name of the machine on which the MDEX Engine is running.
   b) **MDEX Port**: The port on which the MDEX Engine is listening for requests.
   c) **SSL Enabled**: Toggle this field to `true` if the MDEX Engine is SSL-enabled.
   d) **Batch Size (Bytes)** : Enter an integer greater than 0 to set the batch size in bytes. Specifying 0 or a negative number will disable batching.
   e) **Maximum number of failed batches**: Enter a positive integer that sets the maximum number of batches that can fail before the operation is ended. Entering 0 allows no failed batches.
3. When you have input all your changes, click **OK**.
4. Save the graph.

After configuration, the Writer Edit Component dialog should look like this example:



Delete Data (ENDECA_DELETE_DATA)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Advanced	
SSL Enabled	☐ false
Batch Size (Bytes)	1000000
Maximum number of failed batches	0

Running the delete data graph

After creating the graph and configuring the components, you can run the graph to delete the specified records and/or record assignments from the MDEX Engine.

To run the graph to delete data from the MDEX Engine:

1. Make sure that you have an MDEX Engine running on the host and port that are configured in the **Delete Data** connector.
2. Run the graph using one of the run methods.

   For example, you can click the green circle with white triangle icon in the Tool bar:

As the graph runs, the process of the graph execution is listed in the Console Tab. The execution is completed successfully when you see final output similar to this example of deleting three records and/or record assignments:

```
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: 27/05/11 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 !
```

As the example shows, the Final Tracking Log lists the number of records that were read in by the **UniversalDataReader** component and the number of records that were sent to the MDEX Engine by the **Delete Data** connector.

Chapter 9

Latitude Connector Reference

This chapter provides a reference for the Endeca Latitude connectors available in the LDI Designer palette.

Bulk Add/Replace Records connector

This connector adds new records or replaces existing records in the MDEX Engine.

The **Bulk Add/Replace Records** connector adds or replaces records via the MDEX Engine's bulk ingest interface (that is, it does not use the Data Ingest Web Service).

The characteristics of this connector are:

- The connector can load data source records only.
- Existing records in the MDEX Engine are replaced, not updated. That is, the replace operation is not additive. Therefore, the key/value pair list of the incoming record will completely replace the key/value pair list of the existing record.
- The connector cannot load PDRs, DDRs, managed attribute values, the GCR, nor the MDEX Engine index configuration files.
- A primary-key attribute (also called the record spec) is required for each record to be added or replaced.
- If an assignment is for a standard attribute (property) that does not exist in the MDEX Engine, the new standard attribute is automatically created with system default values for the PDR (see Chapter 1 in this guide for a list of these values).
- No client-side batching is used and there is only a single, streaming connection to the MDEX Engine.

When added to a graph, the connector icon looks like this:

Metadata schema

The metadata schema for the **Bulk Add/Replace Records** connector is not fixed. Therefore, each LDI field represents a property on an MDEX record.

The metadata type of the LDI field (as shown in the LDI Metadata Editor) translates to the `mdex` property type. For example, the LDI `integer` data type translates to the `mdex:int` data type. Note that this behavior can be overridden to support LDI non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`).

Use cases

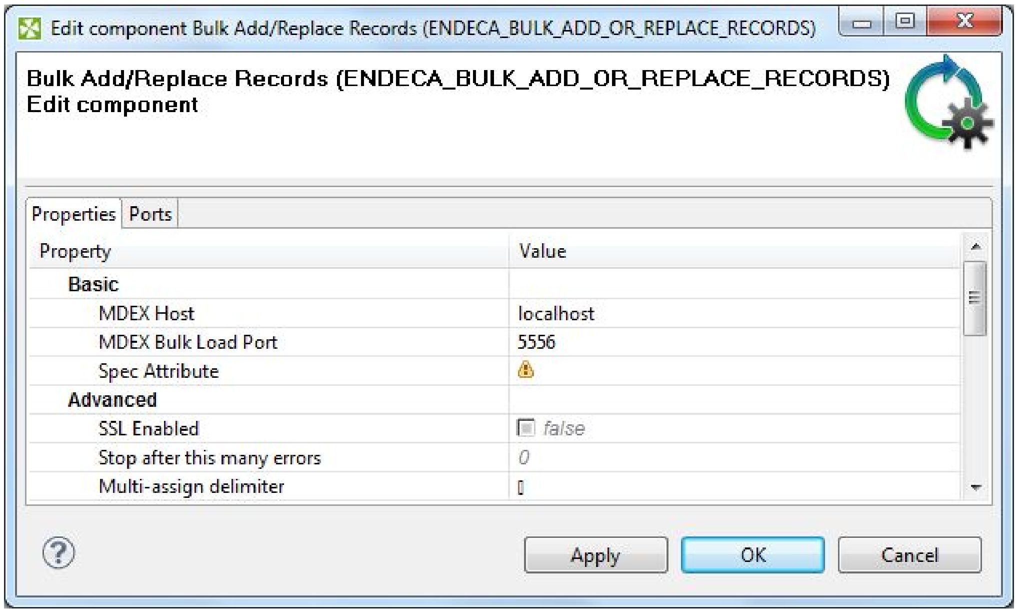
The **Bulk Add/Replace Records** connector is intended to be used with bulk data when delayed update visibility and compromised concurrent query performance are acceptable.

Some of the use cases for this connector are:

- Full index initial load of records, with no loaded schema. In this scenario, the MDEX Engine has no data records and also has no user-created schema (such as no existing PDRs). In this case, all new properties (including the primary-key properties) are created by DIWS with system default values (see Chapter 1 in this guide for a list of these values).
- Full index initial load of records, with your record schema already loaded. You can load the record schema (PDRs and DDRs) with the **Add/Update Records** connector.
- Adding more new records to the MDEX Engine any time after the initial loading of records. As in the initial load case, new standard attributes that do not exist in the MDEX Engine are automatically created with default system values.
- Replacing existing records in the MDEX Engine 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

The configuration for the **Bulk Add/Replace Records** connector is set via the Designer Edit component:


Edit component Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)

Bulk Add/Replace Records (ENDECA_BULK_ADD_OR_REPLACE_RECORDS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Bulk Load Port	5556
Spec Attribute	⚠
Advanced	
SSL Enabled	false
Stop after this many errors	0
Multi-assign delimiter	[]

The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see the "Visual and Common configuration properties" topic in this chapter.

Configuration Property	Purpose	Valid Values
MDEX Host	Identifies the machine on which the MDEX Engine is running.	The name or IP address of the machine. `localhost` can be used as the name.

Configuration Property	Purpose	Valid Values
MDEX Bulk Load Port	Identifies the bulk load port on which the MDEX Engine is listening. Note that this port is different from the HTTP port used by the all the other connectors.	The bulk load port is determined in one of two ways: • The Dgraph `--bulk_load_port` flag is used when the MDEX Engine is started. • If `--bulk_load_port` flag is not used, then the default bulk load port is the standard Dgraph port plus one. This means that the bulk load port is either 5556 (if the Dgraph `--port` flag is not used) or is the value of the `--port` flag plus one.
Spec Attribute	Sets the primary key (record spec) for the records to be added or updated.	The name of the primary key. If the primary-key property does not exist in the MDEX Engine, the property is automatically created with the system default values.
SSL Enabled	Enables or disables SSL for the connector.	• If `false` (the default), SSL is disabled. • If `true`, SSL is used for connections to the MDEX Engine. In this case, the MDEX Engine must also be SSL-enabled.
Stop after this many errors	Sets the maximum number of ingest errors that can occur. The ingest operation is ended after this number of errors is reached.	Either `0` (which means no failures are allowed) or a positive integer.
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.	A single character that is the multi-assign delimiter. You do not have to use this field if your source does not have multi-assign properties.

MDEX Engine 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 in the MDEX Engine. Although the MDEX Engine may accept queries on the ingested records, you should consider the MDEX Engine to be in an inconsistent state.

Add/Update Records connector

This connector adds new records or updates existing records in the MDEX Engine.

The **Add/Update Records** connector adds or updates records via the Data Ingest Web Service (DIWS).

The characteristics of this connector are:

- The connector can load data source records, PDRs (Property Description Records), and DDRs (Dimension Description Records).
- The connector cannot load managed attribute values, the GCR (Global Configuration Record), nor the MDEX Engine index configuration files (such as the search interface configuration).
- A primary-key attribute (also called the record spec) is required for each record to be added or updated.
- If an assignment is for a standard attribute that does not exist in the MDEX Engine, the new standard attribute is automatically created with system default values for the PDR (see Chapter 1 for a list of these values).
- Updates are batched on the client-side with multiple concurrent connections to the MDEX Engine.

When added to a graph, the connector icon looks like this:

Metadata schema

The metadata schema for the **Add/Update Records** connector is not fixed. Therefore, each LDI field represents a property on an MDEX record.

The metadata type of the LDI field (as shown in the LDI Metadata Editor) translates to the `mdex` property type. For example, the LDI `integer` data type translates to the `mdex:int` property type. Note that this behavior can be overridden to support LDI non-native types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`).

Use cases

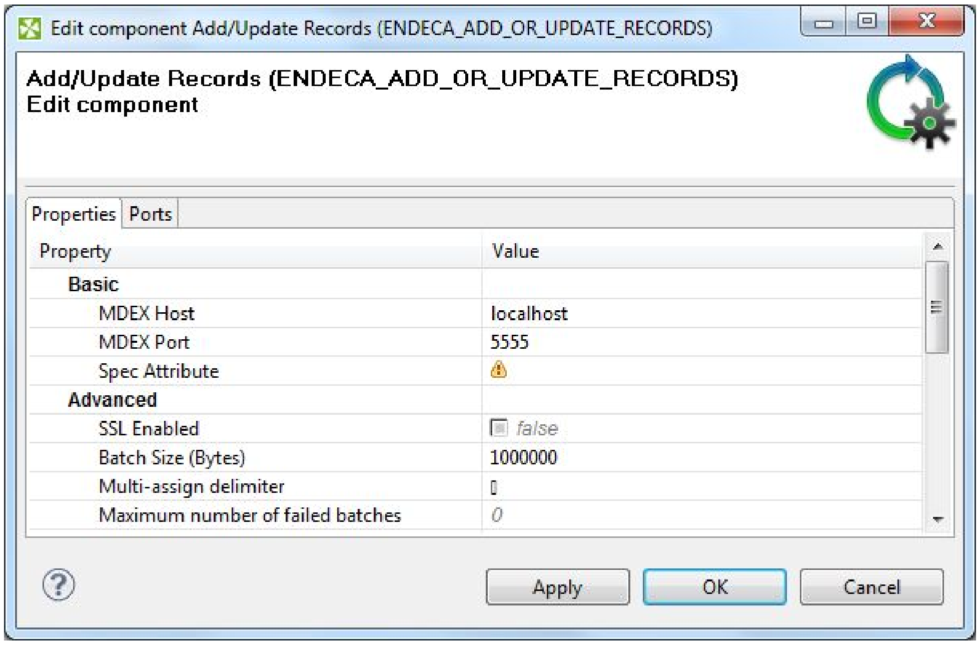
The **Add/Update Records** connector is intended to be used for non-bulk data when immediate update visibility is desired and/or high concurrent query performance is important.

Some of the use cases for this connector are:

- Full index initial load of records, with no loaded schema. In this scenario, the MDEX Engine has no data records and also has no user-created schema (such as no existing PDRs). In this case, all new properties (including the primary-key properties) are created by DIWS with system default values (see the Chapter 1 in this guide for a list of these values).
- Loading of 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, with your record schema already loaded.
- Incremental updates involving the addition of new records to the MDEX Engine any time after the initial loading of records. As in the initial load case, new standard attributes that do not exist in the MDEX Engine are automatically created with default system values.
- Incremental updates to existing records, which means adding key-value pairs. If a standard attribute is configured as multi-assign, a record can have multiple assignments of that attribute. The records to be updated are considered totally additive. That is, the key-value pair list of the update record will be merged into the existing record. If attribute values with the same name already exist, then the new values will be additional values for the same standard attribute (multi-assign). Keep in mind that this operation can also be performed by the **Add KVPs** connector.

Configuration properties

The configuration for the **Add/Update Records** connector is set via the Designer Edit component:


Edit component Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)

Add/Update Records (ENDECA_ADD_OR_UPDATE_RECORDS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Spec Attribute	⚠
Advanced	
SSL Enabled	☐ false
Batch Size (Bytes)	1000000
Multi-assign delimiter	▯
Maximum number of failed batches	0

The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see the "Visual and Common configuration properties" topic in this chapter.

Configuration Property	Purpose	Valid Values
MDEX Host	Identifies the machine on which the MDEX Engine is running.	The name or IP address of the machine. `localhost` can be used as the name.
MDEX Port	Identifies the port on which the MDEX Engine is listening.	The port number on which the MDEX Engine was started.
Spec Attribute	Sets the primary key (record spec) for the records to be added or updated.	The name of the primary key. If the primary-key property does not exist in the MDEX Engine, the property is automatically created with the system default values.
SSL Enabled	Enables or disables SSL for the connector.	• If `false` (the default), SSL is disabled. • If `true`, SSL is used for connections to the MDEX Engine. In this case, the MDEX Engine must also be SSL-enabled.
Batch Size (Bytes)	Sets the batch size for the ingest operation. Each record size is calculated in bytes. A batch consists of one or more records.	• A number equal to or greater than 1 sets the batch size. If the batch size is too small to fit in a record, then it is reset to the size to accommodate that record.

Configuration Property	Purpose	Valid Values
		• Specifying 0 (zero) or a negative number will turn off batching. This means that all records are placed into one batch and sent to the MDEX Engine at the end of the ingest operation.
Multi-assign delimiter	Sets the character that separates multi-assign values in a property in a source record. Keep in mind that this delimiter is different from the delimiter that separates property fields on the source record.	A single character that is the multi-assign delimiter. You do not have to use this field if your source does not have multi-assign properties.
Maximum number of failed batches	Sets the maximum number of batches that can fail before the ingest operation is ended.	Either `0` (which allows no failed batches) or a number greater than `0`.

Batch size adjustments by the connector

Regardless of the batch size you have specified (assuming it is a non-zero, non-negative number), the **Add/Update Records** connector will adjust the batch size on the fly in order to ensure that all the assignments for a given record will fit in the batch. This ensures that assignments for a given record are not split between different batches.

Add KVPs connector

This connector updates Endeca records in the MDEX Engine by adding new key-value pairs to the records.

The **Add KVPs** connector is intended to update records by adding new key-value pair (KVP) assignments to those records. The connector updates records via the Data Ingest Web Service (DIWS).

The characteristics of this connector are:

- The connector can load a new key-value pair for a record.
- Only Endeca standard attribute values can be loaded. Adding managed attribute values is not supported.
- The key-value pairs can only be added. Existing key-value pairs on records cannot be deleted or replaced.
- Multi-assign properties cannot be added. To do this, you need to add separate rows in the input file for multiple assignments of a given property.
- If an assignment is for a standard attribute (property) that does not exist in the MDEX Engine, the new standard attribute is created by DIWS with system default values for the PDR (see Chapter 1 for a list of these values). You can, however, specify a property type for the new standard attribute.
- The main use case is one where your source data is stored in a key-value pair format, as opposed to something like a rectangular data model.

When added to a graph, the connector icon looks like this:

Metadata schema

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

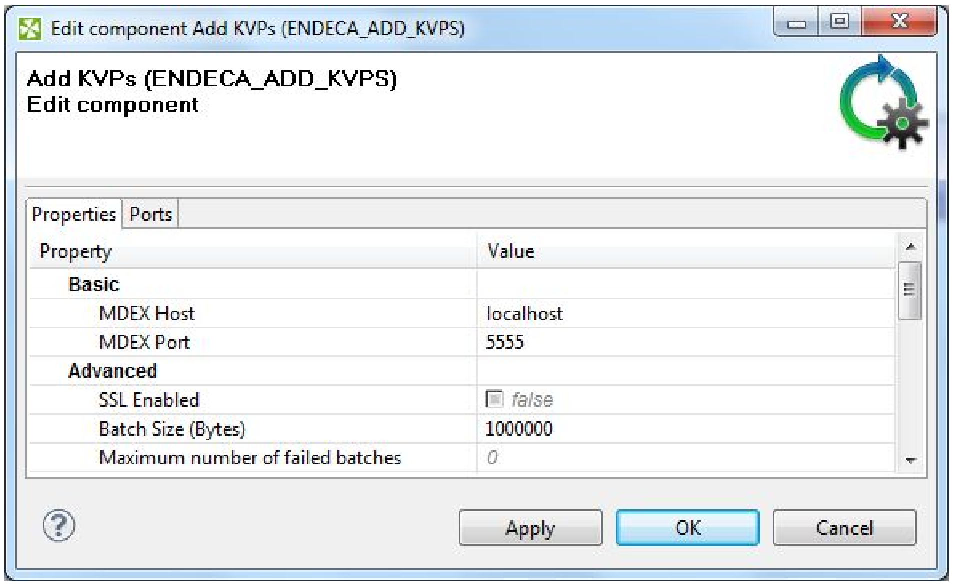
```
specKey|specValue|kvpKey|kvpValue|mdexType
```

where:

- *specKey* is the primary key (record spec) of the record to which the key-value pair will be added.
- *specValue* is the value of the record's primary key.
- *kvpKey* is the name (key) of the Endeca standard attribute to be added to the record. If the standard attribute does not exist in the MDEX Engine, it is automatically created by DIWS with system default values.
- *kvpValue* is the value of the standard attribute to be added.
- *mdexType* specifies the `mdex` property type (such as `mdex:int` or `mdex:dateTime`). 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 will be `mdex:string`. If the standard attribute already exists, you can specify an empty value for *mdexType*.

Configuration properties

The configuration for the **Add KVPs** connector is set via the Designer Edit component:


Edit component Add KVPs (ENDECA_ADD_KVPS)
Add KVPs (ENDECA_ADD_KVPS)
Edit component
Properties
Ports
Apply
OK
Cancel

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Advanced	
SSL Enabled	☐ false
Batch Size (Bytes)	1000000
Maximum number of failed batches	0

The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
MDEX Host	Identifies the machine on which the MDEX Engine is running.	The name or IP address of the machine. `localhost` can be used as the name.

Configuration Property	Purpose	Valid Values
MDEX Port	Identifies the port on which the MDEX Engine is listening.	The port number on which the MDEX Engine was started.
SSL Enabled	Enables or disables SSL for the connector.	• If `false` (the default), SSL is disabled. • If `true`, SSL is used for connections to the MDEX Engine. In this case, the MDEX Engine must also be SSL-enabled.
Batch Size (Bytes)	Sets the batch size for the ingest operation. Each record size is calculated in bytes. A batch consists of one or more records.	• A number equal to or greater than 1 sets the batch size. If the batch size is too small to fit in a record, then it is reset to the size to accommodate that record. • Specifying 0 (zero) or a negative number will turn off batching. This means that all records are placed into one batch sent to the MDEX Engine at the end of the ingest operation.
Maximum number of failed batches	Sets the maximum number of batches that can fail before the ingest operation is ended.	Either `0` (which allows no failed batches) or a positive integer.

Add Managed Values connector

This connector loads a taxonomy into the MDEX Engine's data set.

The **Add Managed Values** connector is intended to load a taxonomy (Endeca managed attribute values) into the MDEX Engine. The taxonomy is loaded via the Data Ingest Web Service (DIWS).

The characteristics of this connector are:

- The connector 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 MDEX Engine, the managed attribute is created by DIWS with system default values for the DDR and (if does not already exist) for the PDR. See Chapter 1 in this guide for a list of the default values.
- Optionally, synonyms can be created for managed values.

When added to a graph, the connector icon looks like this:

Metadata schema

The metadata schema of the **Add Managed Values** connector 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:

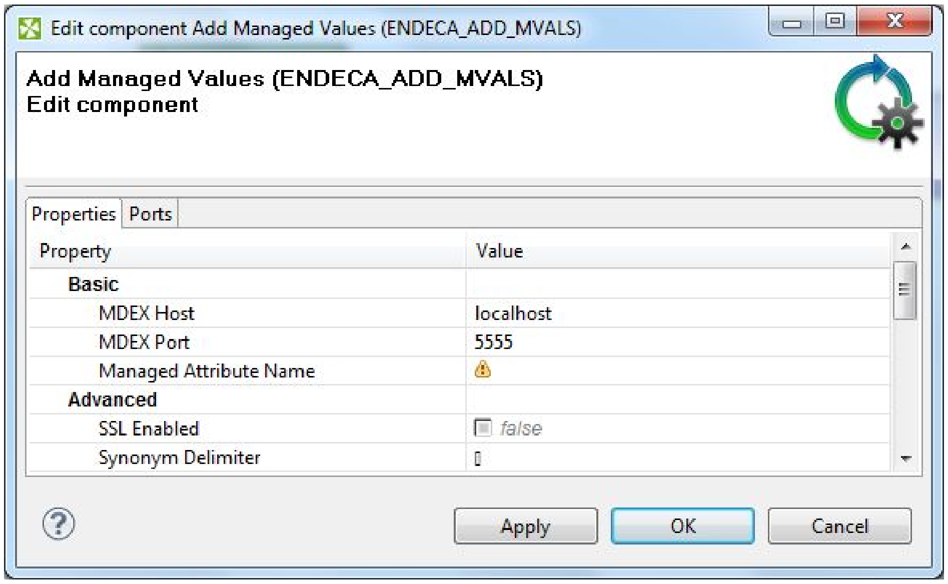
```
spec|displayname|parent|synonym
```

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

The configuration for the **Add Managed Values** connector is set via the Designer Edit component:


Edit component Add Managed Values (ENDECA_ADD_MVALS)

Add Managed Values (ENDECA_ADD_MVALS)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Managed Attribute Name	⚠
Advanced	
SSL Enabled	☐ false
Synonym Delimiter	|

The configuration properties that you can change in the Edit component are:

Configuration Property	Purpose	Valid Values
MDEX Host	Identifies the machine on which the MDEX Engine is running.	The name or IP address of the machine. `localhost` can be used as the name.
MDEX Port	Identifies the port on which the MDEX Engine is listening.	The port number on which the MDEX Engine was started.
Managed Attribute Name	Sets the name of the managed attribute to which the managed values will be added.	The name of a managed attribute. The name must use the NCName format. If the managed attribute does not exist in the MDEX Engine, DIWS automatically creates the managed attribute with system default values.

Configuration Property	Purpose	Valid Values
SSL Enabled	Enables or disables SSL for the connector.	• If `false` (the default), SSL is disabled. • If `true`, SSL is used for connections to the MDEX Engine. In this case, the MDEX Engine must also be SSL-enabled.
Synonym delimiter	Sets the delimiter for specifying multiple synonyms.	A single character that is the synonym delimiter.

Delete Data connector

This connector performs delete operations on Endeca records.

The **Delete Data** connector performs these delete operations via the Data Ingest Web Service (DIWS):

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

Note that the connector cannot remove managed values from records.

When added to a graph, the connector icon looks like this:

Metadata schema

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

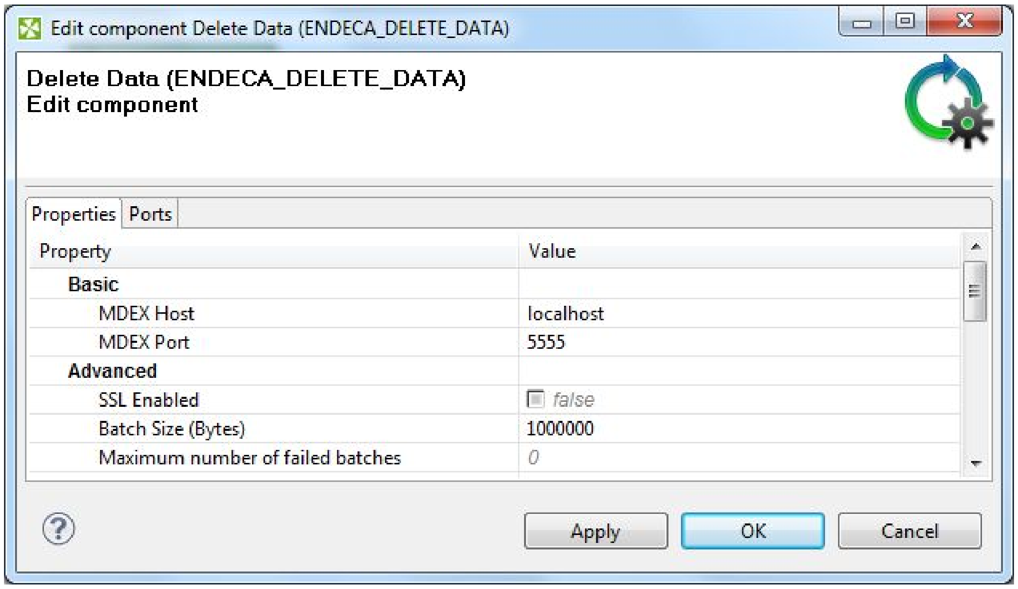
- *specKey* is the name of the primary key (record spec) of the record on which the delete operation will be performed.
- *specValue* is the value of the record's primary key.
- *kvpKey* is the name (key) of the Endeca standard attribute to which the assignment belongs. If *kvpValue* is blank, then all 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** connector:

```
specKey|specValue|kvpKey|kvpValue
WineID|3000|Flavors|peach
WineID|4000|Drinkability|
WineID|5000||
```

Configuration properties

The configuration for the **Delete Data** connector is set via the Designer Edit component:


Edit component Delete Data (ENDECA_DELETE_DATA)

Delete Data (ENDECA_DELETE_DATA)
Edit component

Property	Value
Basic	
MDEX Host	localhost
MDEX Port	5555
Advanced	
SSL Enabled	false
Batch Size (Bytes)	1000000
Maximum number of failed batches	0

The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
MDEX Host	Identifies the machine on which the MDEX Engine is running.	The name or IP address of the machine. `localhost` can be used as the name.
MDEX Port	Identifies the port on which the MDEX Engine is listening.	The port number on which the MDEX Engine was started.
SSL Enabled	Enables or disables SSL for the connector.	• If `false` (the default), SSL is disabled. • If `true`, SSL is used for connections to the MDEX Engine. In this case, the MDEX Engine must also be SSL-enabled.
Batch Size (Bytes)	Sets the batch size for the delete ingest operation. Each record size is calculated in bytes. A batch consists of one or more records to be sent to the MDEX Engine for deletion.	• A number equal to or greater than 1 sets the batch size. If the batch size is too small to fit in a record, then it is reset to the size to accommodate that record. • Specifying 0 (zero) or a negative number will turn off batching. This means that all records are placed into one batch sent to the MDEX Engine at the end of the ingest operation.

Configuration Property	Purpose	Valid Values
Maximum number of failed batches	Sets the maximum number of batches that can fail before the ingest operation is ended.	Either `0` (which allows no failed batches) or a positive integer.

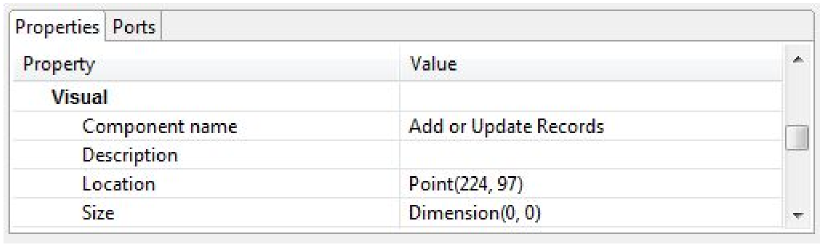
Visual and Common configuration properties

This topic describes the meanings of the **Visual** and **Common** configuration properties of connectors.

All Latitude connectors have **Visual** and **Common** properties in their configuration dialogs. Because the functionality of these properties is the same across all the connectors, an overview of these properties can be described in a common topic. For more information on the purpose of these properties, see the *CloverETL Designer User's Guide*.

Visual properties

Visual properties can be seen in the graph. The **Visual** section looks like this in the Edit component:



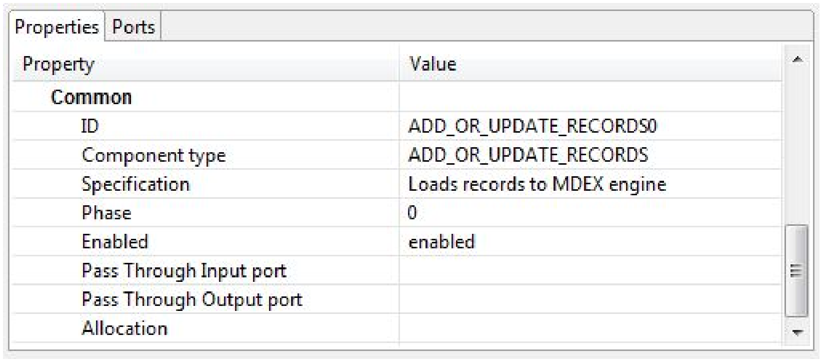
Property	Value
Visual	
Component name	Add or Update Records
Description	
Location	Point(224, 97)
Size	Dimension(0, 0)

The **Visual** configuration properties are:

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 (for example, to one that best describes what type of data is being loaded).
Description	Lets you add some descriptive text that is displayed as a hint when you mouse over the component in the graph.	Text describing what this component does.
Location	Describes the location (using an X-axis and Y-axis) of the component within the graph.	Do not edit this field. Instead, use your cursor to move the component in the graph to the desired position.
Size		Do not edit this field.

Common properties

Common properties are common to all components. The **Common** section looks like this in the Edit component:



Property	Value
Common	
ID	ADD_OR_UPDATE_RECORDS0
Component type	ADD_OR_UPDATE_RECORDS
Specification	Loads records to MDEX engine
Phase	0
Enabled	enabled
Pass Through Input port	
Pass Through Output port	
Allocation	

The **Common** configuration properties are:

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.
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.

Common property	Purpose	Valid values
Allocation	If the graph is executed by a Cluster of LDI Servers, this attribute must be specified in the graph.	

Chapter 10

Configuration Tips and Troubleshooting

This chapter provides some generic configuration tips and also describes solutions to problems you may encounter when building or running graphs.

Configuration tips

This section provides tips and general information for configuration tasks.

Recommended order of loading data

This topic provides a recommended order for loading your configuration information and source data into the MDEX Engine.

Assuming that you are starting with an empty MDEX Engine (that is, only `mkmdex` has been run), the recommended order of loading your data is the following:

1. Global Configuration Record (GCR), which sets the global configuration settings for the MDEX Engine.
2. Attribute Schema Configuration, which creates the standard attributes and managed attributes, in this order:
    a. Property Description Records (PDRs)
    b. Dimension Description Records (DDRs)
    c. Managed attribute values (mvals)
3. Attribute Group Configuration, which consists of creating groups and adding attributes to them
4. Index Configuration, which consists of the index configuration documents in this order:
    a. `relrank_strategies` document (necessary if a relevance ranking strategy is referenced by the next two documents)
    b. `recsearch_config` document
    c. `dimsearch_config` document
    d. `precedence_rules` document
    e. `stop_words` document
    f. `thesaurus` document
5. Application Source Records, which consists of the data on which user queries will be made.

You may alter the order to fit the needs of your Latitude application. For example, if you are satisfied with the default settings of the GCR, then there is no need to load the GCR. Or, to use another example, you do not need to load your attribute group configuration if you intend to create and manage attribute groups with Latitude Studio's Attribute Settings component.

Creating mdexType Custom properties

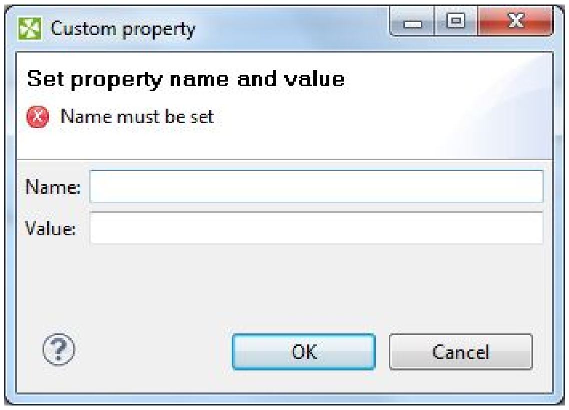
LDI Designer allows you to create an **mdexType** Custom property that you can use to explicitly specify the MDEX type to which a particular Endeca standard attribute should map.

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 Designer. In this case, the ETL developer has to send a string through the Designer, making sure that the string value is formatted in the way that the MDEX Engine expects. The new **mdexType** Custom property, in other words, overrides the Designer native property type when the records are sent to the MDEX Engine.

This functionality is particularly useful for non-String multi-assign properties, because the Designer 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 MDEX Engine with `mdex:int` (for example) as the MDEX property type.

Important: Although the property will be designated as Designer 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 MDEX Engine, then the String value must use the `mdex:duration` format.

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



The Name field must be **mdexType** and the Value field must be one of the MDEX property types (such as `mdex:duration`). The Name and Value are used by the Latitude connector to specify (to the MDEX Engine) 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:

```
WineID|Wine|Duration|Location
95000|Cambridge Chardonnay|P429DT2M3.25S|42.365615 -71.075647
```

It creates only one record with four standard attributes:

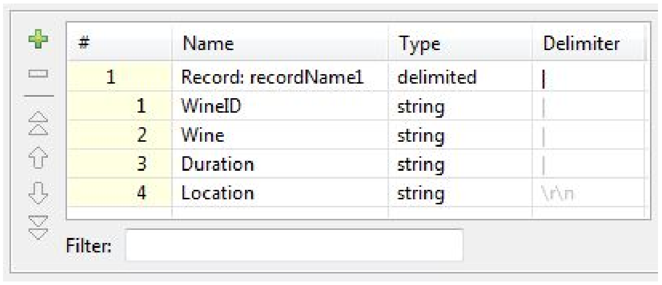
- The WineID attribute is the primary key and is an Integer. Its value is 9500.
- The Wine attribute is a String type with a value of "Cambridge Chardonnay".

- 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 Designer 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, a Latitude connector (such as the **Add/Update Records** connector), 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:

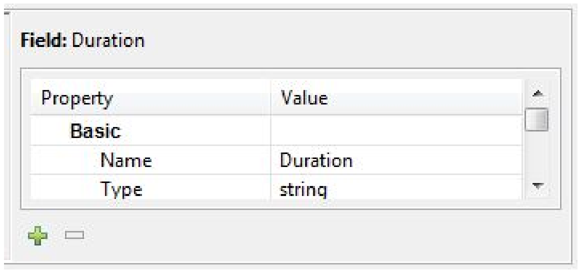


#	Name	Type	Delimiter
1	Record: recordName1	delimited	|
1	WineID	string	|
2	Wine	string	|
3	Duration	string	|
4	Location	string	\r\n

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 WineID Type to **integer**.
   c) Leave the Wine 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:

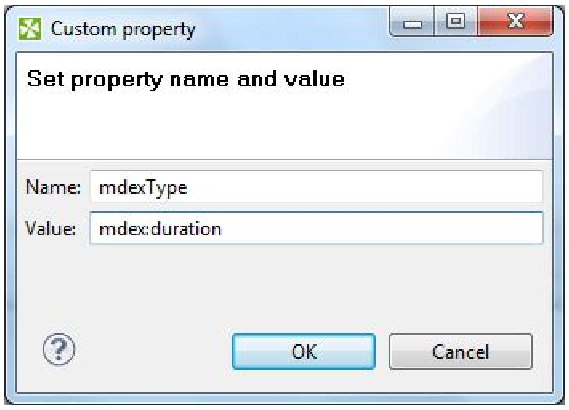


Field: Duration

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 Value field.

      The Custom property editor should look like this:



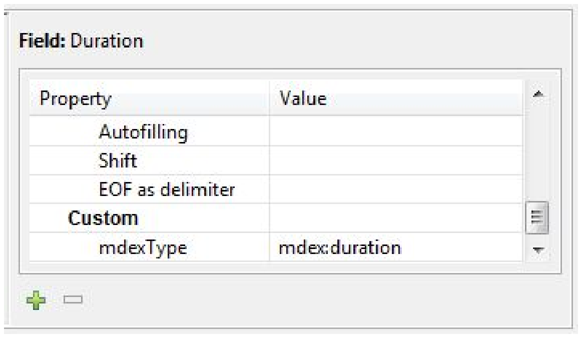
Set property name and value

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 MDEX Engine 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 since the epoch (January 1, 1970).
- `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

Troubleshooting problems

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

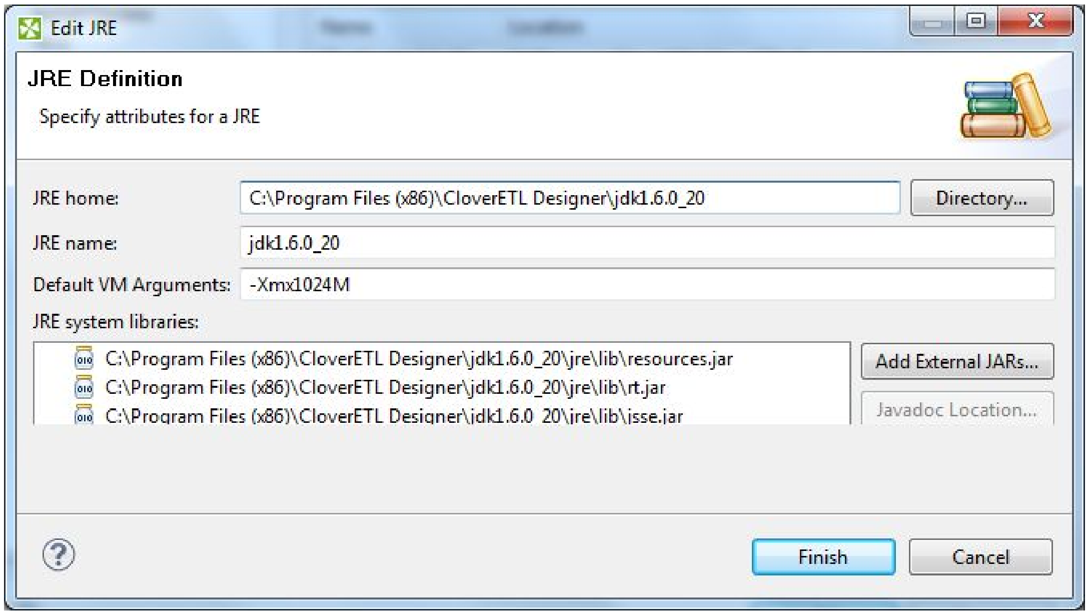
Avoiding OutOfMemory errors

If the Java process has insufficient memory allocated, you may get `OutOfMemory` errors when running the graph.

In an unsuccessful run, the Console Tab will show an `OutOfMemory` error similar to this example:

```
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
```

You can avoid these errors by increasing the memory allocated to the Java process running the service. The **Edit JRE** menu lets you increase the memory size on a global basis.



JRE home: C:\Program Files (x86)\CloverETL Designer\jdk1.6.0_20
JRE name: jdk1.6.0_20
Default VM Arguments: -Xmx1024M
JRE system libraries:
C:\Program Files (x86)\CloverETL Designer\jdk1.6.0_20\jre\lib\resources.jar
C:\Program Files (x86)\CloverETL Designer\jdk1.6.0_20\jre\lib\rt.jar
C:\Program Files (x86)\CloverETL Designer\jdk1.6.0_20\jre\lib\jsse.jar

To avoid `OutOfMemory` errors:

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, specify a Java option to set the heap size, such as `-Xmx1024M`.
   The Edit JRE menu should look like the example above.
5. Click **Finish** to apply your change and close the **Edit JRE** menu.
6. Click **OK** to close the **Preferences** menu.

Avoiding BufferOverflow errors

If the size of the data buffer is too small, you may get `BufferOverflow` errors when running the graph.

In an unsuccessful run, the Console Tab will show a `BufferOverflowException` error similar to this example:

```
ERROR [WatchDog] - Node DATA_READER0 error details:
java.lang.RuntimeException: The size of data buffer is only 12288.
Set appropriate parameter in defautProperties file.
    at org.jetel.data.StringDataField.serialize(StringDataField.java:285)
    at org.jetel.data.DataRecord.serialize(DataRecord.java:466)
    at org.jetel.graph.DirectEdge.writeRecord(DirectEdge.java:234)
    at org.jetel.graph.Edge.writeRecord(Edge.java:371)
    at org.jetel.component.DataReader.execute(DataReader.java:264)
    at org.jetel.graph.Node.run(Node.java:425)
    at java.lang.Thread.run(Thread.java:619)
Caused by: java.nio.BufferOverflowException
    at java.nio.Buffer.nextPutIndex(Buffer.java:501)
    at java.nio.DirectByteBuffer.putChar(DirectByteBuffer.java:465)
    at org.jetel.data.StringDataField.serialize(StringDataField.java:282)
    ... 6 more
```

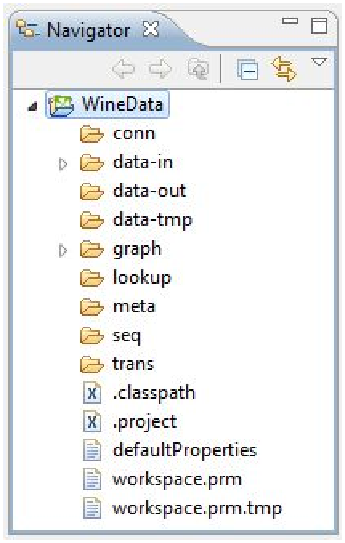
You can avoid these errors by increasing the buffer settings in the `defaultProperties` configuration file, copying the file into your LDI project, and then specifying the file to be used in the run configuration of a graph. The `defaultProperties` configuration file is located in the `cloveretl.engine.jar` JAR file, whose default location is:

```
CloverETL Designer\plugins\com.cloveretl.gui_3.0.1\lib\lib\cloveretl.en¬
gine.jar
```

To modify the `defaultProperties` configuration file and add it to your LDI project:

1. Copy the `cloveretl.engine.jar` JAR file to a temporary location (for example, a `temp` directory).
2. Extract the file `org\jetel\data\defaultProperties` from the JAR file into the `temp` directory.
3. Open the `defaultProperties` in a text editor.
4. Make these changes to the `defaultProperties` file:
   a) Increase `Record.MAX_RECORD_SIZE` to a size such as 65535.
   b) Increase `DataParser.FIELD_BUFFER_LENGTH` to a size such as 8192.
   c) Set `DataFormatter.FIELD_BUFFER_LENGTH` to the same size as `DataParser.FIELD_BUFFER_LENGTH`.
   d) Increase `DEFAULT_INTERNAL_IO_BUFFER_SIZE` to a size such as 130000.

   Note that these are suggested recommendations. Your final settings depend on the characteristics of your data set, which could mean that you may have to further increase these settings if your ingest operations are still failing due to memory problems.
5. Place the `defaultProperties` configuration file in your LDI project folder, by copying it into the Navigator pane.

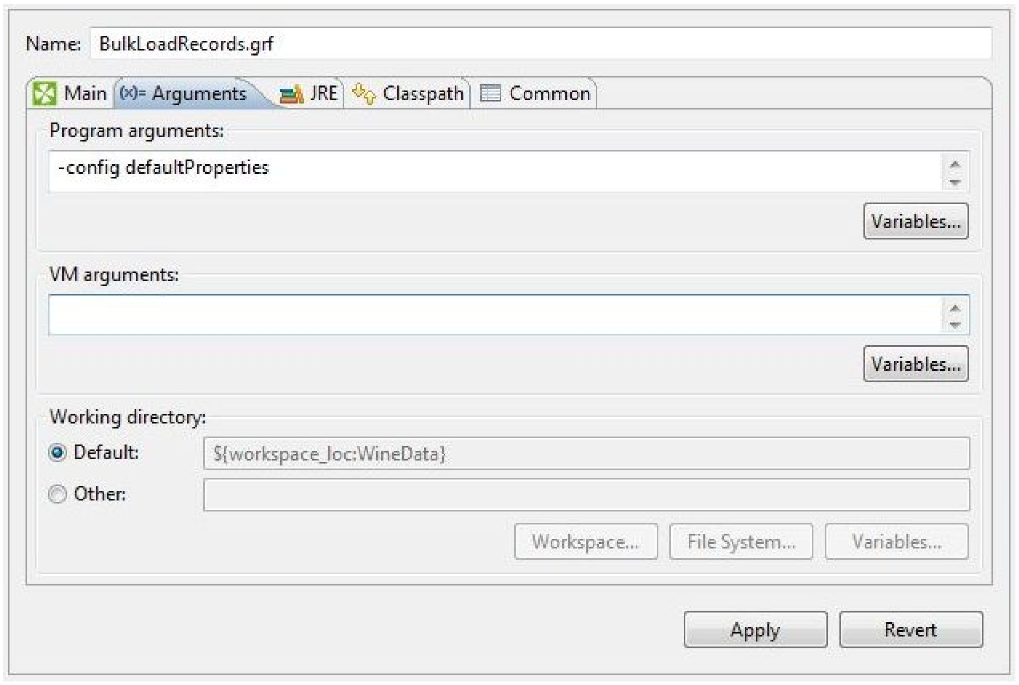


6. From the Designer tool bar, choose **Run** > **Run Configurations**.
7. From the left pane of the Run Configurations menu, select a graph to edit and then click the **Arguments** tab in the run configuration.

   If the graph you want to edit is not listed, you can either run the graph (so that its name will be listed) or create a new configuration for the graph.
8. Enter the following text in the Program arguments field:

   ```
   -config defaultProperties
   ```

   At this point, the Arguments tab should look like this example:



Name: BulkLoadRecords.grf

Program arguments:

-config defaultProperties

VM arguments:

Working directory:

Default: ${workspace_loc:WineData}

9. Click **Apply** to save your changes.
10. Click either Run (to run the graph with the modified run configuration) or **Close** (to close the Run Configurations menu).

Connection errors

This topic illustrates connection errors that may occur between your Endeca connectors and the MDEX Engine.

If the MDEX Engine is not running, this error will result when an Endeca Latitude connector attempts to make a connection to the MDEX Engine:

```
ERROR [ENDECA_ADD_KVPS0_0] - Connection refused: connect Error connecting
to the dgraph.
If applicable, ensure your SSL settings are correct.
ERROR [ENDECA_ADD_KVPS0_0] - Failed with the following exception:
 java.rmi.RemoteException: Connection refused: connect Error connecting to
 the dgraph.
 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
...
ERROR [WatchDog] - !!! Phase finished with error - stopping graph run !!!
```

The error will also occur if the connector is incorrectly configured as to the MDEX Engine's host name and/or port number, or if a connector that is not enabled for SSL attempts to connect to an SSL-enabled MDEX Engine.

Multi-assign delimiter error

A multi-assign delimiter must be specified when loading multi-assign data.

When loading multi-assign attribute data with either the **Bulk Add/Replace Records** connector or the **Add/Update Records** connector, you must remember to specify the multi-assign delimiter character when configuring the connector.

If you do not specify the delimiter (or specify the wrong one), the ingest operation should fail with an error like the following:

```
ERROR [SocketReader] - Received error message from server: Attempt to
   add/replace record WineID:34699 with unknown dimension value
   "Red;Merlot" within dimension "WineType"
ERROR [WatchDog] - Graph execution finished with error
ERROR [WatchDog] - Node ENDECA_BULK_ADD_OR_REPLACE_RECORDS0 finished
   with status: ERROR
```

In this example, the multi-assign source is "Red;Merlot" (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 connector's configuration screen.

Appendix A

MDEX Engine Configuration XML Reference

This reference describes the XML elements in the MDEX Engine configuration documents. The reference describes each element's format, attributes, and sub-elements, and provides an example of its usage.

XML elements

These common elements are available for use in multiple Endeca XML files.

COMMENT

The COMMENT element associates a comment with a pipeline component and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.

Format

```
<!ELEMENT COMMENT (#PCDATA)>
```

Attributes

The COMMENT element has no attributes.

Sub-elements

The COMMENT element has no sub-elements.

Example

This example includes an informational comment.

```
<DIMSEARCH_CONFIG FILTER_FOR_ANCESTORS="FALSE"
  <COMMENT>Displays ancestor managed values.</COMMENT>
/DIMSEARCH_CONFIG>
```

DIMNAME

The DIMNAME element specifies the name of a managed attribute.

Format

```
<!ELEMENT DIMNAME (#PCDATA)>
```

Attributes

The DIMNAME element has no attributes.

Sub-elements

The DIMNAME element has no sub-elements.

Example

This example shows the name of a managed attribute.

```
<RECORD>
   <DIMNAME="WineType">
   ...
</RECORD>
```

PROP

The PROP element represents an Endeca standard attribute. it can optionally contain a PVAL element.

Format

```
<!ELEMENT PROP (PVAL?)>
<!ATTLIST PROP
 NAME CDATA  #REQUIRED
>
```

Attributes

The PROP element has the following attributes.

NAME

Identifies the name of the standard attribute.

Sub-elements

The PROP element can optionally contain a PVAL element (or it can have no PVAL elements).

Example

This example shows a standard attribute name.

```
<RECORD>
   <PROP NAME="Endeca.Title">
      <PVAL>The Simpsons Archive</PVAL>
   </PROP>
   ...
</RECORD>
```

PROPNAME

The PROPNAME element represents an Endeca standard attribute.

Format

```
<!ELEMENT PROPNAME (#PCDATA)>
```

Attributes

The PROPNAME element has no attributes.

Sub-elements

The PROPNAME element has no sub-elements.

Example

This example shows a standard attribute name.

```
<RECORD>
   <PROPNAME="P_Price">
   ...
</RECORD>
```

PVAL

The PVAL element represents a standard attribute value.

Format

```
<!ELEMENT PVAL (#PCDATA)>
```

Attributes

The PVAL element has no attributes.

Sub-elements

The PVAL element has no sub-elements.

Example

This example shows a standard attribute value.

```
<PROP NAME="Endeca.Title">
   <PVAL>The Simpsons Archive</PVAL>
</PROP>
```

Dimsearch_config elements

The Dimsearch_config element controls how value searches behave.

This file configures search matching, spelling correction, filtering, and relevance ranking for value search. These options are configured in the file's root element DIMSEARCH_CONFIG.

DIMSEARCH_CONFIG

A DIMSEARCH_CONFIG element sets up the configuration of standard and managed attributes for value searches. Value searches search against the text collection that consists of the names of all the attribute values in the data set.

Format

```
<!ELEMENT DIMSEARCH_CONFIG (COMMENT?, PARTIAL_MATCH?, AUTO_SUGGEST?)>
<!ATTLIST DIMSEARCH_CONFIG
   FILTER_FOR_ANCESTORS     (TRUE | FALSE)   "FALSE"
   RELRANK_STRATEGY         CDATA            #IMPLIED
>
```

Attributes

The DIMSEARCH_CONFIG element has the following attributes.

FILTER_FOR_ANCESTORS

When set to TRUE, the results of a value search return only the highest ancestor attribute value. This means that if both *red zinfandel* and *red wine* match a search query for "red" and FILTER_FOR_ANCESTORS is set to true, only the red wine attribute value is returned. When set to FALSE, then both attribute values are returned. The default value is FALSE.

RELRANK_STRATEGY

Specifies the name of a relevance ranking strategy for value search.

Sub-elements

The following table provides a brief overview of the DIMSEARCH_CONFIG sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
PARTIAL_MATCH	Specifies if partial query matches should be supported for the dimension.

Example

This example shows a configuration that displays ancestor attribute values.

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

Precedence_rules elements

The precedence_rules elements contain the precedence rules for your application.

Precedence rules allow your application to delay the display of Endeca standard or managed attributes the user triggers the display. In other words, precedence rules are triggers that cause attributes that were not previously displayed to now be available. This makes navigation through the data easier, and is essential to avoid information overload problems.

PRECEDENCE_RULE

The PRECEDENCE_RULE element allows your application to suppress refinements for an Endeca attribute until some condition is met. This makes navigation through the data easier and is essential to avoid information overload problems.

For example, suppose the records in an application have separate City and State attributes. It would make sense to hide the City attribute until the user has narrowed down to a specific State, because it doesn't make sense to pick a City before a State. (For example, choosing "Portland" would select records in both Portland, OR and Portland, ME.) To accomplish this, create a precedence rule with State as the trigger and City as the target.

A precedence rule has a trigger (also known as the source) and a target (also known as the destination). The trigger can be a managed attribute, managed attribute value, standard attribute, or standard attribute value. The target can be a managed attribute or a standard attribute - it can't be a value.

Format

```
<!ELEMENT PRECEDENCE_RULE EMPTY>
<!ATTLIST PRECEDENCE_RULE
    TYPE   (STANDARD | LEAF | PROPERTY)  #REQUIRED
    SRC_DIMENSION    CDATA               #IMPLIED
    SRC_DVAL_SPEC    CDATA               #IMPLIED
    DEST_DIMENSION   CDATA               #IMPLIED
    DEST_DVAL_SPEC   CDATA               #IMPLIED

    SRC_PROPERTY     CDATA               #IMPLIED
    SRC_PVAL         CDATA               #IMPLIED
    DEST_PROPERTY    CDATA               #IMPLIED
>
```

Attributes

The PRECEDENCE_RULE element has the following attributes.

TYPE

The type of source attribute value (either standard attribute value or managed attribute value) or standard attribute for a PRECEDENCE_RULE. If the trigger is a managed attribute or managed attribute value, then the TYPE can be either STANDARD (which will fire when any value is selected), or LEAF (which will only fire if a leaf value is selected). If the trigger is a standard attribute or standard attribute value, then the TYPE must be PROPERTY (not STANDARD).

To summarize these options:

- PROPERTY. Use this type for specifying a standard attribute. Thus, PROPERTY means that if the standard attribute value specified as the trigger or any of its descendants are in the navigation state, then the target is presented (one trigger, one target). The target for the PROPERTY type can be either a standard attribute or a managed attribute.
- STANDARD or LEAF. Use one of these types for specifying a managed attribute:
  - STANDARD means that 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).
  - LEAF means that querying any leaf managed value from the trigger managed attribute will cause the target managed value to be displayed (many triggers, one target).

  The target for STANDARD and LEAF types can be a standard attribute or a managed attribute.

Note: If, for a managed attribute, you by mistake specify PROPERTY as the type of source attribute value instead of either STANDARD or LEAF, the precedence rule may not trigger, if you make a selection from this managed attribute in navigation.

SRC_DIMENSION and **SRC_DVAL_SPEC**

Specifies the source managed attribute name (SRC_DIMENSION) and source managed attribute value spec (SRC_DVAL_SPEC) that must be selected before the user can see the destination attribute.

DEST_DIMENSION and **DEST_DVAL_SPEC**

Specifies the destination managed attribute name (DEST_DIMENSION) and destination managed value spec (DEST_DVAL_SPEC) that appears after the source attribute value is selected.

SRC_PROPERTY and **SRC_PVAL**

Specifies the source standard attribute name (SRC_PROPERTY) and that must be selected before the user can see the destination attribute. Optionally, a source standard attribute value (SRC_PVAL) can also be specified to further refine the trigger to a specific standard value.

DEST_PROPERTY

Specifies the destination standard attribute name (DEST_PROPERTY) that appears after the source attribute value is selected.

To summarize the rules for specifying TYPE:

- Specify SRC_DIMENSION and SRC_DVAL_SPEC when the trigger is a managed attribute or managed attribute value. (SRC_DVAL_SPEC is required, but should be "/" if the trigger is a managed attribute.)
- Specify SRC_PROPERTY and (optionally SRC_PVAL when the trigger is a standard attribute or standard attribute value.
- Specify DEST_DIMENSION and DEST_DVAL_SPEC when the target is a managed attribute. (DEST_DVAL_SPEC is required and *must* be set to "/".)
- Specifiy DEST_PROPERTY when the target is a standard attribute.

Sub-elements

PRECEDENCE_RULE contains no sub-elements.

Example

This example shows a STANDARD-type precedence rule taken from the wine reference.

```
<PRECEDENCE_RULES>
  <PRECEDENCE_RULE
    DEST_DIMENSION="Winery" DEST_DVAL_SPEC="/"
    SRC_DIMENSION="Region" SRC_DVAL_SPEC="/" TYPE="STANDARD"/>
</PRECEDENCE_RULES>
```

PRECEDENCE_RULES

A PRECEDENCE_RULES element specifies the individual precedence rules available to your application.

Each precedence rule is represented by an individual PRECEDENCE_RULE element.

Format

```
<!ELEMENT PRECEDENCE_RULES
    ( COMMENT?
    , PRECEDENCE_RULE*
    )
>
```

Attributes

The PRECEDENCE_RULES element has no attributes.

Sub-elements

The following table provides a brief overview of the PRECEDENCE_RULES sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
PRECEDENCE_RULE	Allows your application to delay dimension display until the user triggers the display.

Example

This example shows a STANDARD-type precedence rule.

```
<PRECEDENCE_RULES>
  <PRECEDENCE_RULE
    DEST_DIMENSION="Winery" DEST_DVAL_SPEC="/"
    SRC_DIMENSION="Region" SRC_DVAL_SPEC="/" TYPE="STANDARD"/>
</PRECEDENCE_RULES>
```

Recsearch_config elements

The Recsearch_config element configures record search.

RECSEARCH_CONFIG

A RECSEARCH_CONFIG element sets up the configuration of attributes for record searches.

Record searches search against the text collection that consists of the names of all the attribute values in the data set.

Format

```
<!ELEMENT RECSEARCH_CONFIG
    ( COMMENT?
    , SEARCH_INTERFACE*
    )
>
<!ATTLIST RECSEARCH_CONFIG
    WORD_INTERP    (TRUE | FALSE)    "FALSE"
>
```

Attributes

The RECSEARCH_CONFIG element has the following attributes.

WORD_INTERP

Specifies whether to enable word interpretation forms (see-also suggestions) of user query terms considered by the text search engine while processing record search requests. The default value is FALSE.

Sub-elements

The following table provides a brief overview of the RECSEARCH_CONFIG sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
SEARCH_INTERFACE	Represents a named collection of dimensions and/or properties.

Example

This example shows the configuration for a wine implementation.

```
<RECSEARCH_CONFIG>
   <SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="All" NAME="All">
     <MEMBER_NAME RELEVANCE_RANK="4">P_WineType</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="3">P_Name</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="2">P_Winery</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="1">P_Description</MEMBER_NAME>
   </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

Relrank_strategies elements

The Relrank_strategies elements contain the relevance ranking strategies for an application.

The strategies are grouped in the root element RELRANK_STRATEGIES. Each strategy is expressed in a RELRANK_STRATEGY element, which in turn is made of individual relevance ranking modules such as RELRANK_EXACT, RELRANK_FIELD, and so on.

For more information about relevance ranking, see the *Latitude Developer's Guide*.

RELRANK_APPROXPHRASE

The RELRANK_APPROXPHRASE element implements the Approximate Phrase relevance ranking module.

This module is similar to RELRANK_PHRASE, except that in the higher stratum, only the first instance of an exact match of the user's phrase is considered, which improves system performance.

Note: The RELRANK_APPROXPHRASE element is no longer supported. Use the RELRANK_PHRASE element with the APPROXIMATE attribute instead.

Format

```
<!ELEMENT RELRANK_APPROXPHRASE EMPTY>
```

Attributes

The RELRANK_APPROXPHRASE element has no attributes.

Sub-elements

The RELRANK_APPROXPHRASE element has no sub-elements.

RELRANK_EXACT

The RELRANK_EXACT element implements the Exact relevance ranking module.

This module groups results into strata based on how well they match a query string, with the highest stratum containing results that match the user's query exactly. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_EXACT EMPTY>
```

Attributes

The RELRANK_EXACT element has no attributes.

Sub-elements

The RELRANK_EXACT element has no sub-elements.

Example

In this example, the ranking strategy MyStrategy includes the RELRANK_EXACT element.

```
<RELRANK_STRATEGY NAME=”MyStrategy”>
   <RELRANK_STATIC NAME="Availability" ORDER="DESCENDING"/>
   <RELRANK_EXACT/>
   <RELRANK_STATIC NAME="Price" ORDER="ASCENDING"/>
</RELRANK_STRATEGY>
```

RELRANK_FIELD

The RELRANK_FIELD element implements the Field relevance ranking module.

This module assigns a score to each result based on the static rank of the standard attribute or managed attribute member of the search interface that caused the document to match the query. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_FIELD EMPTY>
```

Attributes

The RELRANK_FIELD element has no attributes.

Sub-elements

The RELRANK_FIELD element has no sub-elements.

Example

In this example, the field module is included in a strategy called All_Fields.

```
<RELRANK_STRATEGY NAME="All_Fields">
   <RELRANK_EXACT/>
   <RELRANK_INTERP/>
   <RELRANK_FIELD/>
</RELRANK_STRATEGY>
```

RELRANK_FIRST

The RELRANK_FIRST element implements the First relevance ranking module.

This module ranks documents by how close the query terms are to the beginning of the document. This module takes advantage of the fact that the closer something is to the beginning of a document, the more likely it is to be relevant. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_FIRST EMPTY>
```

Attributes

The RELRANK_FIRST element has no attributes.

Sub-elements

The RELRANK_FIRST element has no sub-elements.

Example

In this example, the ranking strategy All includes the First relevance ranking module.

```
<RELRANK_STRATEGY NAME="All">
   <RELRANK_FIRST/>
   <RELRANK_INTERP/>
   <RELRANK_FIELD/>
</RELRANK_STRATEGY>
```

RELRANK_FREQ

The RELRANK_FREQ element implements the Frequency relevance ranking module.

This module provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_FREQ EMPTY>
```

Attributes

The RELRANK_FREQ element has no attributes.

Sub-elements

The RELRANK_FREQ element has no sub-elements.

Example

This example implements a strategy called Frequency.

```
<RELRANK_STRATEGY NAME="Frequency">
   <RELRANK_FREQ/>
</RELRANK_STRATEGY>
```

RELRANK_GLOM

The RELRANK_GLOM element implements the Glom relevance ranking module.

This module ranks single-field matches ahead of cross-field matches. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_GLOM EMPTY>
```

Attributes

The RELRANK_GLOM element has no attributes.

Sub-elements

The RELRANK_GLOM element has no sub-elements.

Example

This example implements a strategy called Single_Field.

```
<RELRANK_STRATEGY NAME="Single_Field">
   <RELRANK_GLOM/>
</RELRANK_STRATEGY>
```

RELRANK_INTERP

The RELRANK_INTERP element implements the Interpreted (Interp) relevance ranking module.

This module provides a general-purpose strategy that assigns a score to each result document based on the query processing techniques used to obtain the match. Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_INTERP EMPTY>
```

Attributes

The RELRANK_INTERP element has no attributes.

Sub-elements

The RELRANK_INTERP element has no sub-elements.

Example

In this example, the Interpreted module is included in a strategy called All_Fields.

```
<RELRANK_STRATEGY NAME="All_Fields">
   <RELRANK_EXACT/>
   <RELRANK_INTERP/>
   <RELRANK_FIELD/>
</RELRANK_STRATEGY>
```

RELRANK_MAXFIELD

The RELRANK_MAXFIELD element implements the Maximum Field (Maxfield) relevance ranking module.

This module is similar to the Field strategy module, except it selects the static field-specific score of the highest-ranked field that contributed to the match. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_MAXFIELD EMPTY>
```

Attributes

The RELRANK_MAXFIELD element has no attributes.

Sub-elements

The RELRANK_MAXFIELD element has no sub-elements.

Example

This example implements a strategy called High_Rank.

```
<RELRANK_STRATEGY NAME="High_Rank">
   <RELRANK_MAXFIELD/>
</RELRANK_STRATEGY>
```

RELRANK_MODULE

The RELRANK_MODULE element is used to refer to and compose other relevance ranking modules into strategies.

Format

```
<!ELEMENT RELRANK_MODULE (RELRANK_MODULE_PARAM*)>
<!ATTLIST RELRANK_MODULE
     NAME     CDATA     #REQUIRED
>
```

Attributes

The RELRANK_MODULE element has the following attribute.

NAME

NAME refers to another defined relevance ranking module.

Sub-elements

The RELRANK_MODULE element has no supported sub-elements. RELRANK_MODULE_PARAM is not supported.

Example

In this example, a strategy called Best Price is defined. Later, this strategy is included in another strategy definition using the RELRANK_MODULE element.

```
<RELRANK_STRATEGY NAME=”Best Price”>
   <RELRANK_STATIC NAME="Price" ORDER="ASCENDING"/>
</RELRANK_STRATEGY>
<RELRANK_STRATEGY NAME=”MyStrategy”>
   <RELRANK_STATIC NAME="Availability" ORDER="DESCENDING"/>
   <RELRANK_EXACT/>
   <RELRANK_MODULE NAME=”Best Price”/>
</RELRANK_STRATEGY>
```

RELRANK_NTERMS

The RELRANK_NTERMS element implements the Number of Terms (Nterms) relevance ranking module.

This module assigns a score to each result record based on the number of query terms that the result record matches. For example, in a three-word query, results that match all three words are ranked above results that match only two words, which are ranked above results that match only one word. For details, see the *Latitude Developer's Guide*.

This module applies only to search modes where the number of results can vary in how many query terms they match. These search modes include matchpartial, matchany, matchallpartial, and matchallany. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_NTERMS EMPTY>
```

Attributes

The RELRANK_NTERMS element has no attributes.

Sub-elements

The RELRANK_NTERMS element has no sub-elements.

Example

In this example, the Nterms module is included in a strategy called NumberOfTerms.

```
<RELRANK_STRATEGY NAME="NumberOfTerms">
   <RELRANK_NTERMS/>
</RELRANK_STRATEGY>
```

RELRANK_NUMFIELDS

The RELRANK_NUMFIELDS element implements the Number of Fields (Numfields) relevance ranking module.

This module ranks results based on the number of fields in the associated search interface in which a match occurs. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_NUMFIELDS EMPTY>
```

Attributes

The RELRANK_NUMFIELDS element has no attributes.

Sub-elements

The RELRANK_NUMFIELDS element has no sub-elements.

Example

This example implements the Numfields relevance ranking module.

```
<RELRANK_STRATEGY NAME="NumFields">
   <RELRANK_NUMFIELDS/>
</RELRANK_STRATEGY>
```

RELRANK_PHRASE

The RELRANK_PHRASE element implements the Phrase relevance ranking module.

This module states that results containing the user’s query as an exact phrase, or a subset of the exact phrase, should be considered more relevant than matches simply containing the user’s search terms scattered throughout the text. Note that records that have the phrase are ranked higher than records which do not contain the phrase. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_PHRASE EMPTY>
<!ATTLIST RELRANK_PHRASE
    SUBPHRASE        (TRUE | FALSE)   "FALSE"
    APPROXIMATE      (TRUE | FALSE)   "FALSE"
    QUERY_EXPANSION  (TRUE | FALSE)   "FALSE"
>
```

Attributes

The RELRANK_PHRASE element has the following attributes.

SUBPHRASE

If set to TRUE, enables subphrasing, which ranks results based on the length of their subphrase matches.

If set to FALSE (the default), subphrasing is not enabled, which means that results are ranked into two strata: those that matched the entire phrase and those that did not.

APPROXIMATE

If set to TRUE, approximate matching is enabled. In this case, the Phrase module looks at a limited number of positions in each result that a phrase match could possibly exist, rather than all the positions. Only this limited number of possible occurrences is considered, regardless of whether there are later occurrences that are better, more relevant matches.

QUERY_EXPANSION

If set to TRUE, enables query expansion, in which spelling correction, thesaurus, and stemming adjustments are applied to the original phrase. With query expansion enabled, the Phrase module ranks results that match a phrase's expanded forms in the same stratum as results that match the original phrase.

Sub-elements

The RELRANK_PHRASE element has no sub-elements.

Example

This example of the Phrase module enables approximate matching and query expansion, and disables subphrasing.

```
<RELRANK_STRATEGY NAME="PhraseMatch">
   <RELRANK_PHRASE APPROXIMATE="TRUE"
     QUERY_EXPANSION="TRUE" SUBPHRASE="FALSE"/>
</RELRANK_STRATEGY>
```

RELRANK_PROXIMITY

The RELRANK_PROXIMITY element implements the Proximity relevance ranking module.

This module ranks how close the query terms are to each other in a document by counting the number of intervening words. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_PROXIMITY EMPTY>
```

Attributes

The RELRANK_PROXIMITY element has no attributes.

Sub-elements

The RELRANK_PROXIMITY element has no sub-elements.

Example

This example implements a strategy called All that includes the Proximity module.

```
<RELRANK_STRATEGY NAME="All">
   <RELRANK_PROXIMITY/>
   <RELRANK_INTERP/>
   <RELRANK_FIELD/>
</RELRANK_STRATEGY>
```

RELRANK_SPELL

The RELRANK_SPELL element implements the Spell relevance ranking module.

This module ranks matches that do not require spelling correction ahead of spelling-corrected matches. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_SPELL EMPTY>
```

Attributes

The RELRANK_SPELL element has no attributes.

Sub-elements

The RELRANK_SPELL element has no sub-elements.

Example

This example implements a strategy called TrueMatch.

```
<RELRANK_STRATEGY NAME="TrueMatch">
   <RELRANK_SPELL/>
</RELRANK_STRATEGY>
```

RELRANK_STATIC

The RELRANK_STATIC element implements the Static relevance ranking module.

This module assigns a constant score to each result, depending on the type of search operation performed. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_FREQ EMPTY>
<!ATTLIST RELRANK_STATIC
    NAME    CDATA                    #REQUIRED
    ORDER   (ASCENDING|DESCENDING)   #REQUIRED
>
```

Attributes

The RELRANK_STATIC element has the following attributes.

NAME

Specifies the name of a standard or managed attribute that is used for static relevance ranking.

ORDER

Specifies how records should be sorted with respect to the specified standard or managed attribute.

Sub-elements

The RELRANK_STATIC element has no sub-elements.

Example

In this example, the BestPrice strategy consists of the Price managed attribute sorted from lowest to highest.

```
<RELRANK_STRATEGY NAME=”BestPrice”>
   <RELRANK_STATIC NAME="Price" ORDER="ASCENDING"/>
</RELRANK_STRATEGY>
```

RELRANK_STRATEGIES

A RELRANK_STRATEGIES element contains any number of relevance ranking strategies for an application.

Each strategy is specified in a RELRANK_STRATEGY element.

Format

```
<!ELEMENT RELRANK_STRATEGIES
    ( COMMENT?
    , RELRANK_STRATEGY*
    )
>
```

Attributes

The RELRANK_STRATEGIES element has no attributes.

Sub-elements

The following table provides a brief overview of the RELRANK_STRATEGIES sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
RELRANK_STRATEGY	Contains a list of relevance ranking strategies that affect the order in which search results are returned to a user.

Example

This example shows several strategies grouped under the root element RELRANK_STRATEGIES.

```
<RELRANK_STRATEGIES>
   <RELRANK_STRATEGY NAME="Bestseller Strategy">
     <RELRANK_STATIC NAME="Bestseller" ORDER="DESCENDING"/>
   </RELRANK_STRATEGY>
   <RELRANK_STRATEGY NAME="Electronics Strategy">
     <RELRANK_FIELD/>
     <RELRANK_EXACT/>
     <RELRANK_INTERP/>
     <RELRANK_STATIC NAME="Bestseller" ORDER="DESCENDING"/>
     <RELRANK_STATIC NAME="Product_Name" ORDER="ASCENDING"/>
   </RELRANK_STRATEGY>
</RELRANK_STRATEGIES>
```

RELRANK_STRATEGY

The RELRANK_STRATEGY element contains a list of relevance ranking strategies that affect the order in which search results are returned to a user.

Each sub-element of RELRANK_STRATEGY represents a specific type of strategy. If you want several relevance ranking strategies to affect search result, then the order of the sub-elements, which represent the strategies, is significant. The order of the sub-elements defines the order in which the strategies are applied to the search results. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_STRATEGY (
      RELRANK_STATIC
    | RELRANK_EXACT
    | RELRANK_PHRASE
    | RELRANK_APPROXPHRASE
    | RELRANK_GLOM
    | RELRANK_SPELL
    | RELRANK_FIELD
    | RELRANK_MAXFIELD
    | RELRANK_INTERP
    | RELRANK_FREQ
    | RELRANK_WFREQ
    | RELRANK_NTERMS
    | RELRANK_PROXIMITY
    | RELRANK_FIRST
    | RELRANK_NUMFIELDS
    | RELRANK_MODULE
    )+>
<!ATTLIST RELRANK_STRATEGY
    NAME    CDATA    #REQUIRED
>
```

Attributes

The RELRANK_STRATEGY element has the following attribute.

NAME

Specifies the name of the strategy.

Sub-elements

The following table provides a brief overview of the RELRANK_STRATEGY sub-elements.

Sub-element	Brief description
RELRANK_STATIC	Assigns a constant score to each result, depending on the type of search operation perform.

Sub-element	Brief description
RELRANK_EXACT	Groups results into strata based on how well they match the query string, with the highest stratum containing results that match the user's query exactly.
RELRANK_PHRASE	Considers results containing the user’s query as an exact phrase, or a subset of the exact phrase, to be more relevant than matches simply containing the user’s search terms scattered throughout the text.
RELRANK_APPROXPHRASE	Not supported.
RELRANK_GLOM	Ranks single-field matches ahead of cross-field matches.
RELRANK_SPELL	Ranks true matches ahead of spelling-corrected matches.
RELRANK_FIELD	Assigns a score to each result based on the static rank of the dimension or property member of the search interface that caused the document to match the query.
RELRANK_MAXFIELD	Similar to the Field strategy, except it selects the static field-specific score of the highest-ranked field that contributed to the match.
RELRANK_INTERP	A general-purpose strategy that assigns a score to each result document based on the query processing techniques used to obtain the match. Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching.
RELRANK_FREQ	Provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text.
RELRANK_WFREQ	Scores results based on the frequency of user query terms in the result, while weighing the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term.
RELRANK_NTERMS	Assigns a score to each result record based on the number of query terms that the result record matches.
RELRANK_PROXIMITY	Ranks how close the query terms are to each other in a document by counting the number of intervening words.
RELRANK_FIRST	Ranks documents by how close the query terms are to the beginning of the document.
RELRANK_NUMFIELDS	Ranks results based on the number of fields in the associated search interface in which a match occurs.
RELRANK_MODULE	Used to refer to other RELRANK elements and compose them into cohesive strategies.

Example

This example presents a ranking strategy called Product_Search_Rank, which itself is composed of multiple strategies.

```
<RELRANK_STRATEGY NAME="Product_Search_Rank">
   <RELRANK_MODULE NAME="IsAvailable"/>
   <RELRANK_FIELD/>
   <RELRANK_PHRASE/>
   <RELRANK_MODULE NAME="BestPrice"/>
</RELRANK_STRATEGY>
```

RELRANK_WFREQ

The RELRANK_WFREQ element implements the Weighted Frequency (Wfreq) relevance ranking module.

This module scores results based on the frequency of user query terms in the result, while weighing the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term. For details, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT RELRANK_WFREQ EMPTY>
```

Attributes

The RELRANK_WFREQ element has no attributes.

Sub-elements

The RELRANK_WFREQ element has no sub-elements.

Example

This example implements a strategy called Term_Freq.

```
<RELRANK_STRATEGY NAME="Term_Freq">
   <RELRANK_WFREQ/>
</RELRANK_STRATEGY>
```

Search_interface elements

The Search_interface elements are used to build and configure search interfaces.

The file's root element is SEARCH_INTERFACE. Search interfaces control record search behavior for groups of standard and managed attributes.

MEMBER_NAME

The MEMBER_NAME element specifies the name of an Endeca standard or managed attribute that is part of a SEARCH_INTERFACE.

For information on search interfaces, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT MEMBER_NAME (#PCDATA)>
<!ATTLIST MEMBER_NAME
    RELEVANCE_RANK    CDATA    #IMPLIED
    SNIPPET_SIZE      CDATA    "0"
>
```

Attributes

The MEMBER_NAME element has the following attributes.

RELEVANCE_RANK

RELEVANCE_RANK is an unsigned integer that specifies the relevance rank of a match on the specified Endeca standard or managed attribute.

SNIPPET_SIZE

The presence of SNIPPET_SIZE enables snippeting for a MEMBER_NAME and the value of SNIPPET_SIZE specifies maximum number of words a snippet can contain. Omitting this attribute or setting its value equal to zero disables snippeting. For more information, see "Using Snippeting in Record Searches" in the *Latitude Developer's Guide*.

Sub-elements

The MEMBER_NAME element has no sub-elements.

Example

In the following example for a search interface named WineSearch, four Endeca attributes are listed in MEMBER_NAME elements, each with its own relevance rank. A fifth MEMBER_NAME element enables snippeting for the Description attribute.

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="WineRelRank" NAME="WineSearch">
   <MEMBER_NAME RELEVANCE_RANK="4">WineType</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="3">Name</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="2">Winery</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
   <MEMBER_NAME SNIPPET_SIZE="10">Description</MEMBER_NAME>
</SEARCH_INTERFACE>
```

PARTIAL_MATCH

The PARTIAL_MATCH element specifies if partial query matches should be supported for the SEARCH_INTERFACE that contains this element.

For details about searching and search modes, see the *Latitude Developer's Guide*.

Format

```
<!ELEMENT PARTIAL_MATCH EMPTY>
<!ATTLIST PARTIAL_MATCH
    MIN_WORDS_INCLUDED    CDATA    #IMPLIED
    MAX_WORDS_OMITTED     CDATA    #IMPLIED
>
```

Attributes

The PARTIAL_MATCH element has the following attributes.

MIN_WORDS_INCLUDED

Specifies that search results match at least this number of terms in the search query. This value must be an integer greater than zero. The default value of this attribute is one.

MAX_WORDS_OMITTED

Specifies the maximum number of query terms that may be ignored in the search query. This value must be a non-negative integer. If set to zero or left unspecified, any number of words may be omitted (i.e., there is no maximum). The default value of this attribute is two.

Sub-elements

The PARTIAL_MATCH element has no sub-elements.

Example

In this example, the search interface is subject to partial matching in which at least two of the words in the search query are included, and no more than one is omitted.

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="ALWAYS"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="WineRelRank" NAME="WinePartSearch">
   <MEMBER_NAME RELEVANCE_RANK="2">Body</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
   <PARTIAL_MATCH MAX_WORDS_OMITTED="1" MIN_WORDS_INCLUDED="2"/>
</SEARCH_INTERFACE>
```

SEARCH_INTERFACE

The SEARCH_INTERFACE element is a named collection of Endeca standard attributes and/or managed attributes.

Both standard attributes and managed attributes can co-exist in a SEARCH_INTERFACE. The Endeca attributes in the group are specified in MEMBER_NAME elements.

If a standard attribute or managed attribute is not included in any SEARCH_INTERFACE element, then an implicit SEARCH_INTERFACE element is created with the same name as the standard attribute or managed attribute and that single standard attribute or managed attribute as its only member. The value for the CROSS_FIELD_RELEVANCE_RANK is set to 0.

Format

```
<!ELEMENT SEARCH_INTERFACE
    ( MEMBER_NAME+
    , PARTIAL_MATCH?
    , AUTO_SUGGEST?
    , DID_YOU_MEAN?
    )
>
<!ATTLIST SEARCH_INTERFACE
    NAME                        CDATA        #REQUIRED
    DEFAULT_RELRANK_STRATEGY    CDATA        #IMPLIED
    CROSS_FIELD_RELEVANCE_RANK  CDATA        #IMPLIED
    CROSS_FIELD_BOUNDARY        (ALWAYS
                                |ON_FAILURE
                                |NEVER)      "NEVER"
    STRICT_PHRASE_MATCH         (TRUE|FALSE) #IMPLIED
>
```

Attributes

The SEARCH_INTERFACE element has the following attributes.

NAME

A unique name for this search interface.

DEFAULT_RELRANK_STRATEGY

For record search, a default relevance scoring function assigned to a SEARCH_INTERFACE. For example, if your search interface is called Flavors, the DEFAULT_RELRANK_STRATEGY attribute has the value "Flavors_strategy".

CROSS_FIELD_RELEVANCE_RANK

Specifies the relevance rank score for cross-field matches. The value should be an unsigned 32-bit integer. The default value for CROSS_FIELD_RELEVANCE_RANK is 0.

CROSS_FIELD_BOUNDARY

Specifies when the search engine should try to match search queries across standard attribute/managed attribute boundaries, but within the members of the SEARCH_INTERFACE. If its value is set to ON_FAILURE, then the search engine will only try to match queries across standard attribute/managed attribute boundaries if it fails to find any match within a single standard attribute/managed attribute. If its value is set to ALWAYS, then the engine will always look for matches across standard attribute/managed attribute boundaries, in addition to matches within a standard attribute/managed attribute.

By default, the MDEX Engine will not look across boundaries for matches.

STRICT_PHRASE_MATCH

Specifies that the MDEX Engine should interpret a query strictly when comparing white space in the query with punctuation in the source text. If set to FALSE, partial word tokens connected in the source text by punctuation can be matched to a phrase query where the partial tokens are separated by spaces instead of matching punctuation. The default value of this attribute is TRUE.

Sub-elements

The following table provides a brief overview of the SEARCH_INTERFACE sub-elements.

Sub-element	Brief description
MEMBER_NAME	Specifies the name of a property or dimension that is part of a SEARCH_INTERFACE.
PARTIAL_MATCH	Specifies if partial query matches should be supported for the SEARCH_INTERFACE that contains this element.
AUTO_SUGGEST	Specifies how to configure automatic spelling correction for either record searches or value searches.
DID_YOU_MEAN	Specifies how to configure explicit alternatives for search terms as a part of spelling correction.

Example

This example establishes a search interface called AllFields, which contains four members.

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="All" NAME="AllFields">
   <MEMBER_NAME RELEVANCE_RANK="4">WineType</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="3">WineName</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="2">Winery</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
</SEARCH_INTERFACE>
```

Stop_words elements

The Stop_words elements contain words that should be eliminated from a query before it is processed by the MDEX Engine.

Each stop is specified in a STOP_WORD element.

STOP_WORD

The STOP_WORD element identifies words that should be eliminated from a query before it is processed.

Examples of common stop words include the words "the" and "of".

Format

```
<!ELEMENT STOP_WORD (#PCDATA)>
```

Attributes

The STOP_WORD element has no attributes.

Sub-elements

The STOP_WORD element has no sub-elements.

Example

This example shows a common set of stop words.

```
<STOP_WORDS>
   <STOP_WORD>a</STOP_WORD>
   <STOP_WORD>an</STOP_WORD>
   <STOP_WORD>of</STOP_WORD>
   <STOP_WORD>the</STOP_WORD>
</STOP_WORDS>
```

STOP_WORDS

A STOP_WORDS element specifies the stop words enabled in your application.

Each stop word is represented by a STOP_WORD element.

Format

```
<!ELEMENT STOP_WORDS
    ( COMMENT?
    , STOP_WORD*
    )
>
```

Attributes

The STOP_WORDS element has no attributes.

Sub-elements

The following table provides a brief overview of the STOP_WORDS sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
STOP_WORD	Identifies words that should be eliminated from a query before it is processed.

Example

This example shows a common set of stop words.

```
<STOP_WORDS>
   <STOP_WORD>a</STOP_WORD>
   <STOP_WORD>an</STOP_WORD>
   <STOP_WORD>of</STOP_WORD>
   <STOP_WORD>the</STOP_WORD>
</STOP_WORDS>
```

Thesaurus elements

The Thesaurus elements contain thesaurus entries for your application.

Thesaurus entries provide a means to account for alternate forms of a user's query. These entries provide concept-level mappings between words and phrases. For details, see the *Latitude Developer's Guide*.

THESAURUS

A THESAURUS element contains the term equivalence mappings for an application.

THESAURUS is the root element for all thesaurus entries.

Note that the order of sub-elements within THESAURUS is significant. You should add sub-elements in the order in which they are listed in the format section.

For example, THESAURUS_ENTRY sub-elements appear before THESAURUS_ENTRY_ONEWAY. See the example below.

Format

```
<!ELEMENT THESAURUS
    ( COMMENT?
    , THESAURUS_ENTRY*
    , THESAURUS_ENTRY_ONEWAY*
    )
>
```

Attributes

The THESAURUS element has no attributes.

Sub-elements

The following table provides a brief overview of the THESAURUS sub-elements.

Sub-element	Brief description
COMMENT	Associates a comment with a parent element and preserves the comment when the file is rewritten. This element provides an alternative to using inline XML comments of the form <!-- ... -->.
THESAURUS_ENTRY	Indicates a set of word forms (contained in THESAURUS_FORM elements) that are equivalent.
THESAURUS_ENTRY_ONEWAY	Specifies single-direction equivalency mappings.

Example

This example shows the thesaurus entries for an application.

```
<THESAURUS>
   <THESAURUS_ENTRY>
     <THESAURUS_FORM>france</THESAURUS_FORM>
     <THESAURUS_FORM>french</THESAURUS_FORM>
   </THESAURUS_ENTRY>
   <THESAURUS_ENTRY_ONEWAY>
     <THESAURUS_FORM_FROM>Red wine</THESAURUS_FORM_FROM>
     <THESAURUS_FORM_TO>Merlot</THESAURUS_FORM_TO>
     <THESAURUS_FORM_TO>Shiraz</THESAURUS_FORM_TO>
     <THESAURUS_FORM_TO>Bordeaux</THESAURUS_FORM_TO>
   </THESAURUS_ENTRY_ONEWAY>
</THESAURUS>
```

THESAURUS_ENTRY

The THESAURUS_ENTRY element indicates a set of word forms that are equivalent.

The word forms are contained in THESAURUS_FORM elements. A search for any of these forms (including stemming-matched versions) returns hits for all of the forms.

Format

```
<!ELEMENT THESAURUS_ENTRY (THESAURUS_FORM+)>
```

Attributes

The THESAURUS_ENTRY element has no attributes.

Sub-elements

The following table provides a brief overview of the THESAURUS_ENTRY sub-element.

Sub-element	Brief description
THESAURUS_ENTRY	Indicates a set of word forms that are equivalent.

Example

In this example, the noun and adjective forms of a word are made equivalent.

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

THESAURUS_ENTRY_ONEWAY

A THESAURUS_ENTRY_ONEWAY element specifies a single-direction mapping.

Searches for any of the "from" forms (THESAURUS_FORM_FROM elements) also return hits for all of the "to" forms (THESAURUS_FORM_TO elements). The other direction is not enabled; that is, searches for the "to" forms do not return results for either the "from" forms or the other "to" forms.

Format

```
<!ELEMENT THESAURUS_ENTRY_ONEWAY
    ( THESAURUS_FORM_FROM
    , THESAURUS_FORM_TO+
    )
>
```

Attributes

The THESAURUS_ENTRY_ONEWAY element has no attributes.

Sub-elements

The following table provides a brief overview of the THESAURUS_ENTRY_ONEWAY sub-elements.

Sub-element	Brief description
THESAURUS_FORM_FROM	Specifies the "from" form in a one-way word mapping.
THESAURUS_FORM_TO	Specifies the "to" form in a one-way word mapping.

Example

In this example, searches for `Red wine` would return hits for `Red wine` as well as for `Merlot`, `Shiraz`, and `Bordeaux`. Since the equivalence is one-way, more specific searches such as `Shiraz` or `Bordeaux` would not return results for the more general concept `Red wine`.

```
<THESAURUS_ENTRY_ONEWAY>
  <THESAURUS_FORM_FROM>Red wine</THESAURUS_FORM_FROM>
  <THESAURUS_FORM_TO>Merlot</THESAURUS_FORM_TO>
  <THESAURUS_FORM_TO>Shiraz</THESAURUS_FORM_TO>
  <THESAURUS_FORM_TO>Bordeaux</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

THESAURUS_FORM

The THESAURUS_FORM element contains a word form that is used by the THESAURUS_ENTRY element to set an equivalence.

Format

```
<!ELEMENT THESAURUS_FORM (#PCDATA)>
```

Attributes

The THESAURUS_FORM element has no attributes.

Sub-elements

The THESAURUS_FORM element has no sub-elements.

Example

In this example, the noun and adjective forms of a word are made equivalent.

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

THESAURUS_FORM_FROM

The THESAURUS_FORM_FROM element provides the "from" form within a THESAURUS_ENTRY_ONEWAY element.

Format

```
<!ELEMENT THESAURUS_FORM_FROM (#PCDATA)>
```

Attributes

The THESAURUS_FORM_FROM element has no attributes.

Sub-elements

The THESAURUS_FORM_FROM element has no sub-elements.

Example

In this example, searches for `home theater` would return hits for `home theater` as well as for `stereo` and `television`. Because the equivalence is one-way, more specific searches such as `stereo` or `television` would not return results for the more general concept `home theater`.

```
<THESAURUS_ENTRY_ONEWAY>
   <THESAURUS_FORM_FROM>home theater</THESAURUS_FORM_FROM>
   <THESAURUS_FORM_TO>stereo</THESAURUS_FORM_TO>
   <THESAURUS_FORM_TO>television</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

THESAURUS_FORM_TO

The THESAURUS_FORM_TO element provides the "to" form within a THESAURUS_ENTRY_ONEWAY element.

Format

```
<!ELEMENT THESAURUS_FORM_TO (#PCDATA)>
```

Attributes

The THESAURUS_FORM_TO element has no attributes.

Sub-elements

The THESAURUS_FORM_TO element has no sub-elements.

Example

In this example, searches for `home theater` would return hits for `home theater` as well as for `stereo` and `television`. Because the equivalence is one-way, more specific searches such as `stereo` or `television` would not return results for the more general concept `home theater`.

```
<THESAURUS_ENTRY_ONEWAY>
   <THESAURUS_FORM_FROM>home theater</THESAURUS_FORM_FROM>
   <THESAURUS_FORM_TO>stereo</THESAURUS_FORM_TO>
   <THESAURUS_FORM_TO>television</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

Index