Oracle® Endeca Information Discovery

Integrator Components Guide

Version 2.3.0 • June 2012 • Revision A

ORACLE

Copyright and disclaimer

Table of Contents

Copyright and disclaimer . . . . . **ii**

Preface . . . . . **ix**
About this guide ix
Who should use this guide ix
Conventions used in this guide ix
Contacting Oracle Customer Support ix

Chapter 1: Integrator Overview **1**
Integrator UI 1
List of Information Discovery connectors 2
Integrator Server 5

Chapter 2: Before You Begin **6**
Data loading strategies and concepts 6
Which updates to run 6
When to use outer transactions 7
Recommended order of loading data 8
Supported data types 8
Default values for new attributes 9
Additional documentation 11

Chapter 3: Integrator Configuration **12**
Endeca-specific parameters in workspace.prm 12
Creating mdexType Custom properties 14
Setting a default time zone for incoming data 17
Verifying installed Web service versions 18
Specifying multiple record delimiters 19
Configuring SSL 20

Chapter 4: Working with Outer Transaction Graphs **22**
About outer transactions 22
Requirements for running graphs within a transaction 22
Wrapping existing graphs in an outer transaction 23
Creating a Transaction RunGraph graph 24
Format of the steps input file 24
Adding components to the transaction graph 25
Configuring the Reader for the transaction input file 26
Configuring the Edge for Reader component 26
Configuring the Transaction RunGraph connector 27
Running the transaction graph 29
Committing or rolling back an outer transaction 29

Performance impact of transactions 31

Chapter 5: Full Initial Load of Records 32
Overview of the full initial load 32
Creating a project 33
Source data format 34
Adding the source data to the project 36
Creating a graph 37
Adding components to the graph 38
Configuring the components 39
Configuring the Reader component 39
Configuring metadata for the Reader Edge 41
Configuring the Reformat component 42
Configuring metadata for the Reformat Edge 45
Configuring the Bulk Add/Replace Records connector 45
Running the graph to load records 47

Chapter 6: Incremental Updates 48
Overview of incremental updates 48
Adding components to the incremental updates graph 49
Configuring the Reader and the Edge for incremental updates 50
Configuring the Add/Update Records connector 50
Running the incremental updates graph 51

Chapter 7: Loading the Attribute Schema 53
About attribute schema files 53
Loading the standard attribute schema 53
Format of the PDR input file 54
Adding components to the standard attributes schema graph 55
Configuring the Reader for the PDR input file 56
Configuring the Reader Edge 56
Configuring the Reformat component for standard attributes 57
Configuring the Reformat Edge 59
Configuring the Denormalizer component 61
Configuring the Denormalizer Edge 63
Configuring the WebServiceClient component for standard attributes 64
Loading the managed attribute schema 67
Format of the DDR input file 67
Adding components to the managed attributes schema graph 68
Configuring the Reader and the Edge for DDRs 69
Configuring the Reformat component for managed attributes 69
Configuring the Reformat Edge 72
Configuring the Denormalizer and the Edge for DDRs 73
Configuring the WebServiceClient component for managed attributes 73
Using a transaction graph to load the schemas 74

Chapter 8: Loading Configuration Files 75
Types of configuration documents 75
Global Configuration Record 76
dimsearch_config document 77
recsearch_config document 78
relrank_strategies document 79
stop_words document 80
thesaurus document 81
Loading the configuration documents 82
Creating a graph 83
Adding components to the graph 84
Configuring the Reader for the configuration document 84
Configuring metadata for the Reader Edge 85
Configuring the FastSort component 86
Setting metadata for the FastSort component 88
Configuring the first Denormalizer component 88
Configuring metadata for the first Denormalizer 90
Configuring the second Denormalizer component 90
Setting metadata for the second Denormalizer 92
Configuring the WebServiceClient component 92
Loading the GCR 95

Chapter 9: Loading Stemming Files 96
About stemming files 96
Configuring the Reader in the stemming graph 97
Configuring the Edge in the stemming graph 97
Configuring the stemming WebServiceClient component 99

Chapter 10: Loading Precedence Rules 102
About precedence rules 102
Schema for precedence rules 103
Format of the precedence rules input file 104
Adding components to the precedence rules graph 105
Configuring the precedence rules Reader 106
Configuring the Reader Edge 107
Configuring the Reformat component for precedence rules 107
Configuring the precedence rules Reformat Edge 109
Configuring the precedence rules WebServiceClient component 111
Deleting precedence rules 114

Chapter 11: Adding Key-Value Pairs 116
About key-value pair data 116
Format of the KVP input file 116
Configuring the Reader for the KVP input file 117
Configuring the Add KVPs connector 118
Configuring KVP metadata 119
Running the KVPs graph 121

Chapter 12: Loading Taxonomies 122
Overview of loading a taxonomy 122
Format of the taxonomy input file 123
Creating a graph for the taxonomy 124
Adding components to the taxonomy graph 125
Configuring the Reader for the taxonomy input file 125
Configuring the Add Managed Values connector 127
Configuring taxonomy metadata 128
Running the taxonomy graph 129

Chapter 13: Implementing Text Enrichment 130
About Text Enrichment 130
Text Enrichment prerequisites 132
Text Enrichment properties file 132
Creating a Text Enrichment project and graph 139
Configuring the Text Enrichment Reader 140
Configuring the Reader Edge 141
Configuring the Text Enrichment component 142
Configuring the Text Enrichment Edge 143
Configuring the Add/Update Records connector 146

Chapter 14: Using the Text Tagger 147
About the Text Tagger components 147
Building a Text Tagger Regex graph 147
Using regular expressions 148
Creating a Text Tagger Regex graph 149
Configuring the Sample Data Reader 150
Configuring the Reader Edge 150
Configuring the Text Tagger Regex component 151
Configuring the Edge for the Text Tagger Regex component 152
Configuring the Add/Update Records connector 154
Building a Text Tagger Whitelist graph 154
Format of the tag-rules source 156
Creating a Text Tagger Whitelist graph 156
Configuring the source data Readers 158
Configuring the Reader Edges 158
Configuring the Text Tagger Whitelist component 159
Configuring the Edge for the Text Tagger Whitelist component 160
Configuring the Endeca connector 162

Chapter 15: Record Store Reader 163
About the Record Store Reader 163
Adding components to the Record Store Reader graph 164
Configuring the Record Store Reader component 165
Generating Edge metadata with the Record Store Wizard 166
Configuring the Edge for the Record Store Reader component 170
Configuring the Add/Update Records connector 170

Running the Record Store Reader graph 170

Chapter 16: Importing and Exporting the Configuration 172
About importing and exporting 172
Exporting the configuration 173
Adding components to the export graph 173
Configuring the Export Config connector 173
Configuring the Edge in the export graph 174
Configuring the UniversalDataWriter component 176
Importing the configuration 177
Adding components to the import graph 177
Configuring the Reader in the import graph 178
Configuring the Edge in the import graph 178
Configuring the Import Config connector 180
Running the configuration graphs with a transaction graph 181

Chapter 17: Restoring View Definitions 183
About restoring view definitions 183
Adding components to the views graph 184
Configuring the Reader for the views input file 184
Configuring both Edges in the views graph 185
Configuring the Reformat component for views 187
Configuring the views WebServiceClient component 189

Chapter 18: Deleting Data 192
Format of the delete input file 192
Adding components to the delete data graph 193
Configuring the Reader for the delete input file 194
Configuring the metadata for data deletes 194
Configuring the Delete Data connector 195
Running the delete data graph 196

Chapter 19: Creating an Endeca Data Store Snapshot 198
Snapshot operation 198
Configuring the snapshot WebServiceClient component 199
Running the graph to generate the snapshot 201

Chapter 20: Wizards 202
About the Information Discovery wizards 202
Quick Start project 203
New Project from a Flat File Wizard 205
Creating a new project from a flat file 205

Chapter 21: Connector Reference 207
Bulk Add/Replace Records connector 207
Add/Update Records connector 212
Add KVPs connector 215
Add Managed Values connector 217

Delete Data connector 219
Export Config connector 222
Import Config connector 224
Record Store Reader 226
Reset Data Store connector 228
Text Enrichment component 230
Text Tagger Regex component 231
Text Tagger Whitelist component 233
Transaction RunGraph connector 234
Visual and Common configuration properties 238
Connector output ports 241

Chapter 22: Troubleshooting Problems 243
OutOfMemory errors 243
Transaction-related errors 245
Connection errors 246
Multi-assign delimiter error 246

Preface

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

Information is loaded from disparate source systems and stored in a faceted data model that dynamically supports changing data. This integrated and enriched data is made available for search, discovery, and analysis via interactive and configurable applications.

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

About this guide

This guide describes the components in the Oracle Endeca Information Discovery Integrator that are used to ingest data into an Endeca data store.

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

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

Who should use this guide

This guide is intended for developers who are responsible for loading source data and configuration documents into an Endeca data store.

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 Oracle Customer Support

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

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

Chapter 1

Integrator Overview

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

Integrator UI

List of Information Discovery connectors

Integrator Server

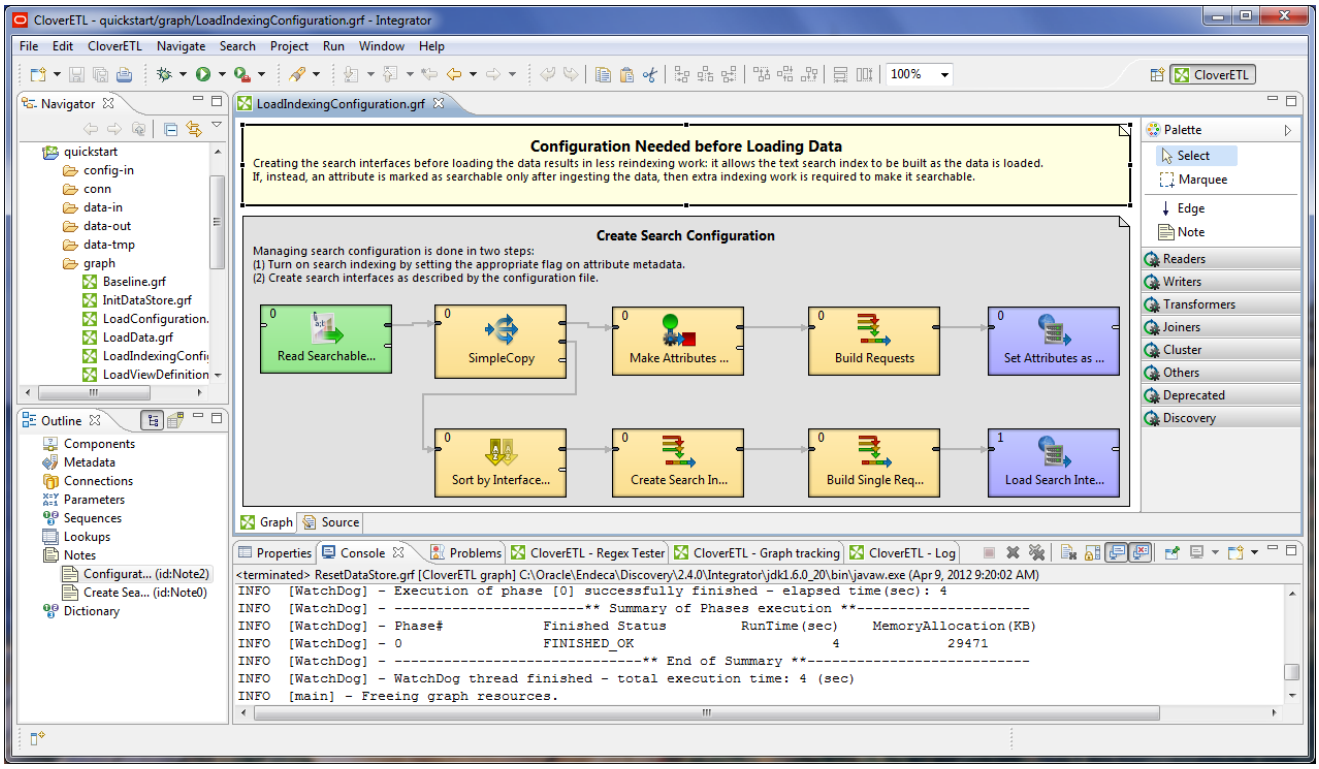
Integrator UI

The Integrator has an easy-to-use interface that lets you quickly create the graphs for loading and updating 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 Information Discovery components to write (send) the data to the Endeca data store. More complex graphs will use additional components, such as Transformer and Joiner components.

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

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



These panes are:

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

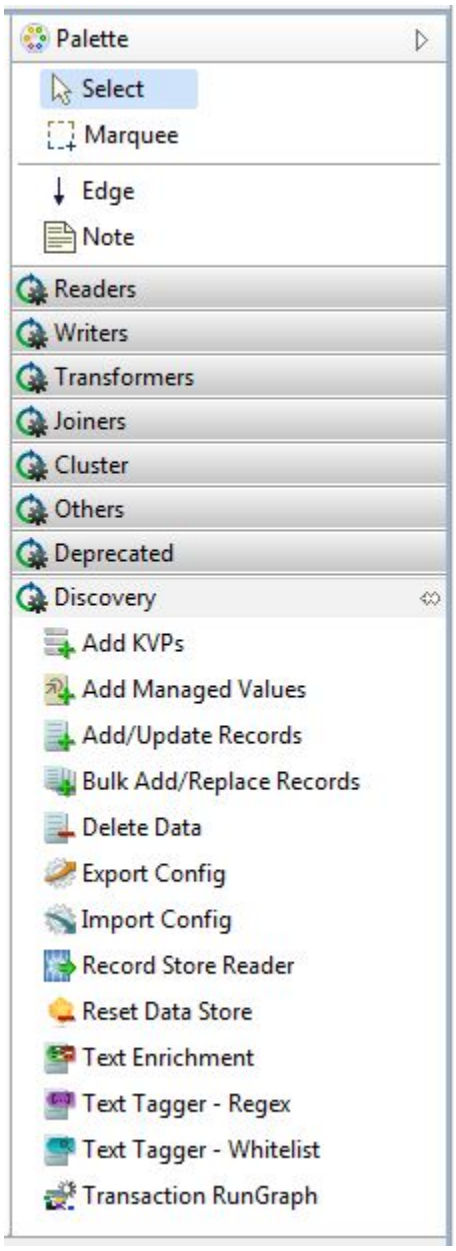
For more information on the Integrator user interface, see the *Oracle Endeca Information Discovery Integrator Guide*.

List of Information Discovery connectors

The Information Discovery connectors are used to load records into an Endeca data store, delete records, export and import an Endeca data store configuration, and start an outer transaction.

The Endeca-developed connectors are specifically designed to work with the records and configuration stored in an Endeca data store. They utilize the Dgraph's Web services and the Bulk Load Interface.

The Information Discovery connectors are grouped in the Integrator Palette under the **Discovery** section:



To use a specific Information Discovery connector, select it from the palette and drag it into your graph.

The following table provides a brief overview of all Information Discovery connectors. For a comprehensive reference of the connectors and their configuration properties, see *Connector Reference on page 207*.

Information Discovery Connector	Description
Add KVPs	Updates 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.
Add Managed Values	Adds a taxonomy (managed values) to a running Endeca data store. If the managed values belong to a managed attribute that currently does not exist in the Endeca data store, the managed attribute is created automatically.
Add/Update Records	Adds new records to a running Endeca data store. You can add records to an empty Endeca data store (this operation is called a full initial load), or to one that already contains records. You can also use this connector to load the user's records schema, by loading the PDRs (Property Description Records) and DDRs (Dimension Description Records). If an Endeca standard attribute to be added does not exist, it is created automatically.
Bulk Add/Replace Records	Adds new records to a running Endeca data store. You can add records to an empty Endeca data store index (this operation is called a full index initial load), or to one that already contains records. If an Endeca standard attribute to be added does not exist, it is created automatically.
Delete Data	Removes KVP assignments from records in the Endeca data store or deletes entire records (that you specify for deletion).
Export Config	Exports the schema and configuration from a specific Endeca data store.
Import Config	Imports the schema and configuration into a specific Endeca data store.
Record Store Reader	A reader component that reads Endeca records from a CAS Record Store.
Reset Data Store	Resets a specific Endeca data store to the empty state by removing all of its records (including the attribute schema) and configuration documents and updating the spelling dictionary.
Text Enrichment	A transformation component that extracts named entities (such as people and companies), themes, and quotations from text source fields. Also provides a summarization of the content and an overall sentiment score.
Text Tagger - Regex	A transformation component that uses a regex (regular expression) to match text in a specified text field, and then tags records.
Text Tagger -Whitelist	A transformation component that matches text (from a whitelist of terms) in a text field and then tags records with a predefined set of tag values (from the same whitelist).

Information Discovery Connector	Description
Transaction RunGraph	Runs other Integrator graphs within it, similar to the standard **RunGraph** component. Unlike the standard **RunGraph** component, the **Transaction RunGraph** component starts an outer transaction and runs multiple sub-graphs within that transaction.

For a comprehensive reference of the connectors and their configuration properties, see the chapter *Connector Reference on page 207*.

Integrator Server

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

The Information Discovery Integrator Server is not required in order to load data into the Endeca data stores. In other words, the Integrator 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 Apache Tomcat enterprise application server.

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

Chapter 2

Before You Begin

This section provides information that you should know before you begin to build Integrator graphs.

Data loading strategies and concepts

Recommended order of loading data

Supported data types

Default values for new attributes

Additional documentation

Data loading strategies and concepts

Endeca recommends the following common approaches and strategies for loading data and conducting other operational tasks.

Which updates to run

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

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

Type of update	Description
Full initial load	This update is also known as baseline update. Its basic idea is the simple loading of data, without the need to preserve any previously configured settings. It includes loading data into an empty Endeca data store. This update assumes that the Endeca data store is empty, and that the configuration and schema have only their default values acquired at when the Endeca data store was first created. As an example, the Baseline graph from the Quick Start project performs an initial data load.

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

When to use outer transactions

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

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

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

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

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

To run an outer transaction, use the Endeca connector **Transaction RunGraph** in the Integrator graph. This connector lets you create a graph that starts and commits an outer transaction in the Endeca data store, utilizing calls to the Transaction Web Service. Using this connector, you can add sub-graphs and components that will run inner transactions inside an outer transaction. Typically, a graph that runs an outer transaction is

useful for running updates. Once such a graph completes, an update to your records is guaranteed to be fully committed to the Endeca data store.

Working with Outer Transaction Graphs

Wrapping existing graphs in an outer transaction

Recommended order of loading data

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

Assuming that you are starting with an empty Endeca data store (that is, only the Endeca Server `create-ds` command 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 Endeca data store.
2. Attribute Schema Configuration, which creates the standard attributes and managed attributes, in this order:
    1. Standard attribute schema, which are the Property Description Records (PDRs)
    2. Managed attribute schema, which are the Dimension Description Records (DDRs)
    3. Managed attribute values (mvals)
3. Attribute Group Configuration, which consists of creating groups and adding attributes to them.
4. Configuration Documents, which consists of these configuration documents in this order:
    1. `relrank_strategies` document (necessary if a relevance ranking strategy is referenced by the next two documents)
    2. `recsearch_config` document
    3. `dimsearch_config` document
    4. `stop_words` document
    5. `thesaurus` document
5. Application Source Records, which consist of the data on which user queries will be made.

You may alter the order to fit the needs of your Information Discovery 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 Studio's Attribute Settings component.

Supported data types

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

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

Integrator Data Types in Metadata	Maps to Dgraph Data Type
`boolean`	`mdex:boolean`
`byte`	Not supported
`cbyte`	Not supported
`date`	`mdex:dateTime`
`decimal`	`mdex:double`
`integer`	`mdex:int`
`long`	`mdex:long`
`number`	`mdex:double`
`string`	`mdex:string`
`string` with an `mdexType` Custom property set to `mdex:duration`	`mdex:duration`
`string` with an `mdexType` Custom property set to `mdex:geocode`	`mdex:geocode`

As the table notes, you can create an `mdexType` Custom property type for the input property's metadata and the Dgraph will use that type when creating the standard attribute's PDR. For details, see the topic *Creating mdexType Custom properties on page 14*.

Default values for new attributes

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

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

Standard attribute default values

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

PDR property	Default setting
`mdex-property_Key`	Set to the standard attribute name specified in the request.

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 Dgraph. The default values for the PDR are the same as listed in the table above, except that mdex-property_IsPropertyValueSearchable will be false (i.e., the managed attribute will be disabled for value search).

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

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

DDR property	Default setting
`mdex-dimension_IsRecordSearchHierarchical`	`false` (hierarchical search is disabled during record searches)

Additional documentation

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

Information Discovery documentation set

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

- *Oracle Endeca Information Discovery Integrator Getting Started Guide* – a guide for ETL developers and data architects who want to explore the basics of the Integrator.
- *Oracle Endeca Information Discovery Integrator Guide* – a comprehensive user's guide for the Integrator.
- *Oracle Endeca Information Discovery Integrator Server Guide* – a comprehensive user's guide for the Integrator Server.

Documentation online

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

- *CloverETL Designer User's Guide* – the online version of the *Oracle Endeca Information Discovery Integrator Guide*.
- *Workbench User Guide* – describes the Eclipse Workbench development environment.
- *Java Development User Guide* – describes how to use the Java development tools.
- *Eclipse Marketplace User Guide* – describes how to use the Eclipse Marketplace Client development tools.

Chapter 3

Integrator Configuration

This chapter provides configuration information for the Integrator.

Endeca-specific parameters in workspace.prm

Creating mdexType Custom properties

Setting a default time zone for incoming data

Verifying installed Web service versions

Specifying multiple record delimiters

Configuring SSL

Endeca-specific parameters in workspace.prm

The `workspace.prm` file contains parameters that define your project.

When a project is created, a default `workspace.prm` file is placed in the project's root directory, and is viewable from the **Navigator** pane. This file lists parameters that are frequently referenced by components in your project, such as locations of the `data-in` and `data-out` directories. Instead of referencing these values directly, you can specify them once in the `workspace.prm` and then reference the parameters when configuring your project's components.

You can add parameters to this file that are specific to the Endeca Server and the Endeca data store being accessed by the project's graphs. The following table lists parameters that affect the Endeca projects in which data is sent to an Endeca data store.

Parameter	Description
`ENDECA_SERVER_HOST`	The host name of the machine on which the Endeca Server is running.
`ENDECA_SERVER_PORT`	The port on which the Endeca Server is listening.
`DATA_STORE_NAME`	The name of the Endeca data store that will be accessed by the Integrator graphs.

Parameter	Description
OUTER_TRANSACTION_ID	The ID of the outer transaction for the Endeca data store. In a new project, this parameter is not specified, and you must add it as follows (with an empty value): `OUTER_TRANSACTION_ID=` This ensures that in your project, you can run components within graphs that either use or do not use transactions: • In a graph that uses an outer transaction, the Information Discovery components (such as **Bulk Add/Replace Records**) and Integrator standard components (such as **WebServiceClient**) rely on the outer transaction ID provided to them by the graph that runs an outer transaction (this graph overrides the ID in this file, for the duration of the outer transaction). Generic components must have the element `OuterTransactionId="${OUTER_TRANSACTION_ID}"` specified as the first element in their request structure. • In a graph that does not use transactions, both types of components ignore this ID if it is empty, which allows them to run outside of an outer transaction.

Note: If you have any of the sample projects loaded in the Integrator, a few additional specific parameters may be listed in this file. These additional parameters are optional and are created in this file for the purposes of the sample projects.

Example: How to specify an outer transaction ID parameter

This example illustrates how to specify an outer transaction ID parameter in the component's configuration.

Note: Information Discovery components automatically reference this ID, if it is specified in the `workspace.prm` file for your project with an empty value. However, you need to configure Integrator standard components that use Endeca Web services to reference this ID, if you plan to use these components in graphs that run outer transactions, in addition to using them in graphs that do not run outer transactions.

For example, if you are using a **WebServiceClient** component to communicate with any of the Endeca Web services, and plan to use this component inside an outer transaction, the **Request Structure** field for the component must include as the first element the `OuterTransactionId` element with an ID of an outer transaction.

Note: If you do not use transactions, then this component should still contain the `OuterTransactionID`, however, because its value is empty in `workspace.prm`, it is ignored when this component runs outside of a transaction.

Specify the following request In the **Request Structure** field for your component. (This example shows the Configuration Web Service, but for other Endeca Web services the structure of the request is similar — the namespace should contain a version of the Web service and the request should include an element that lists the outer transaction ID.):

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

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

Creating mdexType Custom properties

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

The Custom Property feature can be used to specify MDEX types (such as `mdex:duration`, `mdex:time`, and `mdex:geocode`) that are not natively supported in the Integrator. In this case, the ETL developer has to send a string through the Integrator, making sure that the string value is formatted in the way that the Dgraph expects. The new **mdexType** Custom property, in other words, overrides the Integrator native property type when the records are sent to the Dgraph.

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

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

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

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

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

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

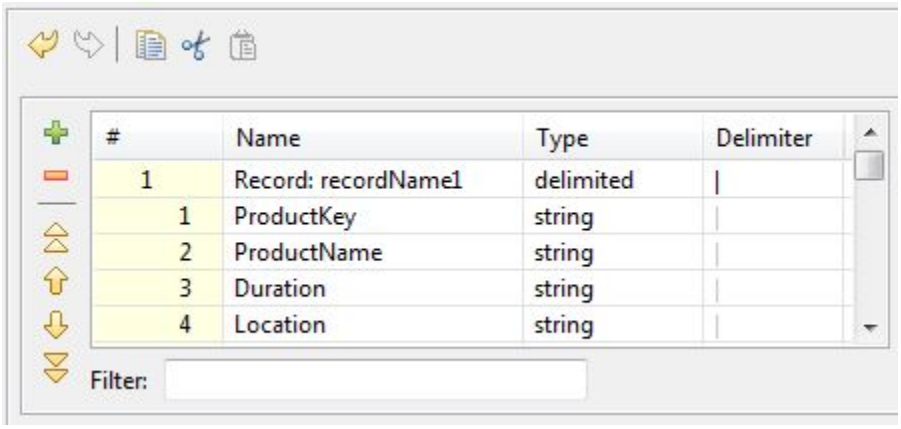
It creates only one record with four standard attributes:

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

To create a Custom property:

1. Create a graph with at least one reader, an Information Discovery 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:

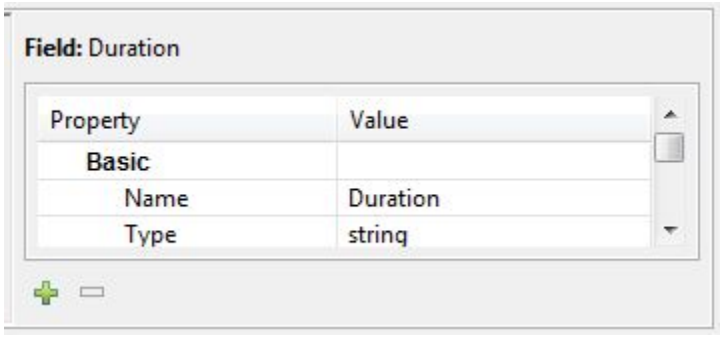


5. In the Record pane of the Metadata editor, make these changes:
   (a) Click the **Record:recordName1** Name field and change the **recordName1** default value to a more descriptive name.
   (b) Change the ProductKey Type to **integer**.

(c) Leave the ProductName Type as **string**.

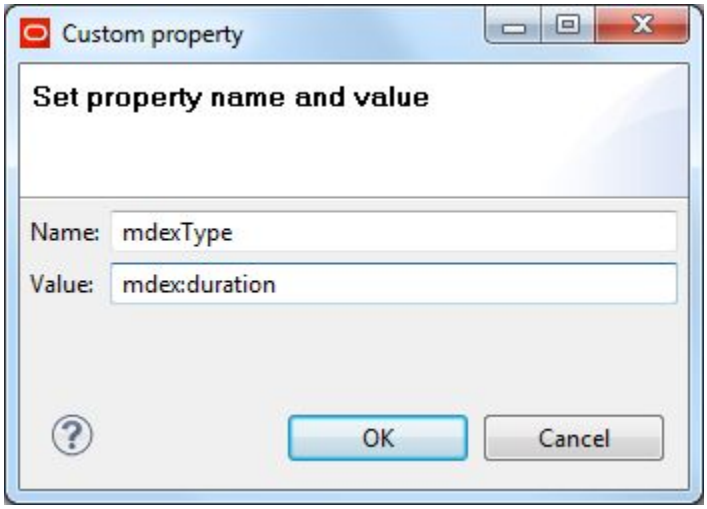
6. To create a Custom property type for the Duration property:

(a) In the Record pane, click the Duration property to high-light it.
The Duration property is displayed in the Field pane on the right, as in this example:

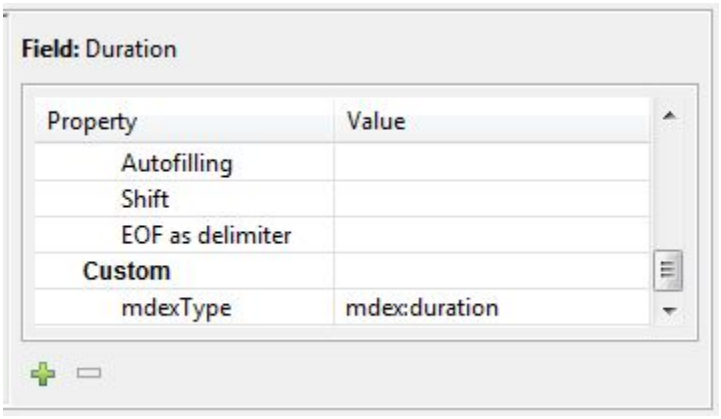


(b) In the Field pane, click the green **+** icon to bring up the Custom property editor.

(c) Enter **mdexType** in the Name field and **mdex:duration** in the Value field.
The Custom property editor should look like this:



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



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

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

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

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

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

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

Setting a default time zone for incoming data

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

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

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

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

Method 1: Modify the source data

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

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

Method 2: Use a Reformat component

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

A sample of the CTL code would be:

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

Method 3: Configure the JVM

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

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

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

Verifying installed Web service versions

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

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

The following example shows how a version number of 1.0 is represented in the namespace for the Configuration Web Service:

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

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

Requests not using these namespaces are rejected.

Requests using a different major version than was released with the Oracle Endeca Server are rejected.

Requests using a minor version that is the same or lower than the version packaged with the server are accepted. For example, if a Transaction Web Service packaged with the server has a version 1.1 and you use a version 1.0 that is listed in the namespace, then this request is accepted. All supported minor versions are listed in the WSDL. Any minor versions higher than supported are rejected.

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

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

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

Specifying multiple record delimiters

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

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

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

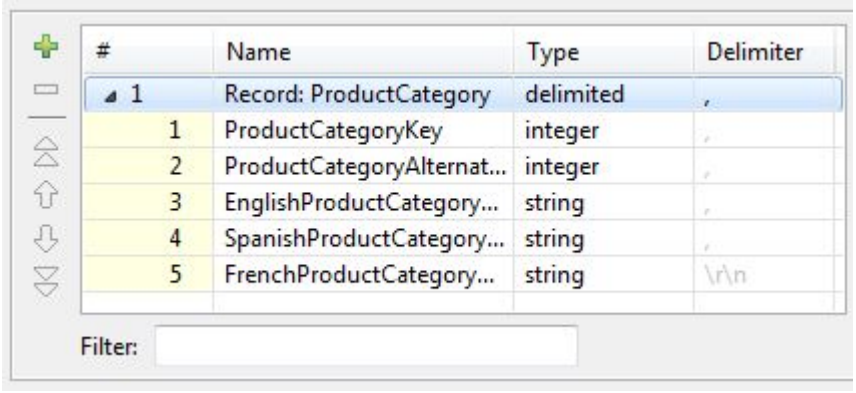
\

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

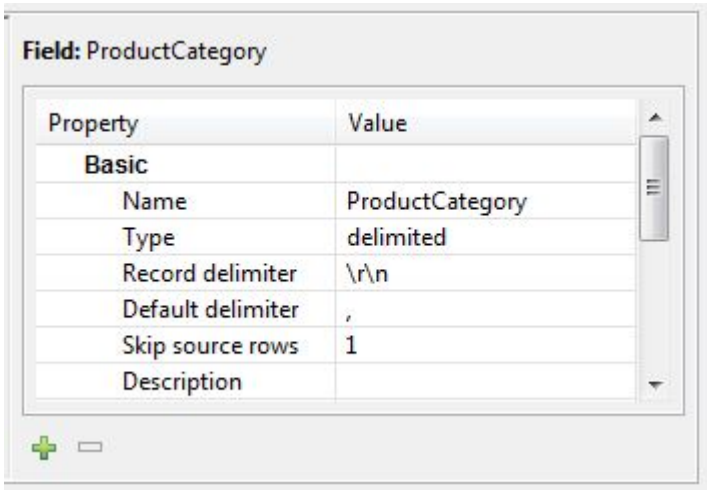
The | (pipe) character is an OR operator and the **\\|** syntax is a way to escape that OR operator in the Integrator interface.

To specify multiple record delimiters in the metadata:

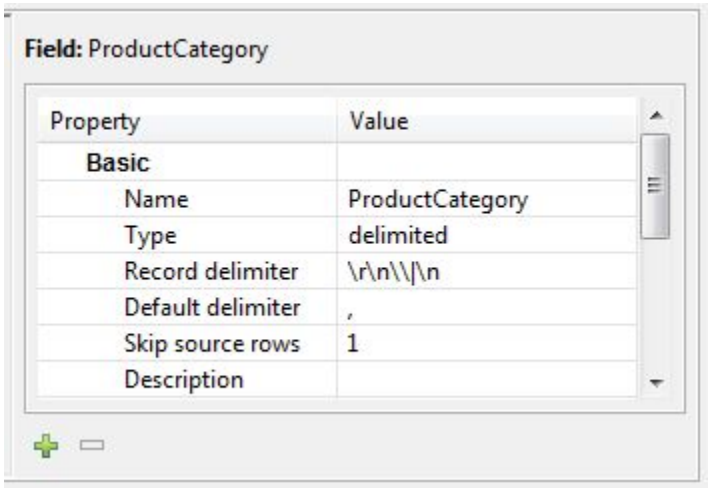
1. In the Record pane of the Metadata Editor, click the first row (the Record row).
   In this example, you would click the Record:ProductCategory row.



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



3. Place the cursor in the Value field of the **Record delimiter** property and select **\r\n\\|\n** from the drop-down menu.
   The Details pane should now look like this:

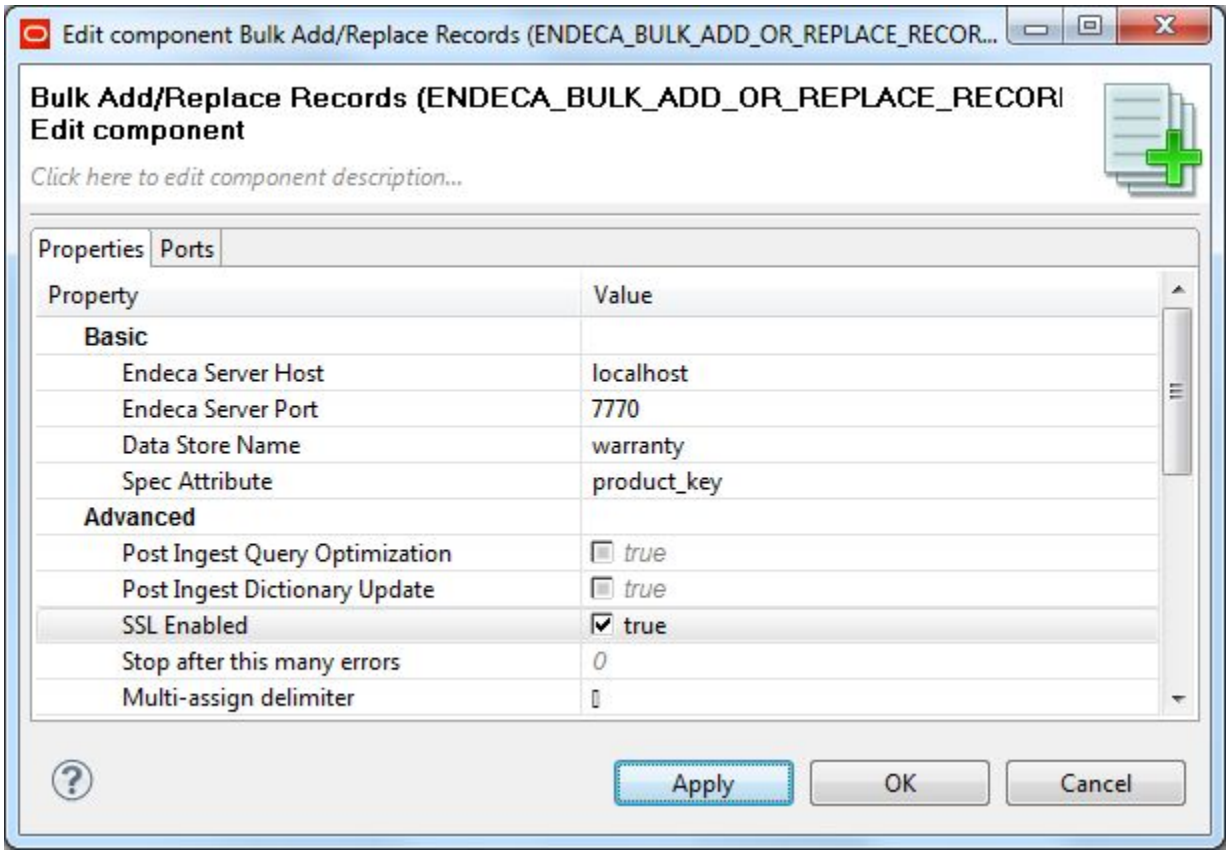


4. Click OK to save your changes made in the Metadata Editor.

Configuring SSL

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

This procedure assumes that you have used the Integrator **Edit component** dialog for the Information Discovery connector and set the **SSL Enabled** configuration property to `true`. For example, this **Bulk Add/Replace Records** connector has been enabled for SSL:



The procedure also assumes that you have created the necessary SSL keystore and truststore certificates.

To configure SSL support for a graph using an Information Discovery connector:

1. Select **Preferences** from the **Window** menu.
2. From the **Preferences** menu, select **Java>Installed JREs**.
3. In the **Installed JREs** menu, click on the checked JRE and then click **Edit**.
   The **Edit JRE** menu is displayed.
4. In the **Default VM Arguments** field, enter the following on a single line. Replace the filenames with the ones you created:

   ```
   -Djavax.net.ssl.keyStore=yourcertkeystorefile.jks
   -Djavax.net.ssl.keyStorePassword=keystorepass
   -Djavax.net.ssl.trustStore=yourtruststorefile.jks
   -Djavax.net.ssl.trustStorePassword=truststorepass
   ```

5. Click **Finish** to apply your change and close the **Edit JRE** menu.
6. Click **OK** to close the **Preferences** menu.

Chapter 4

Working with Outer Transaction Graphs

This chapter describes how to build a transaction graph that runs an open transaction and can include sub-graphs. It also provides information about starting, committing, and rolling back outer transactions.

About outer transactions

Requirements for running graphs within a transaction

Wrapping existing graphs in an outer transaction

Creating a Transaction RunGraph graph

Committing or rolling back an outer transaction

Performance impact of transactions

When to use outer transactions

About outer transactions

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

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

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

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

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

Requirements for running graphs within a transaction

If you would like to use outer transactions in your graphs, consider these requirements.

- Only one outer transaction can run in an Endeca data store at a time. If you have a graph that starts an outer transaction, such as a graph built with the **Transaction RunGraph** connector, it is important not to start another graph that attempts to start another outer transaction, otherwise, a transaction fault error is issued.
- You can run all components specific to an Endeca data store inside a graph that starts an outer transaction. In other words, all such components in the Integrator are transaction-friendly. When any such

component is run within a graph that runs an outer transaction, the underlying update operations from the Web services or Bulk Ingest Interface will reference the outer transaction ID in their calls to the server. This ID is provided to these components by the **Transaction RunGraph** connector. For the duration of the outer transaction, the connector sets the ID to `transaction`. In addition, the ID can be specified as a `OUTER_TRANSACTION_ID=` variable in the `worskpace.prm` file for your project (notice the empty value). This allows the same components to be used in graphs that do not use transactions, without having to modify `worskpace.prm`.

Note: All components specific to an Endeca data store can also run in graphs that do not start an outer transaction. If you have a simple implementation, or if a graph that you are creating is light-weight and is not intended for heavy-duty data loading or configuration updates, it can run on its own and does not necessarily need to be run inside an outer transaction.

- If you are using a **WebServiceClient** component in the Integrator that is configured to run any of the Endeca data store Web services, the **Request Structure** field for the component must include an element `OuterTransactionId` with an ID value of an outer transaction. This element must be specified first in the request. For example, the following request specified in the **Request Structure** references the outer transaction ID as a parameter:

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

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

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

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

Transaction-related errors

Wrapping existing graphs in an outer transaction

You can wrap any of your existing graphs in a graph that uses the **Transaction RunGraph** connector.

To wrap your existing graphs in an outer transaction:

1. Create a parameter in your workspace parameters file `workspace.prm` with this line:

   ```
   OUTER_TRANSACTION_ID=
   ```

   where the value is empty.

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

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

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

3. Modify any standard **RunGraph** components that may possibly run within an outer transaction by specifying `OUTER_TRANSACTION_ID` in the **Graph parameters to pass** field (this field accepts a semicolon-delimited list).
4. Finally, configure a **Transaction RunGraph** connector to reference one or more graphs, and run it.

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

When to use outer transactions

Creating a Transaction RunGraph graph

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

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

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

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

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

In addition, you can configure the **Transaction RunGraph** connector to react to unsuccessful runs, such as it can roll back the outer transaction.

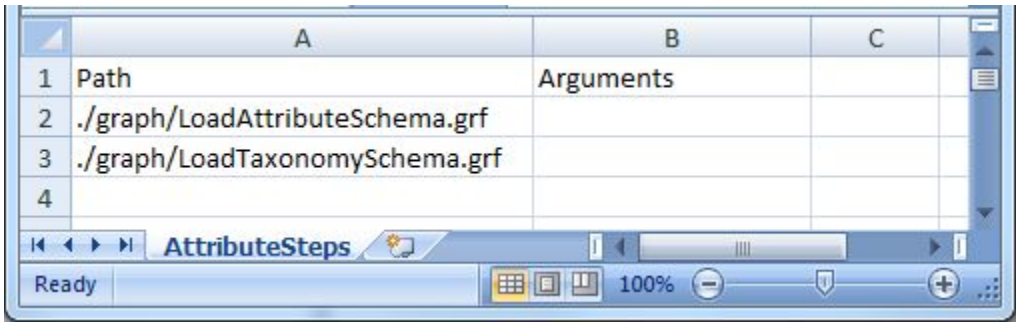
In this section, a sample **Transaction RunGraph** graph will be built to run the two graphs that load the standard attribute and managed attribute schemas into the data store.

Transaction RunGraph connector

Format of the steps input file

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

The `AttributeSteps.csv` sample input file used to list the graphs looks like this:



The first line (the header row) of the sample file has two header properties:

```
Path,Argument
```

The actual names of the header properties can be different from the names used here. The properties are delimited (for example, by the comma in the sample CSV file). After the header row, the second and following rows in the input file contain the input values:

- The `Path` column lists the path names of the graphs to be run by the **Transaction RunGraph** component. The order in which the graphs are listed is the order in which they are run.
- The `Arguments` column specifies any graph command-line arguments. No arguments are specified in our example input file.

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

Transaction ID in the workspace.prm file

When running graphs in a transaction environment, specify an outer transaction ID in your `workspace.prm` file by setting it in the `OUTER_TRANSACTION_ID` variable, as in this example:

```
OUTER_TRANSACTION_ID=
```

Leaving the value empty is important. It allows the sub-graphs to be run outside of an outer transaction if needed, as well as within an outer transaction. When the sub-graphs are run within a **Transaction RunGraph**, the master graph overrides the ID with the string `transaction` for the duration of the outer transaction. When the sub-graphs are run independently of the master graph and outside of a transaction, the empty value from `workspace.prm` is used, which enables the graphs to ignore the ID attribute in the request to the Endeca data store.

Adding components to the transaction graph

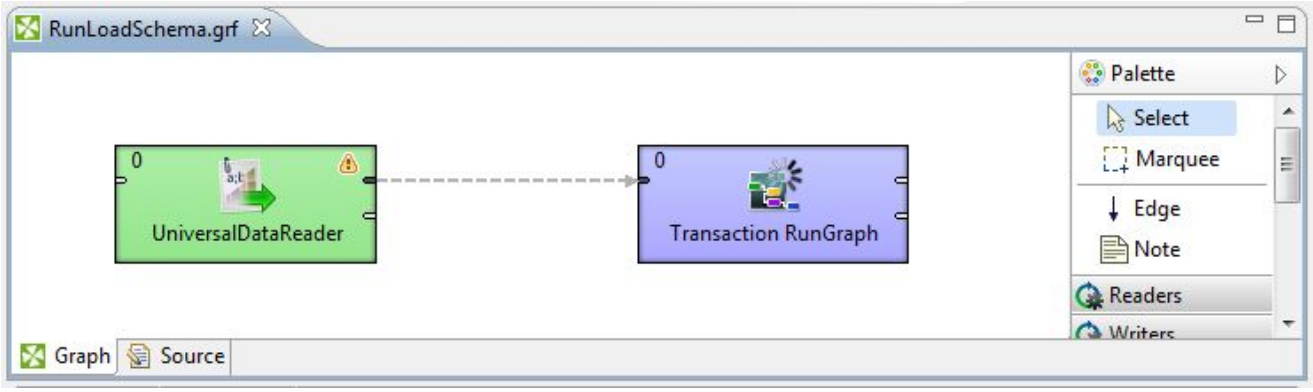
This topic describes the two components that must be added to the transaction graph.

This procedure assumes that you have created an empty graph (named RunLoadSchema in our example).

To add components to the transaction graph:

1. In the Palette pane, drag the **UniversalDataReader** component from the **Readers** section.
2. In the Palette pane, drag the **Transaction RunGraph** component from the **Discovery** section.
3. In the Palette pane, click **Edge** and use it to connect the components.
4. Save the graph.

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



Configuring the Reader for the transaction input file

This task describes how to configure the **UniversalDataReader** component to read in the file that lists the graphs to be run.

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

To configure the Reader component for the run-graphs 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 transaction input file (`AttributeSteps.csv` in our example) and click **OK**.
3. Change the **Number of skipped records per source** field set to 1.
   The reason is that we do not want the first row (the header property row) to be read in as data.
4. Optionally, use the **Component name** field to provide your own name for the component.
5. Click **OK** to apply your configuration changes to the Reader component.
6. Save the graph.

The next step is to configure the Reader's Edge metadata.

Configuring the Edge for Reader component

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

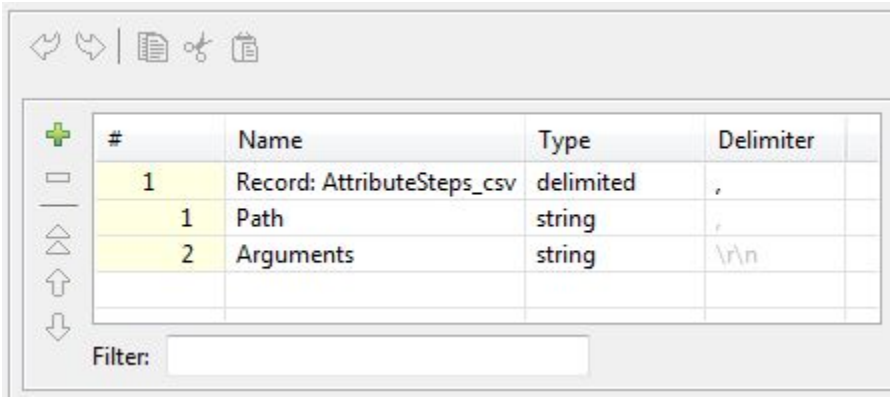
This Metadata definition task will use the Metadata Editor. In the procedure, the column names will be extracted from the input file via a reparsing operation.

To configure the Metadata definition for the Reader Edge in the transaction graph:

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**, browse for the PDR input file, select it, and click **OK**.
   (a) Double-click the **data-in** folder.
   (b) Select the transaction input file and click **OK**.
   You are returned to the Flat File dialog.
4. In the Flat File dialog, make sure that the **Record type** field is set to **Delimited** and then click **Next**.
   The input data is loaded into the Metadata Editor, with the properties named Field1, Field2, and so forth.
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, which should look like this:



6. In the upper pane of the Metadata Editor:
   (a) Optionally, click the **Record** Name field and change the name of the metadata to a more descriptive name.
   (b) Make sure that the **Type** field of all the properties is set to type **string**.
   (c) Verify that the fields have the correct delimiter character set (which is the comma for our example).
7. When you have input all your changes, click **Finish**.
8. Save the graph.

The next step is to configure the **Transaction RunGraph** connector.

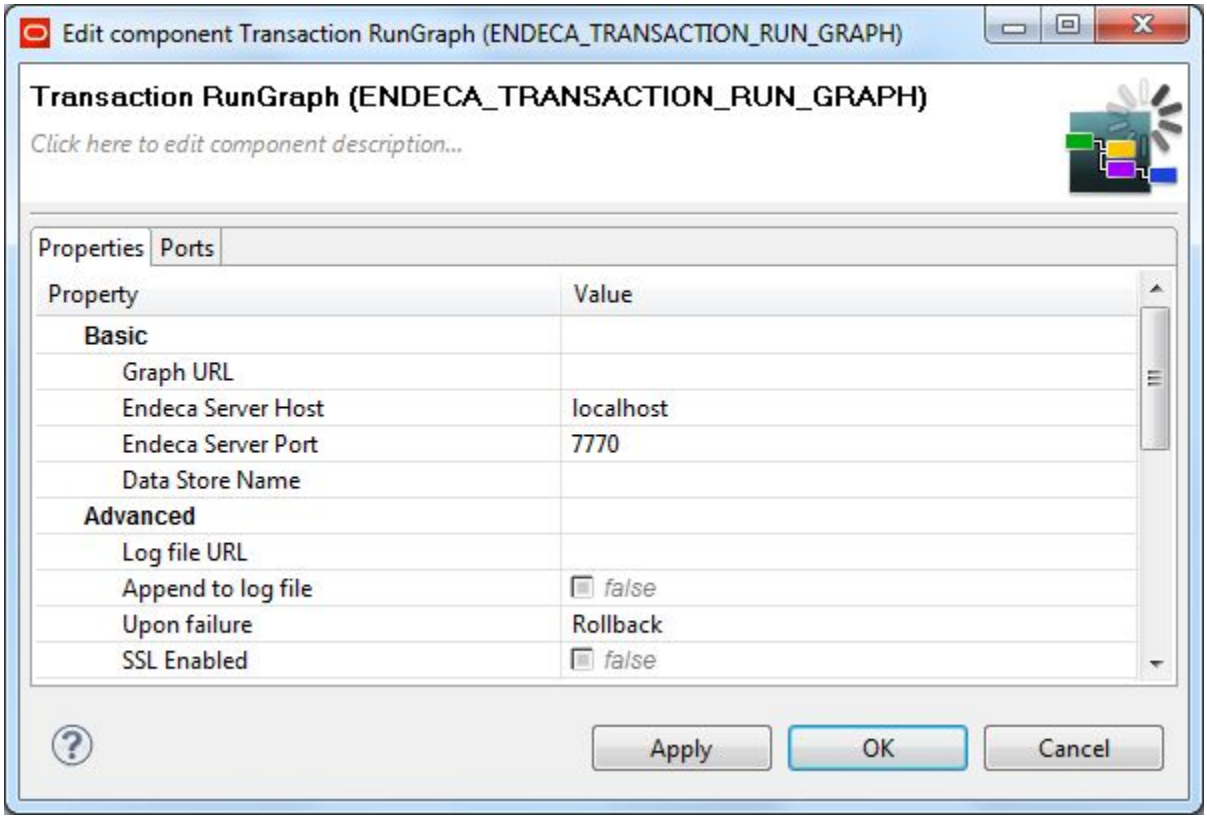
Configuring the Transaction RunGraph connector

This topic describes how to configure the **Transaction RunGraph** connector to run the graphs.

This procedure assumes that you have created a graph and added the **Transaction RunGraph** connector.

To configure the **Transaction RunGraph** connector:

1. In the Graph editor, double-click the **Transaction RunGraph** component.
   The Edit Component dialog is displayed.



2. In the Edit Component dialog, make these settings:
   - **Endeca Server Host**: Enter the host name of the machine on which the Endeca Server is running. `localhost` can be used as the name.
   - **Endeca Server Port**: Enter the number of the port on which the Endeca Server is listening.
   - **Data Store Name**: Enter the name of the Endeca data store on which the transaction will run.
   - **Upon failure**: Select the action that the component should take upon a transaction failure: **Rollback** (roll back to the state before the outer transaction had started, and commit the outer transaction), **Commit** (commit those changes that have been made successfully before the failure occurred, and commit the outer transaction), or **Do nothing** (nothing is done, which means you may need to manually stop the outer transaction).
   - **SSL Enabled**: Toggle this field to `true` only if the Endeca Server is SSL-enabled.

   You can leave the other settings at their defaults.
3. When you have input all your changes, click **OK**.
4. Save the graph.

The final step is to run the transaction graph.

Running the transaction graph

After creating the transaction graph and configuring its components, you can run the graph. This means that its sub-graphs will run inside an outer transaction.

To run the transaction graph:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **Transaction RunGraph** connector and that the Endeca data store has been started.
2. Run the graph by clicking 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 this final output message:

```
INFO  [main] - Execution of graph successful !
```

Committing or rolling back an outer transaction

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

This procedure assumes that you have added a `OUTER_TRANSACTION_ID` variable to the project's `workspace.prm` file.

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

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

Typically, you may need to close an already running outer transaction after you receive a transaction-related error, when trying to run one of your graphs. To identify whether an outer transaction is currently running, use the `listOuterTransaction` operation of the Transaction Web Service.

This topic describes how to create these types of transaction graphs using the Transaction Web Service operations:

- The Commit Transaction graph uses the `commitOuterTransaction` operation to end a transaction. If an outer transaction with the specified ID is in progress and if the operation succeeds, the Endeca data store commits the changes to the index made within this outer transaction, and starts processing unqualified queries and updates against this version of the index.
- The Rollback Transaction graph uses the `rollBackOuterTransaction` operation to roll back an outer transaction. In the event that a running outer transaction fails, this operation lets you roll back to the previously-committed version of the index and stop the outer transaction.

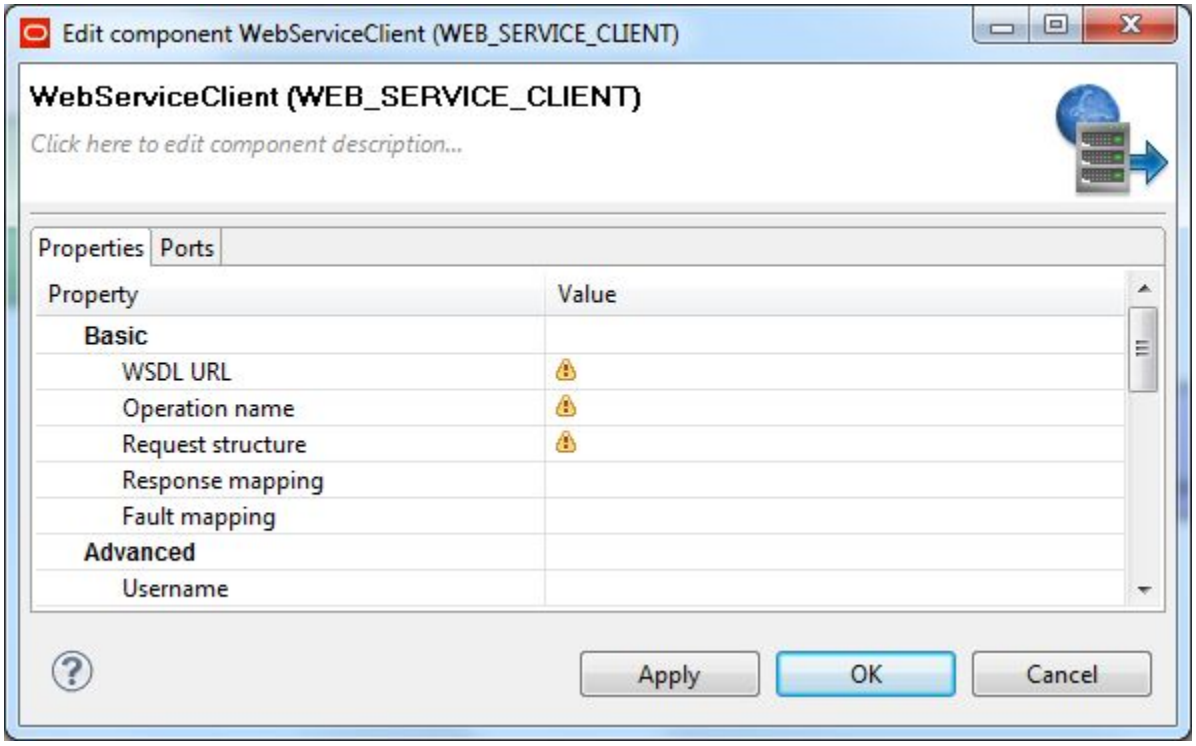
To create a commit transaction or rollback transaction graph:

1. Create an empty graph and add a **WebServiceClient** component.
2. Make sure that the Endeca data store instance is running and its Administration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

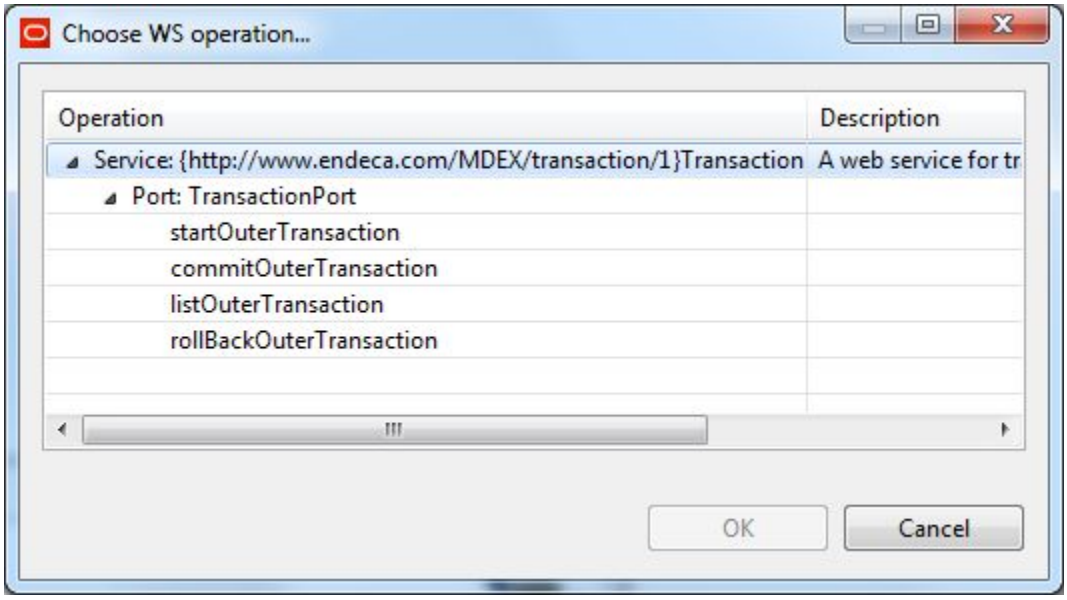
```
http://localhost:7770/ws/transaction/bikes?wsdl
```

The URL command returns the WSDL of the Transaction Web service.

3. In the Graph editor, double-click the **WebServiceClient** component.
The Edit Component dialog is displayed.



4. In the **WSDL URL** field, enter the same URL as in Step 2.
5. In the **Operation name** field, click the **...** browse button, which displays the **Choose WS operation** dialog:



6. In the **Choose WS operation** dialog, select **commitOuterTransaction** and then click **OK**.
   For a rollback graph, you would choose the **rollBackOuterTransaction** operation.
7. In the Edit Component dialog, click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the browse button to display an empty **Edit request structure** dialog.
8. For a **commit** transaction operation, add this text to the **Generate request** field and then click **OK**:

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

   For a **rollback** transaction operation, add this text instead:

```
<ns:request xmlns:ns="http://www.endeca.com/MDEX/transaction/1/0">
  <ns:rollBackOuterTransaction>
    <ns:OuterTransactionId>${OUTER_TRANSACTION_ID}</ns:OuterTransactionId>
  </ns:rollBackOuterTransaction>
</ns:request>
```

9. Save the graph.

Transaction-related errors

Performance impact of transactions

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

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

Chapter 5

Full Initial Load of Records

This section describes how to create an Integrator project and a graph that will perform a full initial load of records into an Endeca data store.

Overview of the full initial load

Creating a project

Source data format

Creating a graph

Adding components to the graph

Configuring the components

Running the graph to load records

Overview of the full initial load

This section walks you through the various tasks involved in creating a graph that can perform initial load of source records into an Endeca data store.

The task that this section covers is how to perform an initial ***full load*** of your source records into an Endeca data store. (Full loads are also known as ***baseline updates***.) As the source records are ingested, they are converted into Endeca records and are indexed by the Dgraph that is servicing that Endeca data store.

This process assumes that:

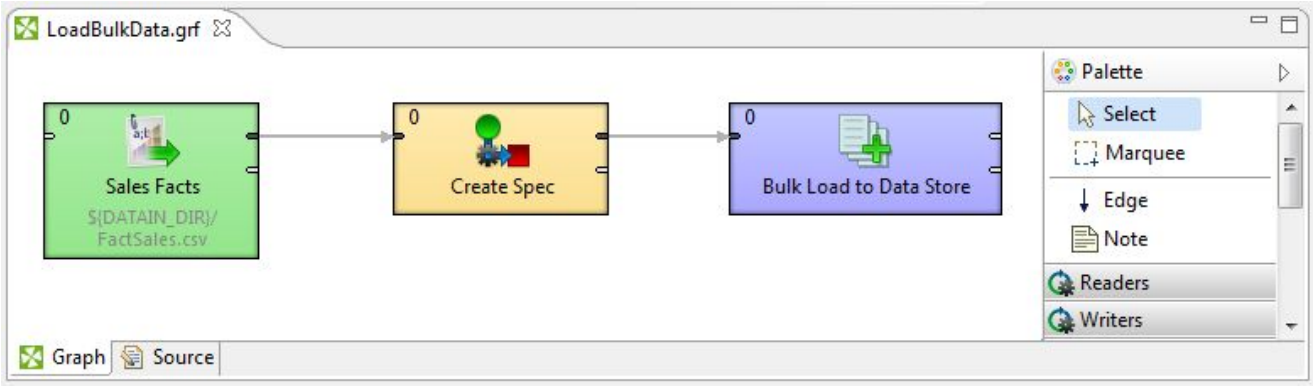
- The Endeca data store is empty of user source data.
- You have already loaded your attribute schema (PDRs and DDRs) into the Endeca data store. The load-schema procedure is documented in *Loading the Attribute Schema on page 53*.

Note: You can also initially your data without having to first load 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.

Sample full load graph

The Integrator Sample Application has an extensive graph (named LoadData) that uses the **Bulk Add/Replace Records** connector. The LoadData graph inputs ten data source files and uses **ExtHashJoin** joiner components to join all the source data.

In order to simplify the description of a bulk load graph, a subset of the LoadData graph is used in this chapter. This sample subset graph (named LoadBulkData) uses three components:



The three components are:

- The **Sales Facts** component is a **UniversalDataReader** that reads in sales transaction records from one source data file.
- The **Create Spec** component is a **Reformat** component that creates the primary-key attribute for the records.
- The **Bulk Load to Data Store** component is a **Bulk Add/Replace Records** connector. This Information Discovery connector sends the records to the Bulk Load Interface of the Dgraph.

The source data is sales transaction information stored in a CSV file, with each source record having multiple columns that are delimited by the comma 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

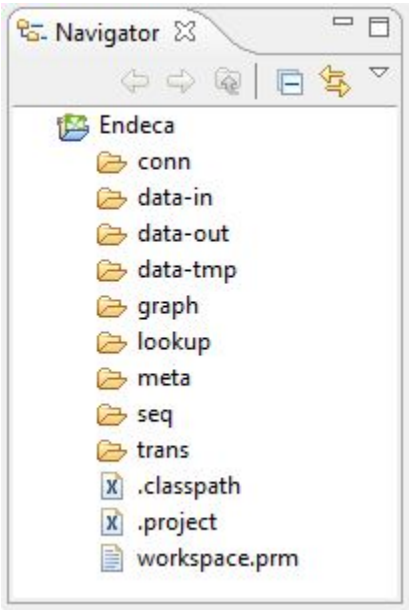
You must create an Integrator project in which you will build your graph.

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 Integrator 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 Endeca project in an expanded format:



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

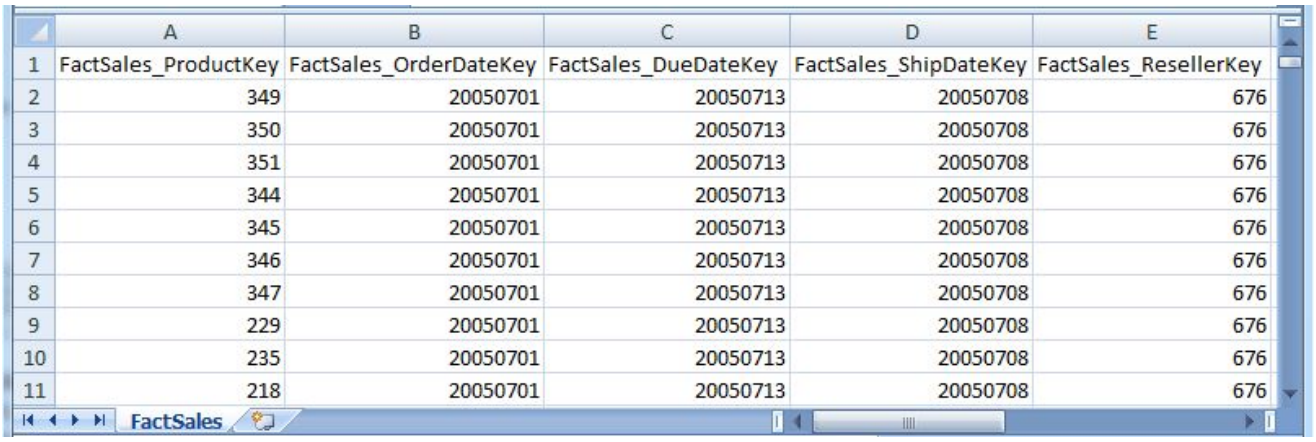
Source data format

You can load source data from a variety of formats.

Your Information Discovery 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 source records are stored in a CSV file named `FactSales.csv` (the file is in the `data-in` folder of the Integrator Quick Start sample application).

The following image, which shows the beginning lines of the `FactSales.csv` input file, illustrates how the source data is organized in a two-dimensional format:



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

FactSales

You specify the location and format of the source data to be loaded in the Integrator reader component in the graph. The reader component passes the data to the Information Discovery 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 Dgraph. The records are then loaded into the Dgraph 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 Dgraph 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 property. For more information on primary keys, see the *Oracle Endeca Server Data Loading Guide*.

In our sample graph, the `FactSales.csv` input file does not have a field that contains unique values. Therefore, the **Create Spec** component creates the FactSales_RecordSpec primary-key attribute by concatenating two attributes. 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 Dgraph will accept attribute names with hyphens (because hyphens are valid NCName characters), the Integrator will not accept source property names with hyphens as metadata. Therefore, if you have a source property name such as "Ship-Date", make sure you remove the hyphen from the name.

Using multi-assign data

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, Color1 and Color2) in which each property has only one value, you can instead have one property (say, Color) with multiple values, as in this simple example:

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

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

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

Keep in mind that an Endeca 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

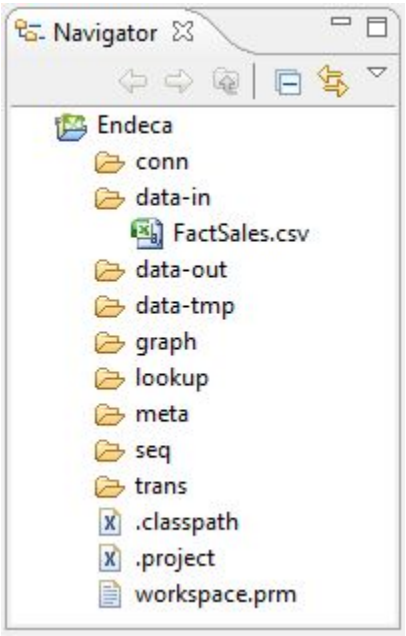
The easiest way to add your source data is to copy it into the project's `data-in` directory.

This procedure assumes that you are copying a CSV (comma-separated value) file named `FactSales.csv`, which contains the delimited records. The comma character is the delimiter. You can use other input file formats, such as a CSV (comma-separated value) file. The source records are stored in a CSV file named `FactSales.csv` (the file is in the `data-in` folder of the Integrator Sample Application).

To add the source data file to your 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 `FactSales.csv` is now available to the project's graphs.

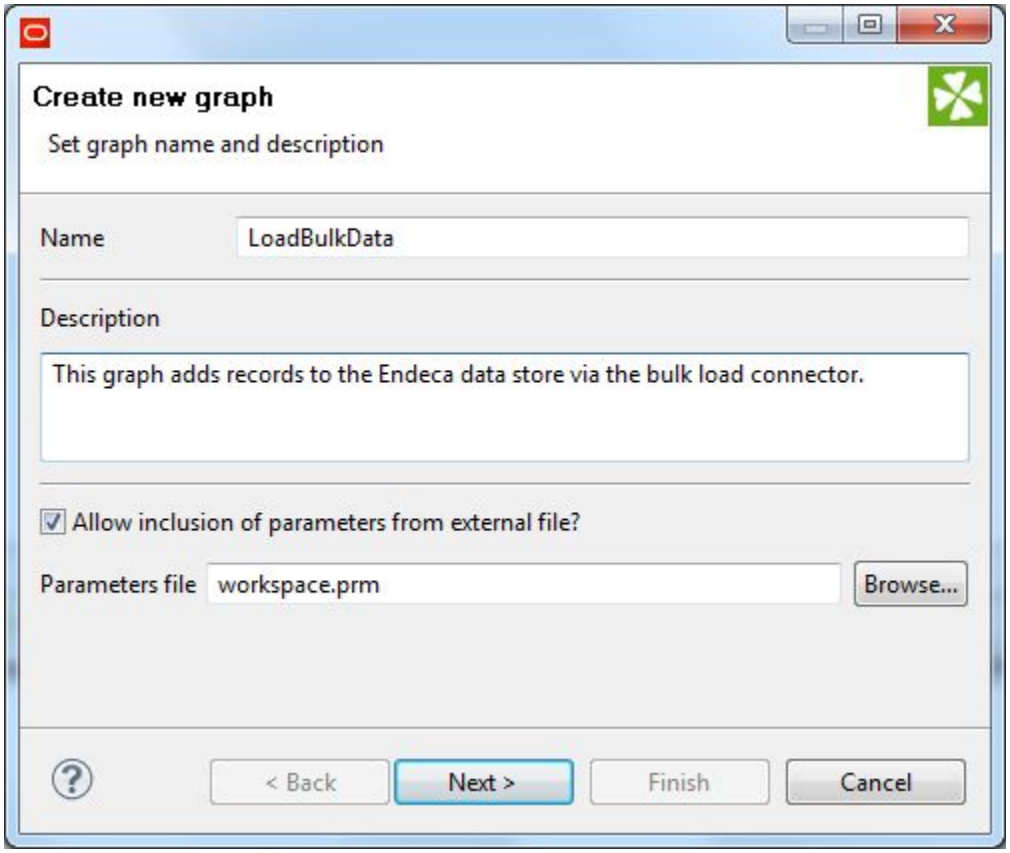
Creating a graph

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 an Integrator 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 **LoadBulkData**.
   (b) Optionally, type in a description.
   (c) 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:



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 components to the graph.

Adding components to the graph

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

The components to be added are:

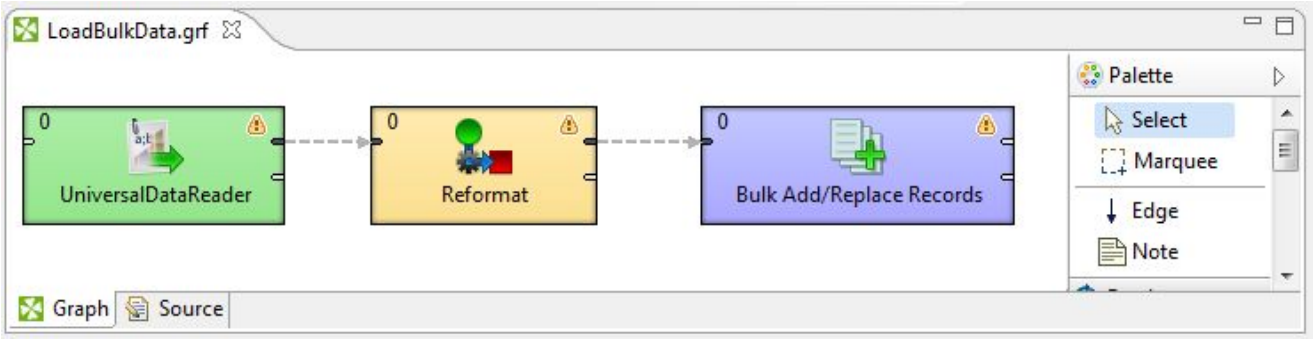
- A *Reader* is a graph component that reads in source data. In our example, the **UniversalDataReader** component is used because it can read in data from CSV files.
- A *Transformer* component can transform the incoming data before it is sent to the next component. In our example, the **Reformat** component is used to create a new primary-key attribute from two existing attributes. Note that this component would not be necessary if you were using an existing attribute as the primary-key attribute.
- 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 the data.

In addition, an *Edge* will be added to connect the components. The configurations for all 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 **Transformers** section and drag the **Reformat** component into the Graph Editor.
3. In the Palette pane, open the **Discovery** section and drag the **Bulk Add/Replace Records** component into the Graph Editor.
4. In the Palette pane, click **Edge** and use it to connect the 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.
5. From the File menu, click **Save** to save the graph.

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



The next tasks are to configure these components for the source data and for a connection to the appropriate Endeca data store.

Configuring the components

This section describes how to configure the **UniversalDataReader**, **Reformat**, and **Bulk Add/Replace Records** components, as well as the metadata for the two Edge components.

The components will be configured in this order:

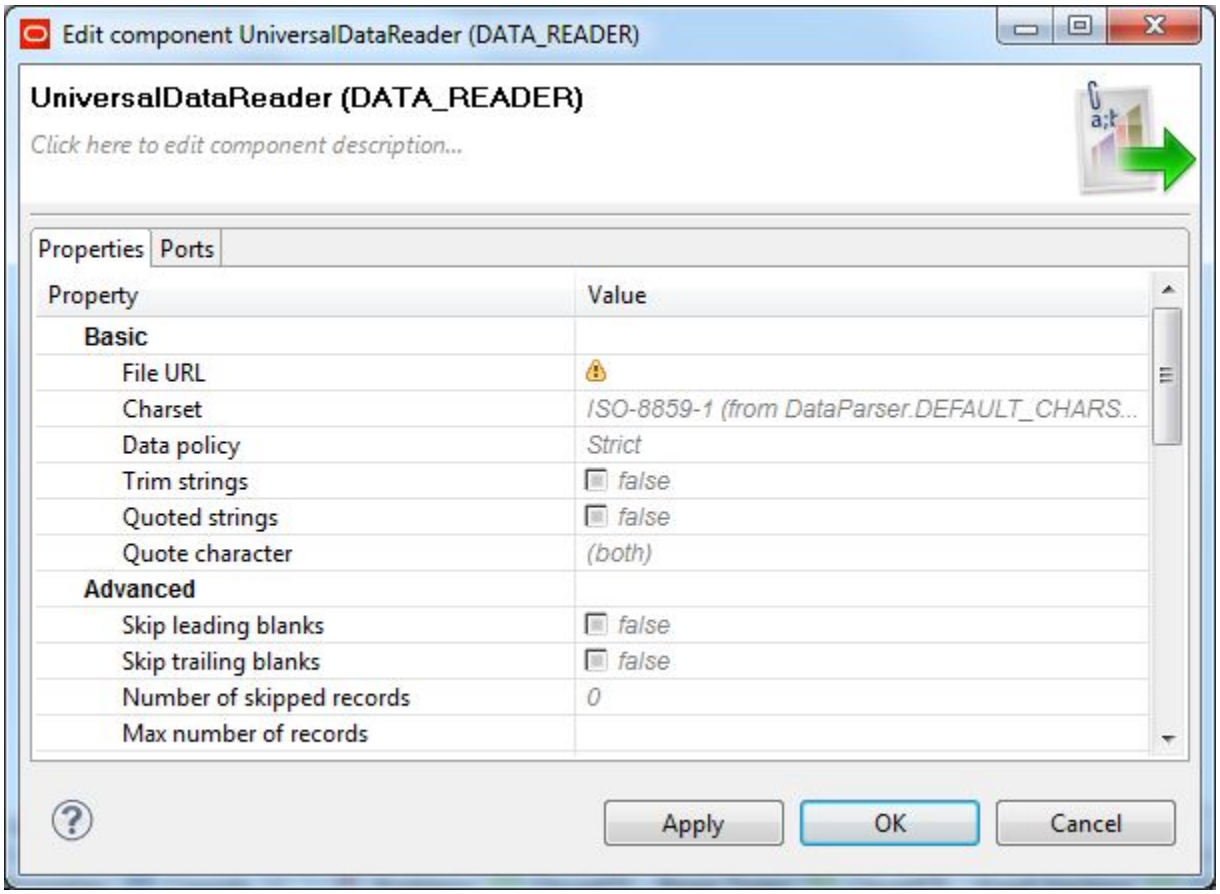
1. The **UniversalDataReader** component.
2. The metadata for the Edge component between the **UniversalDataReader** and **Reformat** components.
3. The metadata for the Edge component between the **Reformat** and **Bulk Add/Replace Records** components.
4. The **Reformat** component.
5. The **Bulk Add/Replace Records** component.

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:



To configure the Reader component:

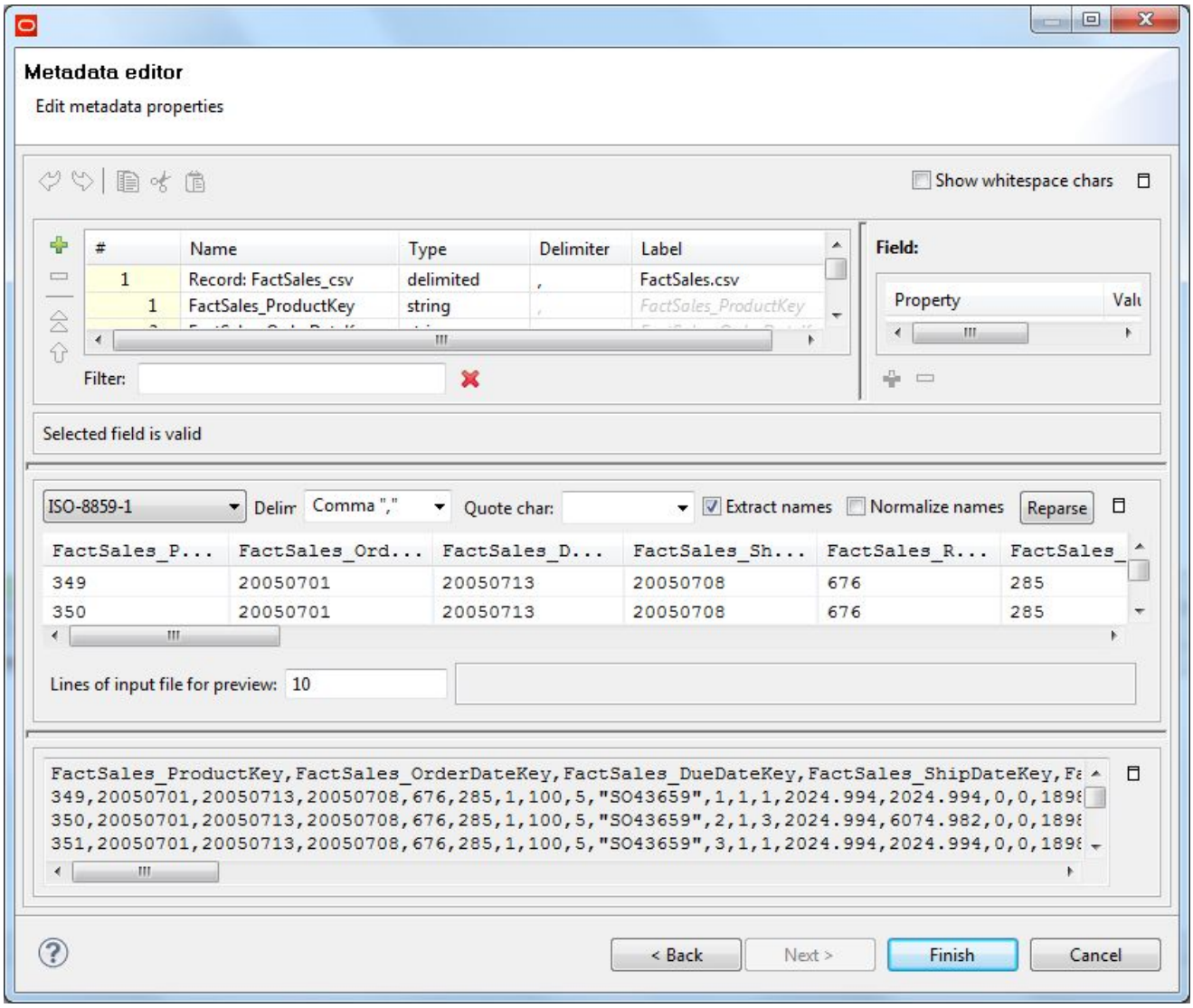
1. In the Graph Editor, double-click the **UniversalDataReader** component to bring up the Reader Edit Component dialog.
2. For the **File URL** property:
   (a) Click inside its Value field, which displays a **...** browse button.
   (b) Click the browse button.
   (c) Click the **Workspace view** tab 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. Optionally, you can use the **Component name** field to provide a customized name (such as "Sales Facts") for this component.
6. Click **OK** to apply your configuration changes to the Reader component.
7. Save the graph.

Configuring metadata for the Reader Edge

The Edge component (between the Reader and Reformat components) has to be associated with a Metadata definition so it knows what fields of data are being passed from the Reader component to the Reformat 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:



You will be using this editor in Steps 6 and 7 of this procedure.

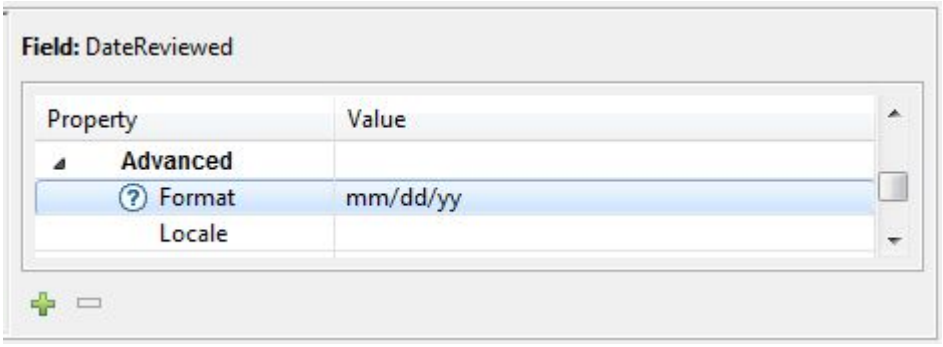
To configure the Metadata definition for the Reader Edge:

1. Right-click on the Edge and select **New metadata>Extract from flat file**.
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 `FactSales.csv` data source 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** Name field and change the default value to a name that is appropriate for your data, such as **FactSales** for the sales transactions data set. In this example, **Record:FactSales** will be the resulting Name value.
   (b) If your source data has date properties, you should set their type to **date** (the type may be set to **string** by the Designer). Then use the **Format** field (in the Field pane on the right) to set the appropriate value, as in this example:



   (c) Verify that the other properties have their property type set correctly.
   For example, change the FactSales_UnitPriceDiscountPct and FactSales_DiscountAmount property types from `integer` to `number` in our sample metadata.
   (d) Verify that all properties have the correct delimiter character set (which is the comma character in our source data).
   (e) When you have input all your changes, click **Finish**.
8. Save the graph.

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

Configuring the Reformat component

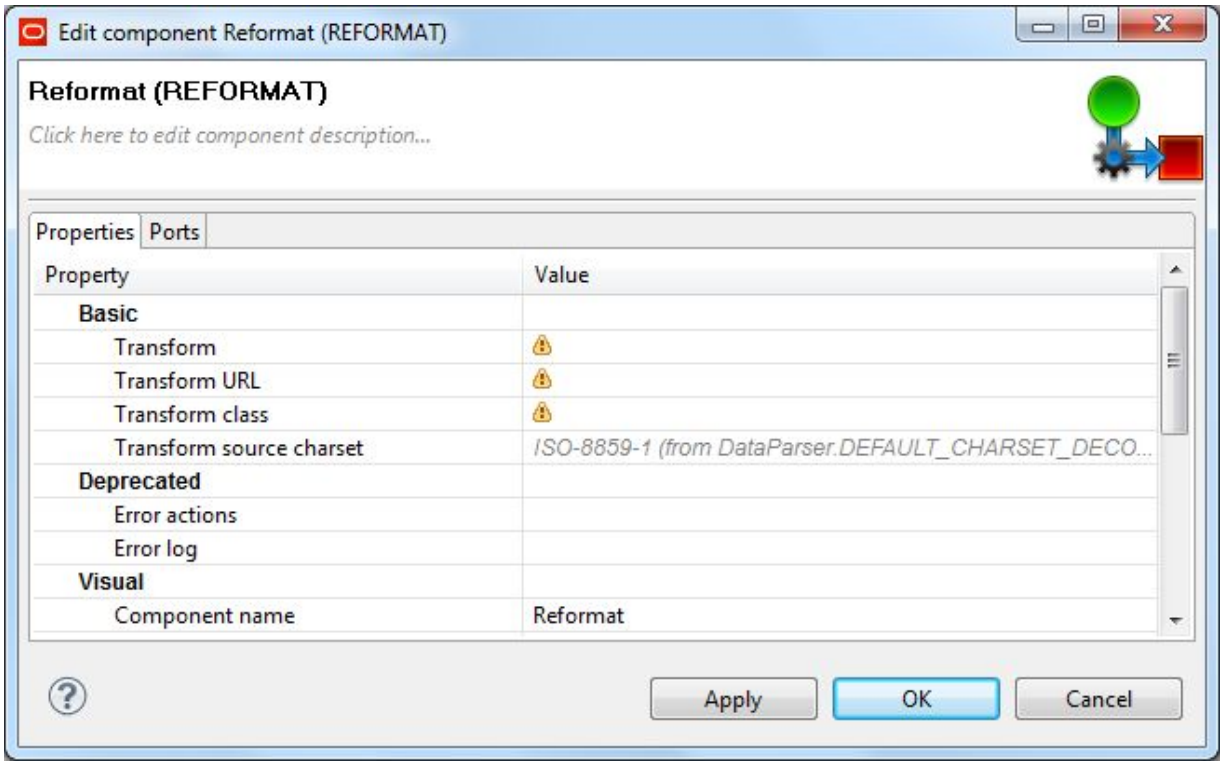
Reformat components are used to transform incoming records and send them to the specified port.

The transformation is done by the CTL function in the **Reformat** component. Return values of the transformation are the numbers of output port(s) to which data records will be sent.

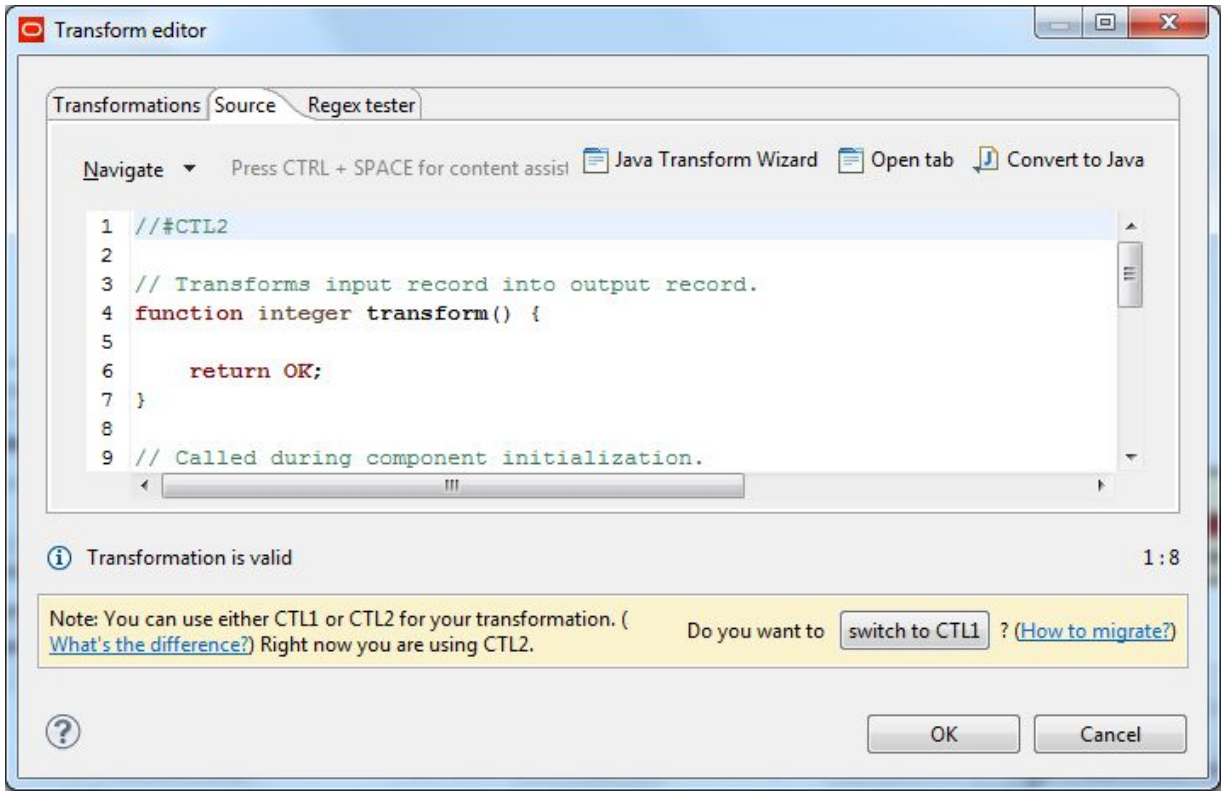
As mentioned earlier, the `FactSales.csv` input file for the graph does not have a field that can be used as the primary-key attribute. Therefore, this **Reformat** component creates a new attribute (named `FactSales_RecordSpec`) by concatenating two existing attributes. After creation, the `FactSales_RecordSpec` attribute is used in the Spec Attribute property of the **Bulk Add/Replace Records** connector.

To configure the **Reformat** component:

1. In the Graph window, double-click the **Reformat** component.
   The Reformat Edit Component dialog is displayed.



2. Single-click in the **Transform** field and then click the **...** button.
   The Transform editor is displayed.
3. Click the **Source** tab in the editor.
   The CTL template for the transform function is shown.



4. Modify the CTL script so that it looks like the following example. Note that the final line of the CTL transformation (just before the `return ALL` line) creates the `FactSales_RecordSpec` attribute.

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

    return ALL;
}
```

   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.

5. When you have finished your edits, click **OK**.

   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.

6. Optionally, you can use the **Component name** field to provide a customized name (such as "Create Spec") for this component.

7. Click **OK** to apply your configuration changes to the component.

8. Save the graph.

The two messages listed above should disappear once you configure the **Reformat** component Edge metadata.

Configuring metadata for the Reformat Edge

The metadata for the Edge component (between the Reformat and the Bulk Add/Replace Records components) also has to configured.

This task will use the same data source file (named `FactSales_RecordSpec`) as the Reader Edge. The main difference is that an additional property will be manually added to the metadata.

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

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 `FactSales.csv` data source 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 Record pane of the Metadata editor, make these settings:
   (a) Click the **Record** Name field and change the default value to a name that is appropriate for your data, such as **FactSalesRecSpec**.
   (b) Make sure that the **Type** fields of these properties match those of the Reader Edge metadata properties.
   (c) Create a new property (for the primary-key attribute) clicking the + button (which creates a new field with a default name such as `field25`), typing the `FactSales_RecordSpec` name of the new property, and leaving the **Type** field as `string`.
   (d) Verify that all properties have the correct delimiter character set (which is the comma character for this source data).
   (e) When you have input all your changes, click **Finish**.
7. Save the graph.

Now the metadata for the two Edge components is set, the next step is to configure the **Bulk Add/Replace Records** connector.

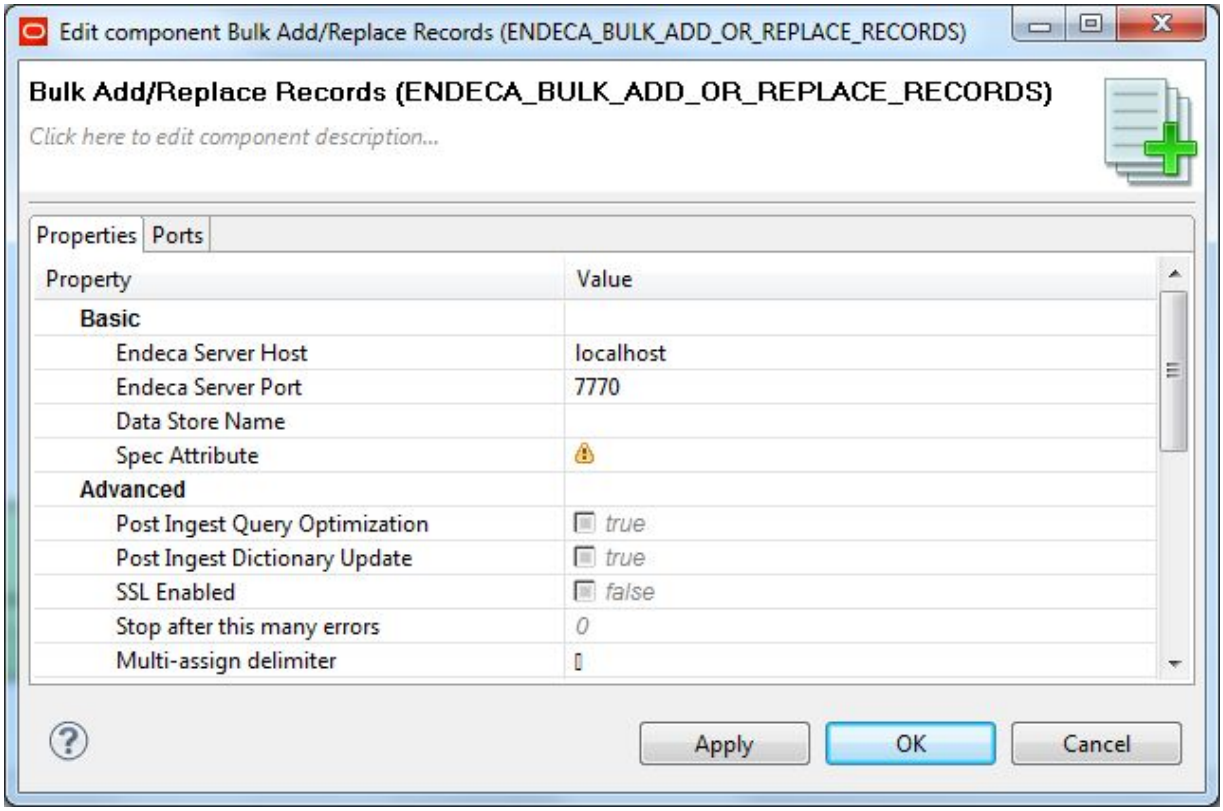
Configuring the Bulk Add/Replace Records connector

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.

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

1. In the Graph editor, double-click the **Bulk Add/Replace Records** component.
   The Edit Component dialog is displayed.



2. In the Writer Edit Component dialog, enter these settings:
   - **Endeca Server Host**: Enter the host name of the machine on which the Endeca Server is running.
   - **Endeca Server Port**: Enter the port which the Endeca Server is listening.
   - **Data Store Name**: Enter the name of the Endeca data store into which the records will be added.
   - **Spec Attribute**: Enter the name of the standard attribute that is the primary key (record spec) for the records.
   - **Post Ingest Query Optimization**: Toggle this field to `false` if you want the data merge into the Endeca data store to be disabled (that is, the data will later be merged according to the Dgraph's merge policy). If the field is left at its default setting of `true`, the data merge is done immediately after the ingest operation finishes.
   - **Post Ingest Dictionary Update**: Toggle this field to `false` if you want the aspell dictionary update to be disabled (you can update the dictionary later via the `updateaspell` administrative operation. If the field is left at its default setting of `true`, the aspell dictionary is updated immediately after the ingest operation finishes.
   - **SSL Enabled**: Toggle this field to `true` only if both the Endeca Server and the Endeca data store are 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

When running the graph, you can use a transaction graph to run this bulk load graph.

A transaction graph uses a **Transaction RunGraph** connector to safely run one or more graphs within the transaction environment of the Dgraph. This connector can start an outer transaction, run the set of graphs so that they succeed or fail as a unit, and finally commit the transaction (or roll it back upon failure).

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 use a transaction graph to bulk load records:

1. Create a transaction graph as described in the chapter *Working with Outer Transaction Graphs on page 22*.
2. Make sure that you have an Endeca Server running on the host and port that are configured in the **Bulk Add/Replace Records** connector and that the Endeca data store has been started.
3. Run the transaction graph using of the methods listed above.

As the graph runs, the process of the graph execution is listed in the Console Tab. The output lists the number of records that were read in by the **UniversalDataReader** component and the number of records that were sent to the Endeca data store by the **Bulk Add/Replace Records** connector.

Chapter 6

Incremental Updates

This chapter describes how to create an Integrator graph that will perform an incremental update of records into an Endeca data store.

Overview of incremental updates

Adding components to the incremental updates graph

Configuring the Reader and the Edge for incremental updates

Configuring the Add/Update Records connector

Running the incremental updates graph

Overview of incremental updates

You can incrementally update the data set in an Endeca data store, 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 Endeca data store.
- 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 Endeca data store, the format of the input will be very similar to the format of the input file for the full index load, as described in *Source data format on page 34*.

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

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

Adding components to the incremental updates graph

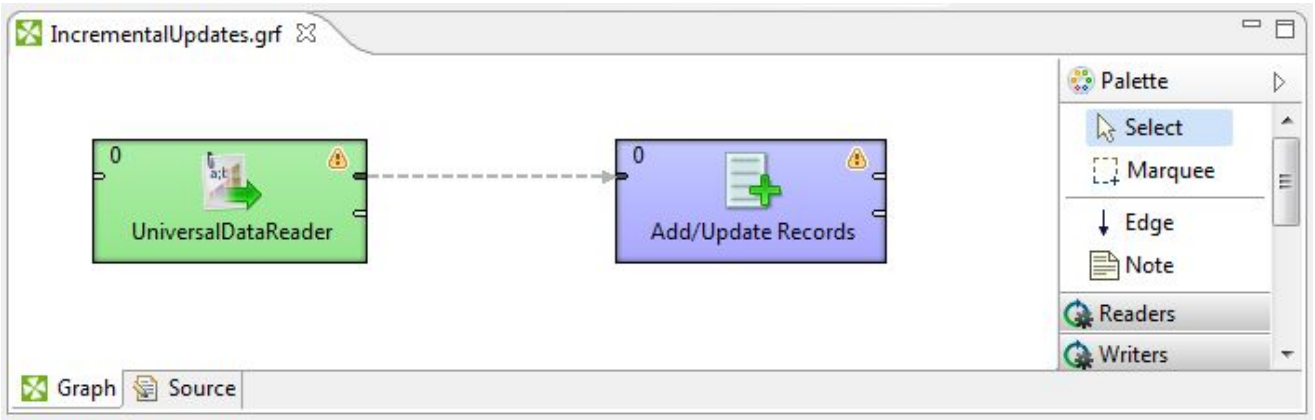
The graph for performing incremental updates requires a reader and the **Add/Update Records** connector.

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 **Discovery** 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 the 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:

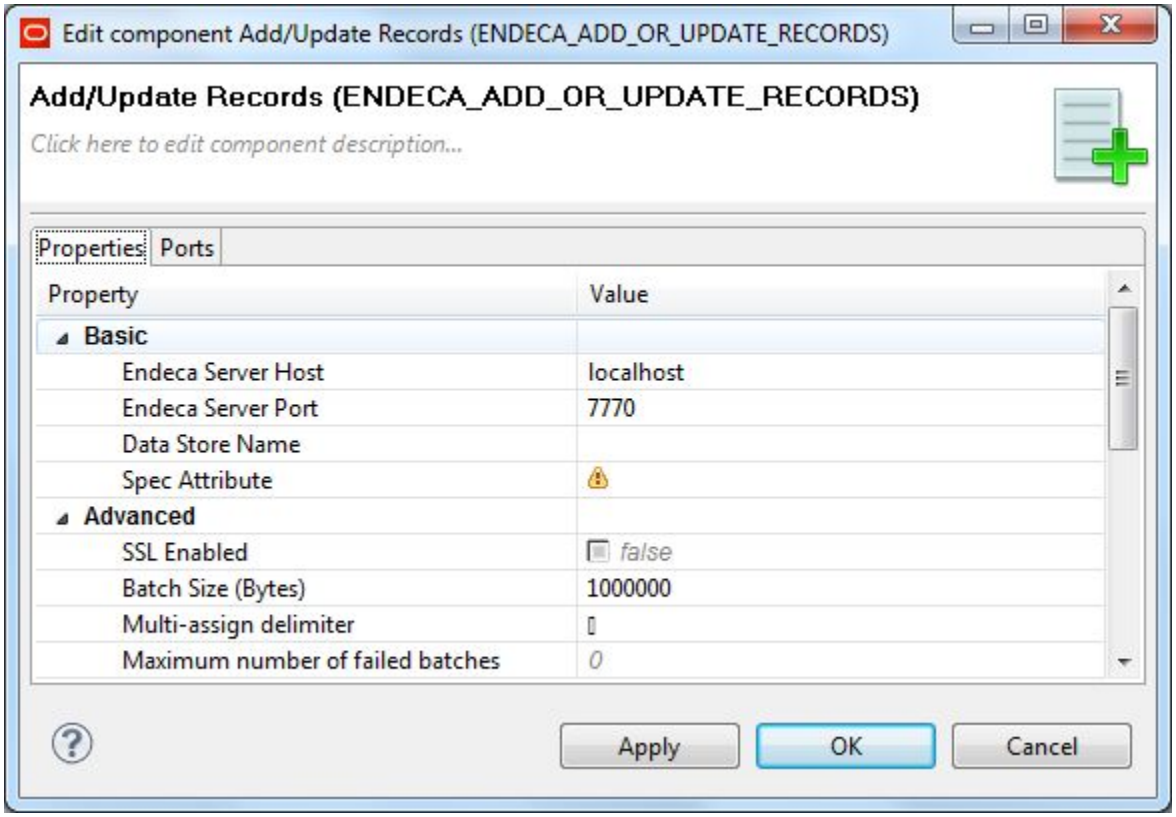
1. To configure the **UniversalDataReader** component for the incremental updates input file, use the same procedure as described in the topic *Configuring the Reader component on page 39*.
   The only difference is that you will be using your incremental updates file as the input file.
2. To configure the Edge component, use the same procedure as described in *Configuring metadata for the Reader Edge on page 41*.
3. When you have finished your configuration, save the graph.

Configuring the Add/Update Records connector

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

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

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



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

1. In the Graph window, double-click the **Add/Update Records** component.
   The Integrator Edit Component dialog is displayed.
2. In the Edit Component dialog, enter these mandatory settings in the Basic section:
   - **Endeca Server Host**: Enter the host name of the machine on which the Endeca Server is running.
   - **Endeca Server Port**: Enter the port on which the Endeca Server is listening.
   - **Data Store Name**: Enter the name of the Endeca data store into which the records will be added.
   - **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 Endeca Server 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 Endeca data store.

To run the graph to load incremental updates:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **Add/Update Records** connector. In addition, make sure that the specified Endeca data store has been started.
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: 04/06/12 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 Endeca data store by the **Add/Update Records** connector.

Chapter 7

Loading the Attribute Schema

This chapter describes how to load your PDR and DDR configuration files into the Endeca data store.

About attribute schema files

Loading the standard attribute schema

Loading the managed attribute schema

Using a transaction graph to load the schemas

About attribute schema files

The attribute schema for your application is defined by the PDR and DDR files in the Endeca data store instance.

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 Dgraph's Bulk Load Interface or Data Ingest Web Service (DIWS) will automatically create the PDRs for your standard attributes, using the system default settings.

However, it is recommended that you create your own PDR and DDR input records and then use the Integrator to load that schema into the Endeca data store. This process uses the **UniversalDataReader** component to read in the schema files and the **WebServiceClient** component to load them into the Endeca data store via the Configuration Web Service.

Loading the standard attribute schema

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 an Endeca data store are:

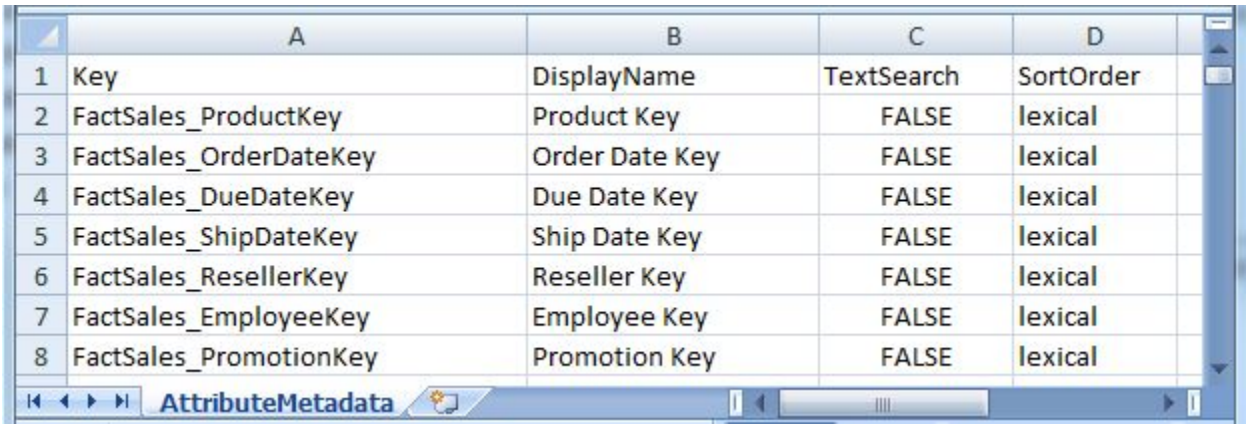
1. Create the PDR input file. Its format is described in this chapter in the topic *Format of the PDR input file on page 54*.
2. Either create a new project or re-use an existing one. (Not described in this chapter, as we will use the same project that was created in the topic *Creating a project on page 33*.)
3. Create a graph and add the **UniversalDataReader**, **Reformat**, **Denormalizer**, and the **WebServiceClient** components. (Described in this chapter.)
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 topic *Running the graph to load records on page 47*.)

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 Endeca data store).

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 sample input file used in this chapter looks like this:



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

AttributeMetadata

The first line (the header row) of the sample file has these header properties:

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

The actual names of the header properties in your input file can be different from the names used here (for example, you can use `AttrName` instead of `Key`). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).

After the header row, the second and following rows in the input file contain the values for the configuration properties.

The header properties map to these PDR properties:

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

The **Reformat** component will take these header properties and values and construct PDRs for the standard attributes.

Keep in mind that you can add additional properties to the input file so that they can be set. As mentioned, any PDR property that is not specified is added with its default value. For information on the system default values for standard attributes, see *Standard attribute default values on page 9*.

Note: Standard attribute names also cannot use hyphens in their names. Although the Dgraph will accept standard attribute names with hyphens, the Designer will not. Therefore, if you have a standard attribute name such as "Sales-Type", make sure you remove the hyphen from the name.

updateProperties operation

You can use the Configuration Service's `updateProperties` operation to load the PDR files into the Endeca data store. The operation creates the standard attributes or updates them if they already exist.

The following is an example of an `updateProperties` operation:

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

This sample operation uses two variables:

- The `OUTER_TRANSACTION_ID` variable specifies the outer transaction ID for the request. The variable and its value is stored in the `workspace.prm` file of the Integrator project.
- The `$xmlString` variable contains the various PDRs that have been constructed by a `Reformat` component in the graph.

The operation would be specified in the request structure of a **WebServiceClient** connector, which will then send the request to the Configuration Web Service.

Adding components to the standard attributes schema graph

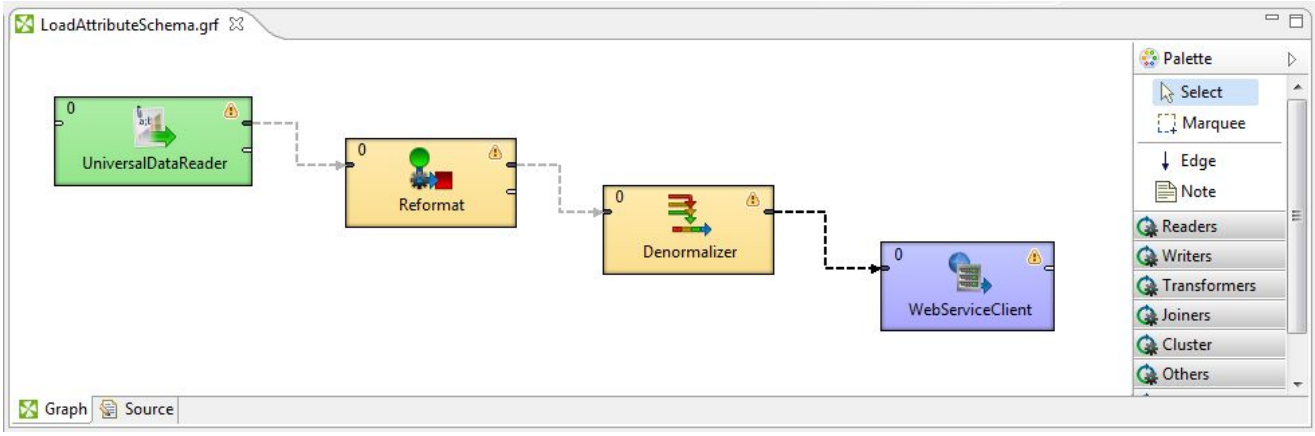
This topic describes the Integrator components that must be added to the graph for your standard attributes schema.

This procedure assumes that you have created an empty graph (our example is named LoadAttributeSchema).

To add components to the graph that loads the standard attribute description files:

1. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag the **UniversalDataReader** component from the **Readers** section.
   (b) Drag the **Reformat** component from the **Transformers** section.
   (c) Drag the **Denormalizer** component from the **Transformers** section.
   (d) Drag the **WebServiceClient** component from the **Others** section.
2. In the Palette pane, click **Edge** and use it to connect the components.
3. From the File menu, click **Save** to save the graph.

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



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. Click **OK** to apply your configuration changes to the Reader component.
4. Save the graph.

Configuring the Reader Edge

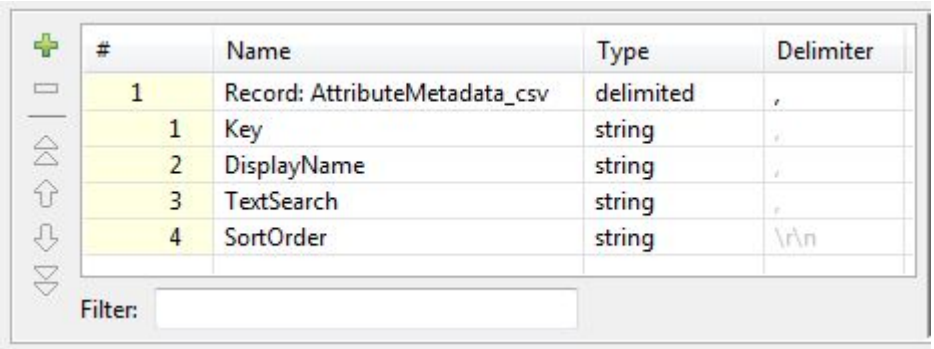
The Edge for the Reader component must be configured with a Metadata definition.

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 Reader 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. In the Flat File dialog, browse for the PDR input file, select it, and click **OK**.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed.

5. 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.
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.
   The correct property names are now displayed in the upper and middle panes of the Metadata Editor, which should look like this example:



#	Name	Type	Delimiter
1	Record: AttributeMetadata_csv	delimited	,
1	Key	string	,
2	DisplayName	string	,
3	TextSearch	string	,
4	SortOrder	string	\r\n

Filter:

7. In the upper pane of the Metadata Editor:
   (a) Optionally, click the **Record** Name field and change the name of the metadata to a more descriptive name.
   (b) Make sure that the **Type** field of all the properties is set to type **string**.
   (c) Verify that the fields have the correct delimiter character set (which is the comma for our example).
8. When you have input all your changes, click **Finish**.
9. Save the graph.

Configuring the Reformat component for standard attributes

A **Reformat** component is used to transform incoming configuration data into a Standard Attribute Description Record.

The transformation is done by this CTL function in the **Reformat** component:

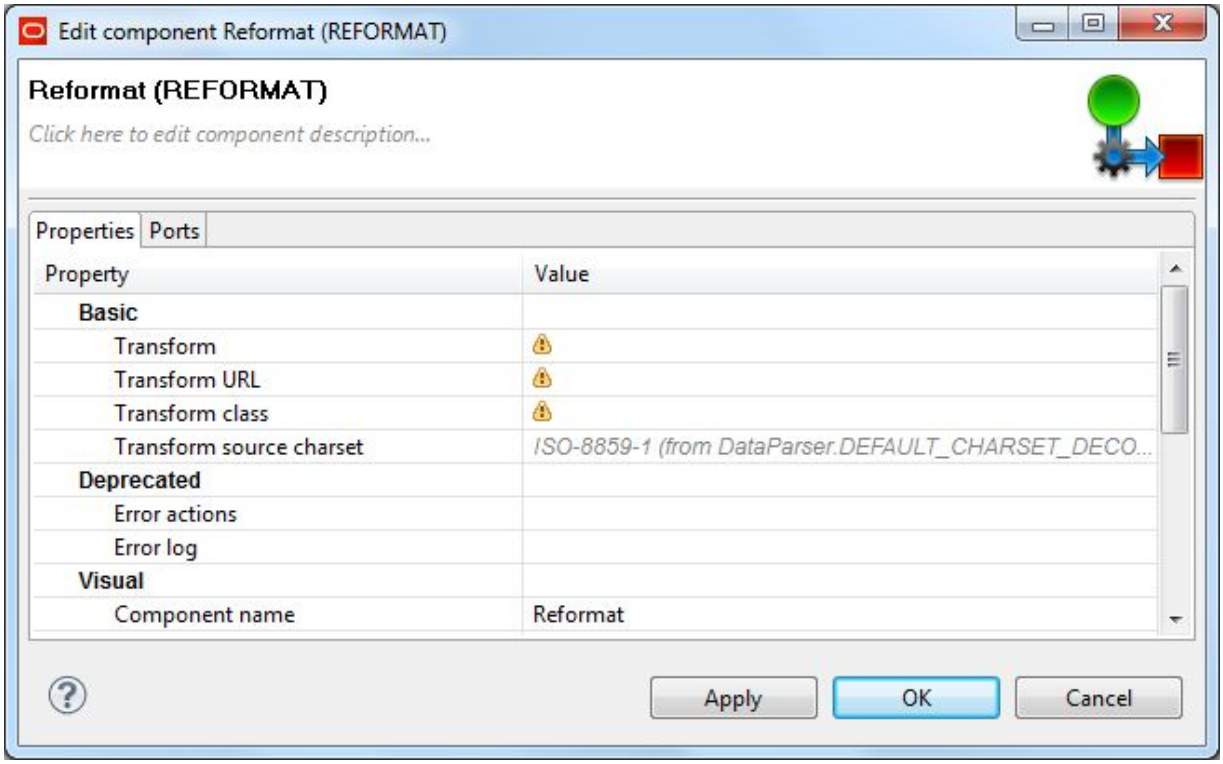
```
integer n = 1;
integer aggrKey = 0;

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

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

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

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

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

  return ALL;
}
```

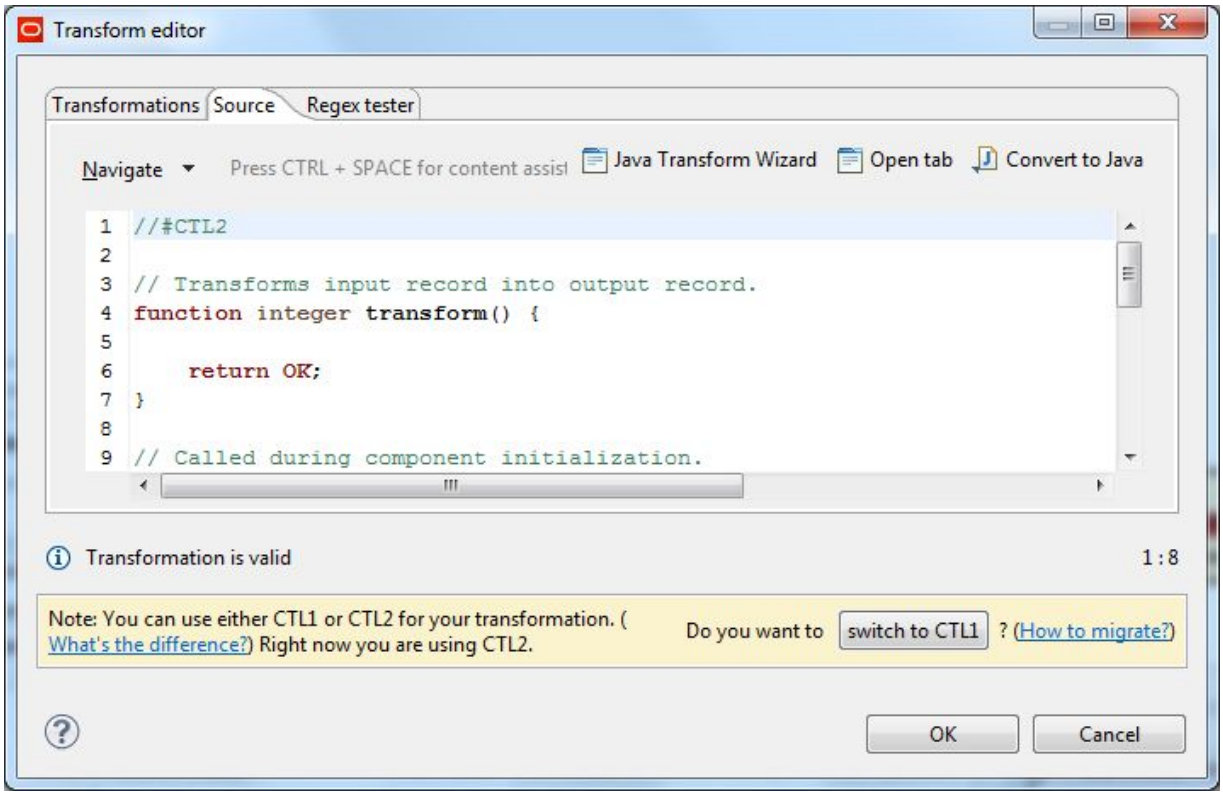
The function builds each Standard Attribute Description Record (SADR) using the configuration data in the input CSV file. Keep in mind that, if you wish, you can add more SADR property definitions; if you do so, be sure to update the input file for the additional input values.

To configure the **Reformat** component in the standard attribute schema graph:

1. In the Graph window, double-click the **Reformat** component.
   The Reformat Edit Component dialog is displayed.



2. Single-click in the **Transform** field and then click the **...** button.
   The Transform editor is displayed.
3. Click the **Source** tab in the editor.
   The CTL template for the transform function is shown.



4. Modify the CTL script so that it looks like the CTL example above.

   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.

5. When you have finished your changes in the Transform editor, click **OK**.

   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.

6. Optionally, you can change the **Component name** field to provide a customized name (such as "Transform Attribute Metadata") for this component.

7. Click **OK** to apply your configuration changes.

8. Save the graph.

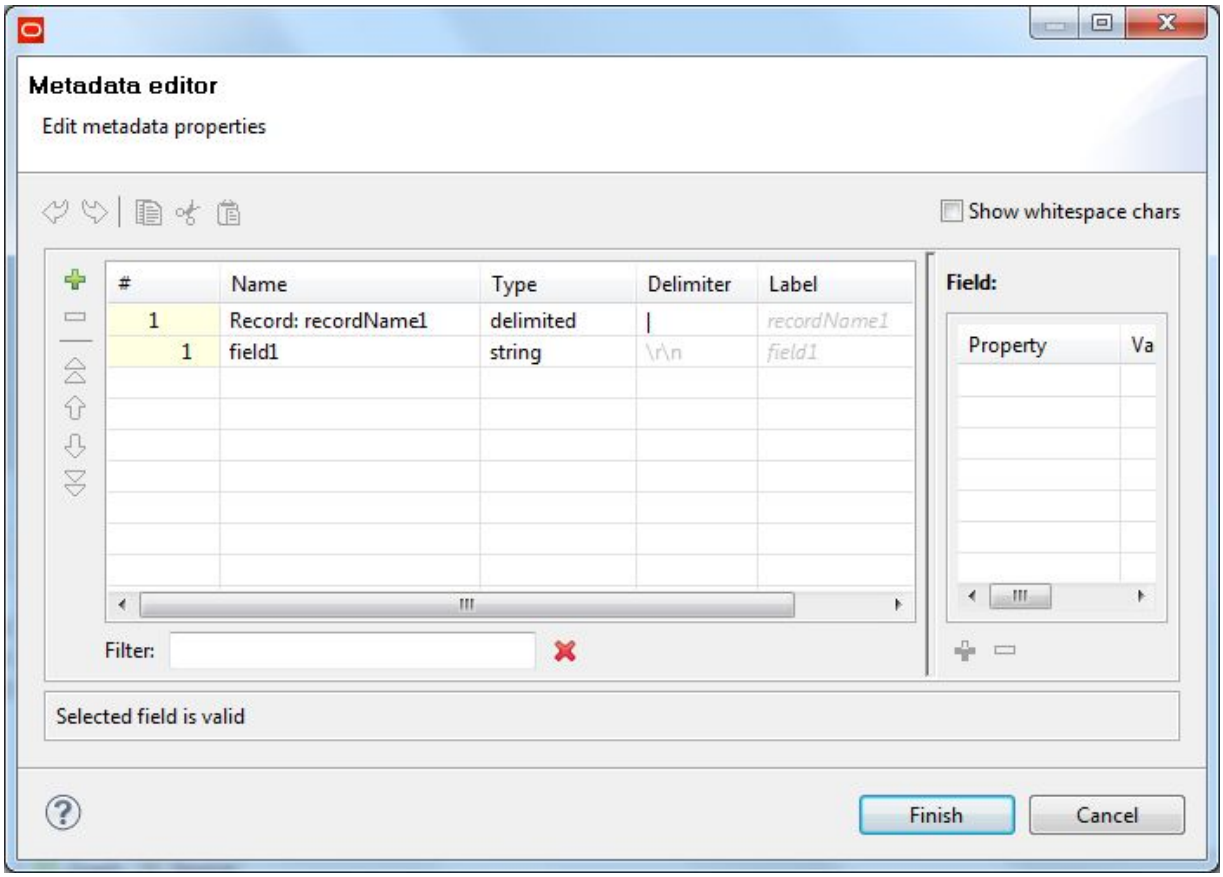
The two messages listed above should disappear once you configure the **Reformat** component Edge metadata.

Configuring the Reformat Edge

This task describes how to configure the Edge component that connects the **Reformat** and **Denormalizer** components.

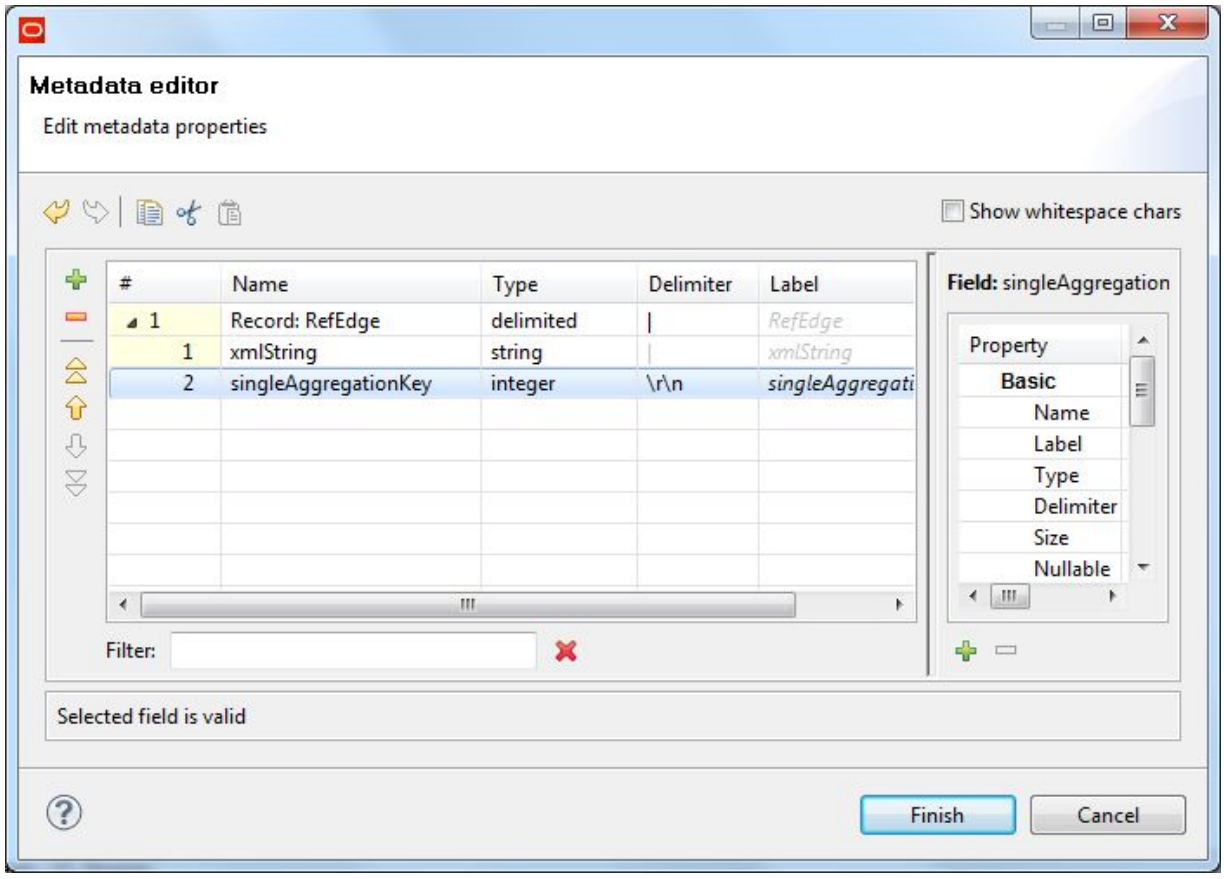
To configure the **Reformat** component's Edge in the attribute schema graph:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. For the other fields:
   (a) Change the **field1** name to `xmlString` and leave its **Type** as `string`.
   (b) Add a new field by using the + (plus sign control). Name the field `singleAggregationKey` and set its **Type** as `integer`.

   At this point, the Metadata editor should look like this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the Denormalizer component

A **Denormalizer** component is used to create a single output record for a group of input records defined by the key.

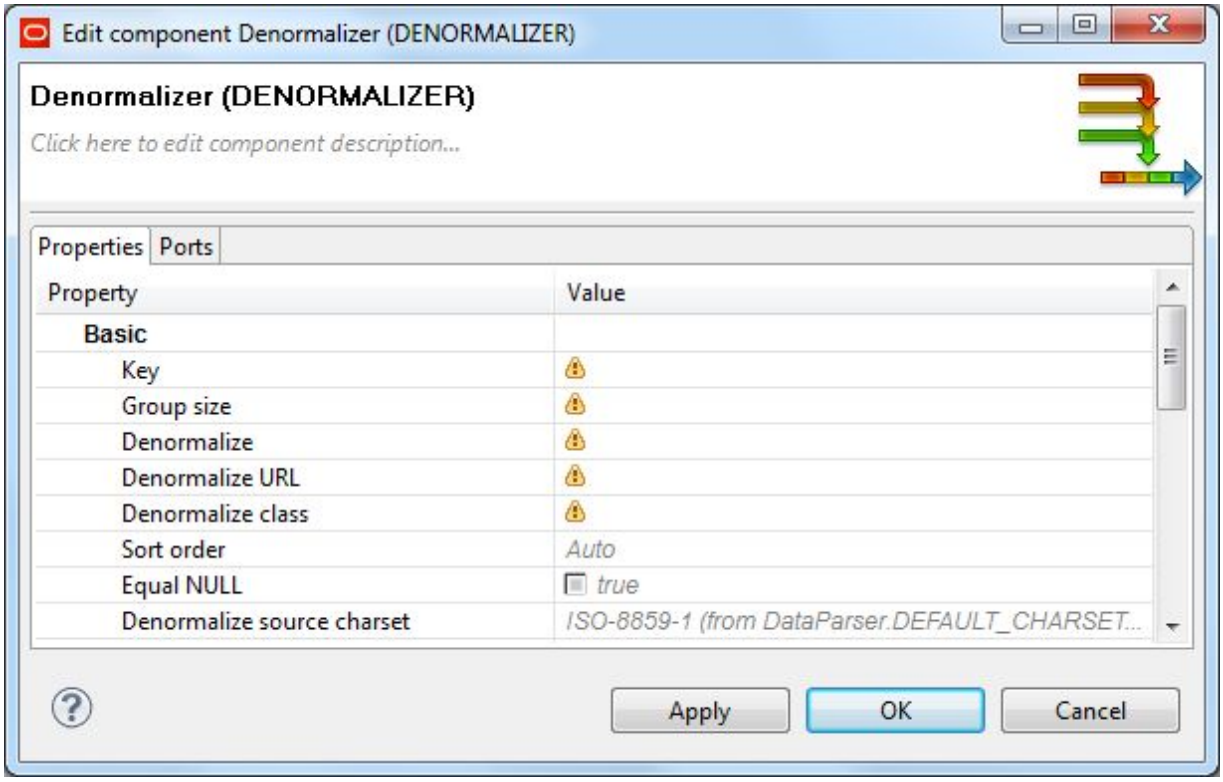
The transformation is done by these CTL functions in the **Denormalizer** component:

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

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

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

```
}
```

To configure the **Denormalizer** component in the attribute schema graph:

1. In the Graph window, double-click the **Denormalizer** component.
   The Denormalizer Edit Component dialog is displayed.



2. Single-click in the **Key** field and then click the **...** button.
   The Edit Key dialog is displayed.
3. In the **Fields** pane of the Edit Key dialog, select **singleAggregationKey** and move it to the **Key parts** pane by clicking the right-arrow button. Click **OK** to apply your change.
4. Single-click in the **Denormalize** field and then click the **...** button.
   The Transform editor is displayed.
5. In the **Source** tab of the editor, modify the CTL script so that it looks like the example above.
   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.
6. When you have finished your edits, click **OK**.
   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.
7. Optionally, you can use the **Component name** field to provide a customized name (such as "Load Schema") for this component.
8. Click **OK** to apply your configuration changes.
9. Save the graph.

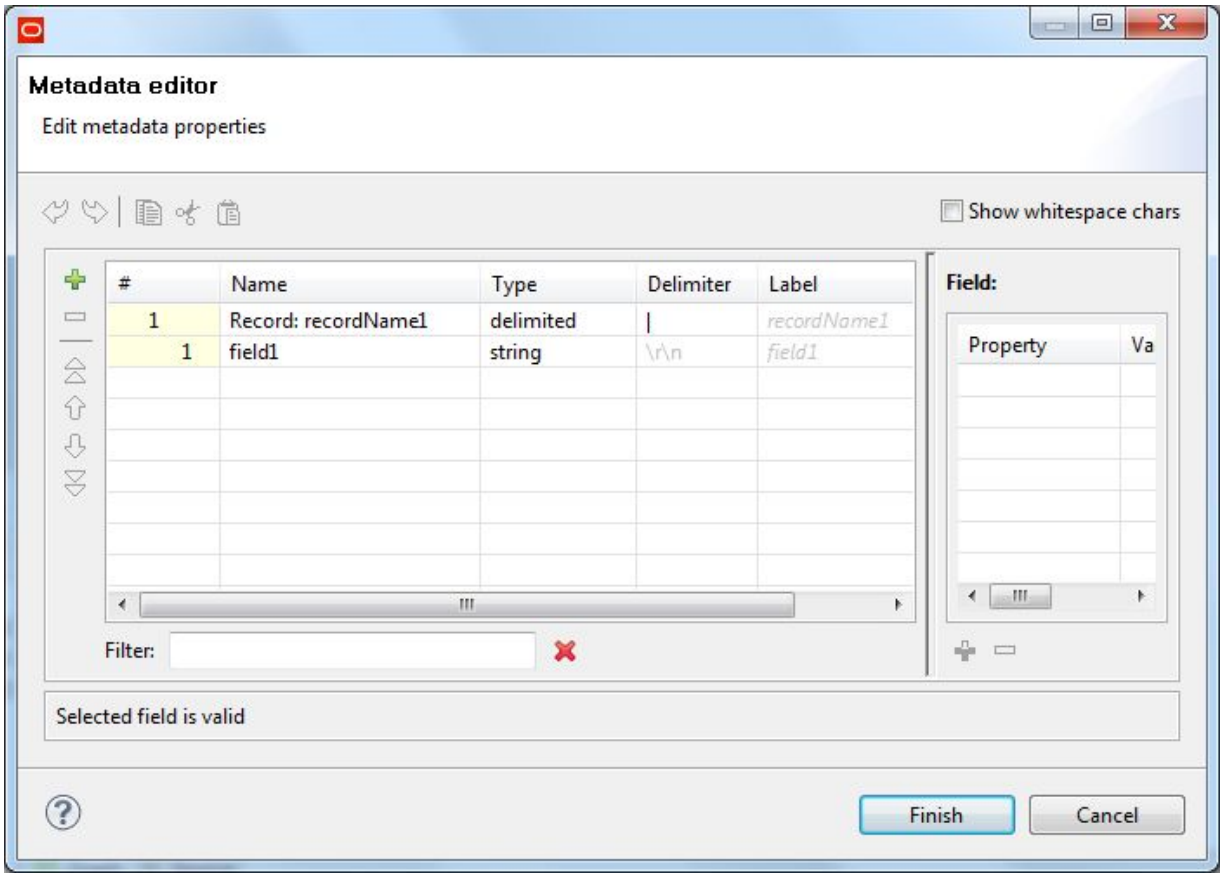
The two messages listed above should disappear once you configure the **Denormalizer** component Edge metadata.

Configuring the Denormalizer Edge

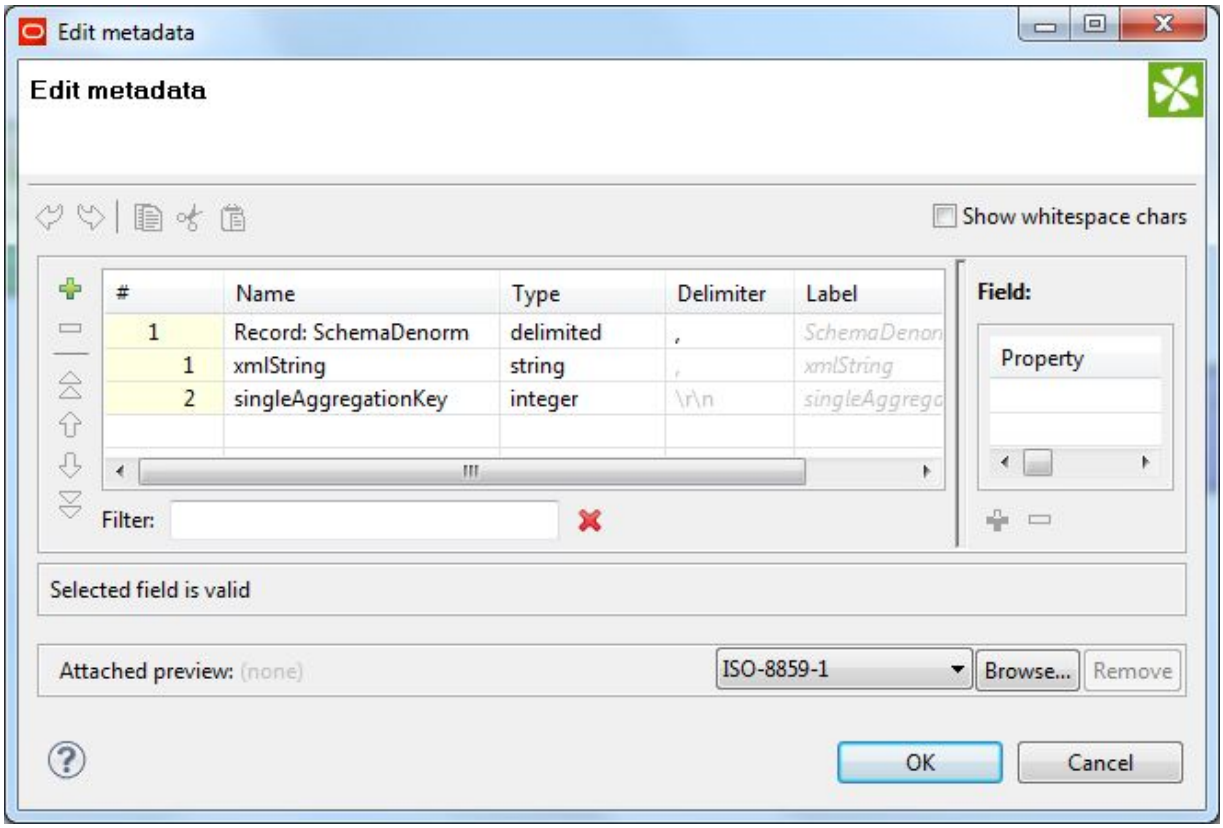
This task describes how to configure the Edge component that connects the **Denormalizer** and **WebServiceClient** components.

To configure the **Denormalizer** component's Edge in the attribute schema graph:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. For the other fields:
   (a) Change the **field1** name to `xmlString` and leave its **Type** as `string`.
   (b) Add a new field by using the + (plus sign control). Name the field `singleAggregationKey` and set its **Type** as `integer`.
   At this point, the Metadata editor should look like this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the WebServiceClient component for standard attributes

This topic describes how to configure the **WebServiceClient** connector for loading standard attribute metadata via the Configuration Web Service.

This procedure assumes that you have created a graph and added the **WebServiceClient** component.

The procedure also assumes that you are specifying an outer transaction ID with the request and that your `workspace.prm` file has defined the ID in the `OUTER_TRANSACTION_ID` variable, as in this example:

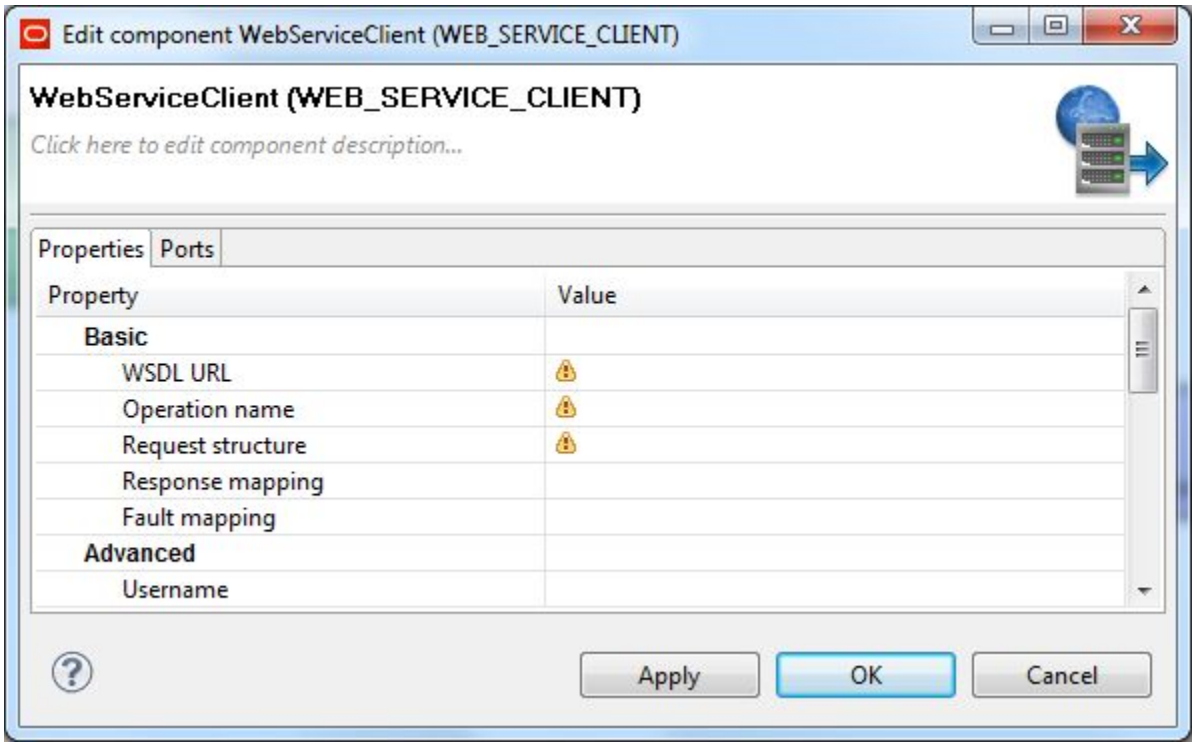
```
OUTER_TRANSACTION_ID=
```

To configure the **WebServiceClient** connector for standard attribute metadata:

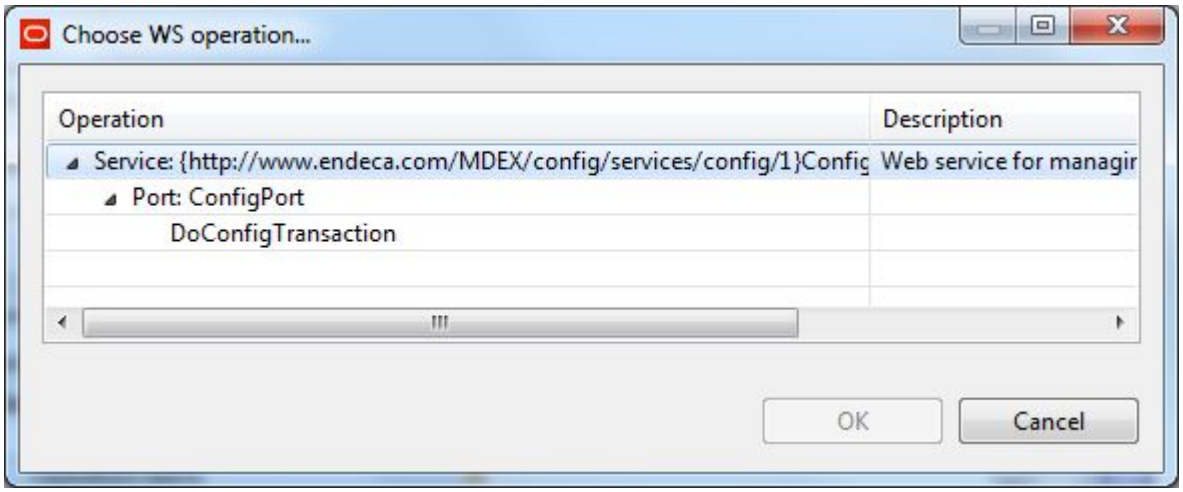
1. Make sure that the Endeca data store instance is running and its Configuration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

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

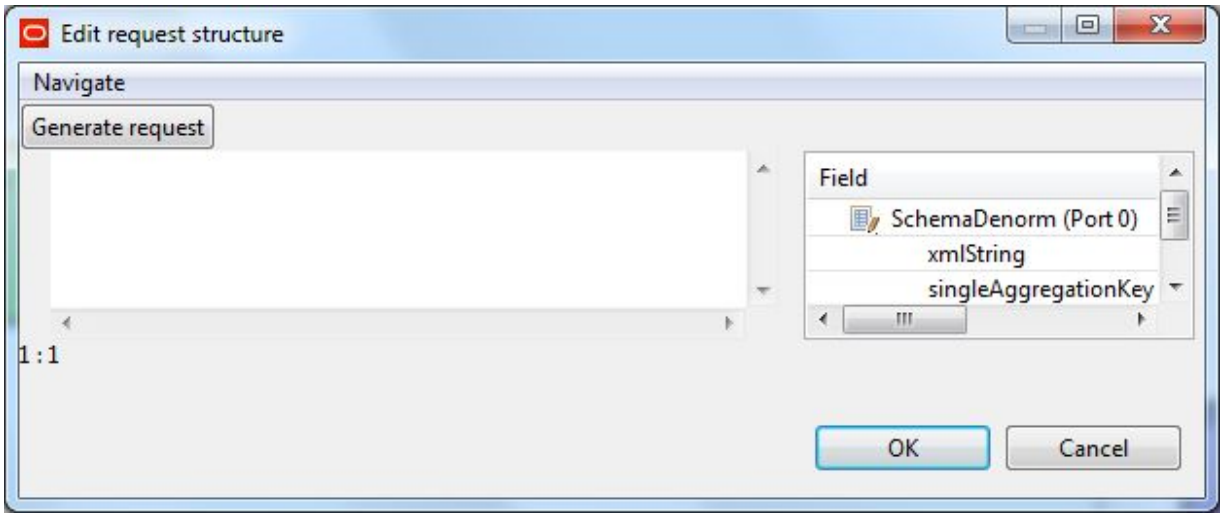
   The URL command returns the WSDL of the Web service.
2. In the Graph window, double-click the **WebServiceClient** component.
   The Writer Edit Component dialog is displayed.



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:



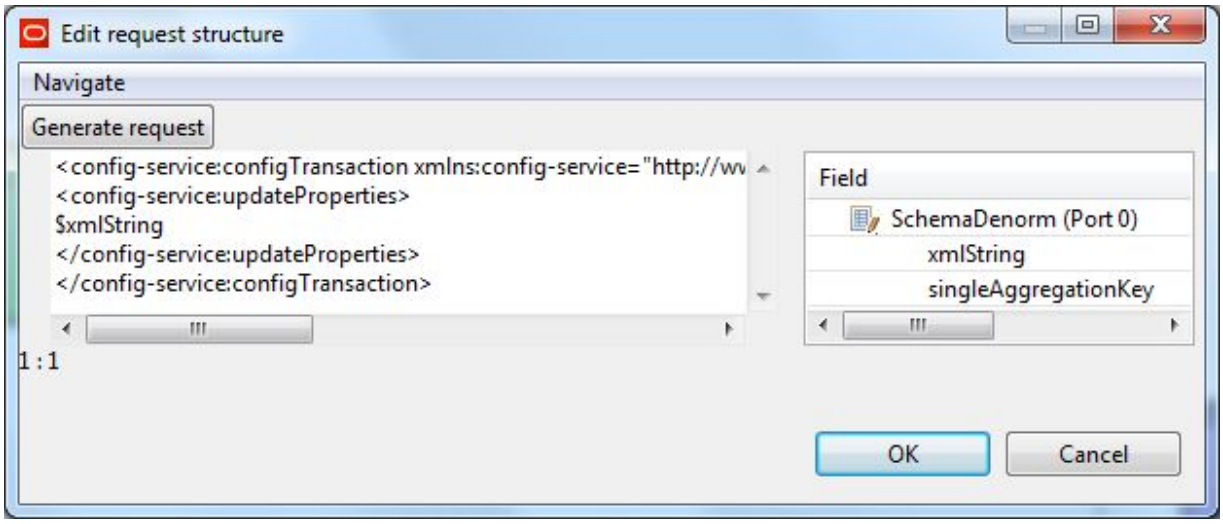
5. In the **Choose WS operation** dialog, select **DoConfigTransaction** and then click **OK**.
   The name of the Web service operation is entered in the **Operation name** field.
6. Click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the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



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

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

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



8. After adding the request text in the **Edit request structure** dialog, click **OK**.
9. Optionally, you can use the **Component name** field to provide a customized name for this component.
10. When you have input all your changes, click **OK**.
11. Save the project.

Instead of running this graph directly, it is recommended that you create a transaction graph (with a **Transaction RunGraph** connector) with this LoadAttributeSchema graph as its child graph, and then run the transaction graph.

Loading the managed attribute schema

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 Dgraph are as follows:

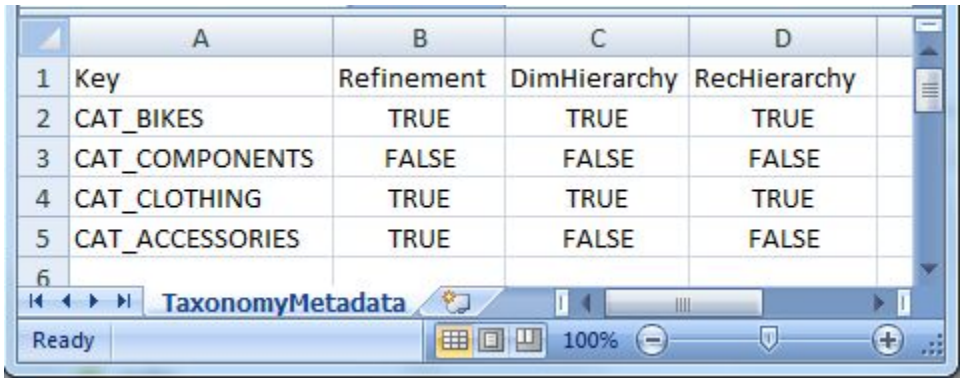
1. Create the managed attributes input file.
2. Either create a new project or re-use an existing one.
3. Create a graph and add the **UniversalDataReader**, **Reformat**, **Denormalizer**, and **WebServiceClient** components.
4. Configure the components.
5. Run this graph as part of a transaction graph.

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 Dgraph).

Format of the DDR input file

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

The `TaxonomyMetadata.csv` sample input file used for the managed attributes looks like this:



The first line (the header row) of the sample file has these header properties:

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

The actual names of the header properties can be different from the names used here. The properties are delimited (for example, by the comma in the sample CSV file). After the header row, the second and following rows in the input file contain the values for the configuration properties.

The header properties map to these DDR properties:

Input Header Property	Maps to PDR Property
`Key`	`mdex-dimension_Key`
`Refinement`	`mdex-dimension_EnableRefinements`
`DimHierarchy`	`mdex-dimension_IsDimensionSearchHierarchical`
`RecHierarchy`	`mdex-dimension_IsRecordSearchHierarchical`

The **Reformat** component will take these header properties and values and construct DDRs for the managed attributes.

updateDimensions operation

The Configuration Service's `updateDimensions` operation can load the DDR files into the Endeca data store. The operation creates the standard attributes or updates them if they already exist. The default DDR properties are listed in the topic *Managed attribute default values on page 10*.

The following is an example of an `updateDimensions` operation:

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

This sample operation uses two variables:

- The `OUTER_TRANSACTION_ID` variable specifies the outer transaction ID for the request. The variable and its value is stored in the `workspace.prm` file of the Integrator Designer project.
- The `$xmlString` variable contains the various DDRs that have been constructed by a `Reformat` component in the graph.

The operation would be specified in the request structure of a **WebServiceClient** connector, which will then send the request to the Configuration Web Service.

Adding components to the managed attributes schema graph

This topic describes the Integrator components that must be added to the graph for your managed attributes schema.

This procedure assumes that you have created an empty graph for your managed attributes schema. This graph will use the same components as the graph for the standard attributes schema.

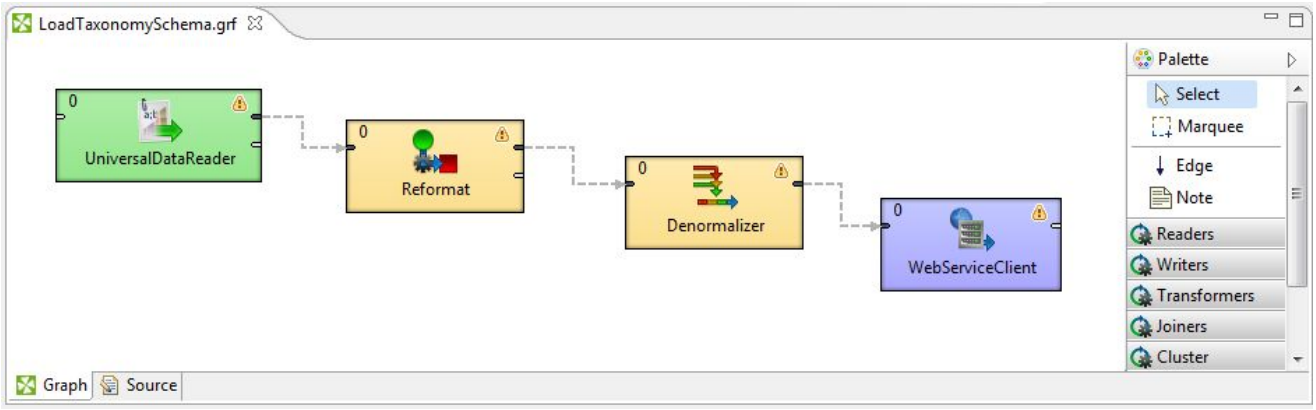
To add components to the graph that loads the managed attribute description files:

1. In the Palette pane, drag the following components into the Graph Editor:
   (a) **UniversalDataReader** component

(b) **Reformat** component
(c) **Denormalizer** component
(d) **WebServiceClient** component

2. In the Palette pane, click **Edge** and use it to connect the components.
3. Save the graph.

The resulting graph should like this:



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. 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 *Configuring the Reader for the PDR input file on page 56*.
   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 *Configuring the Reader Edge on page 56*.
   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 Reformat component for managed attributes

A **Reformat** component is used to transform incoming configuration data into a Managed Attribute Description Record.

The transformation is done by this CTL function in the **Reformat** component:

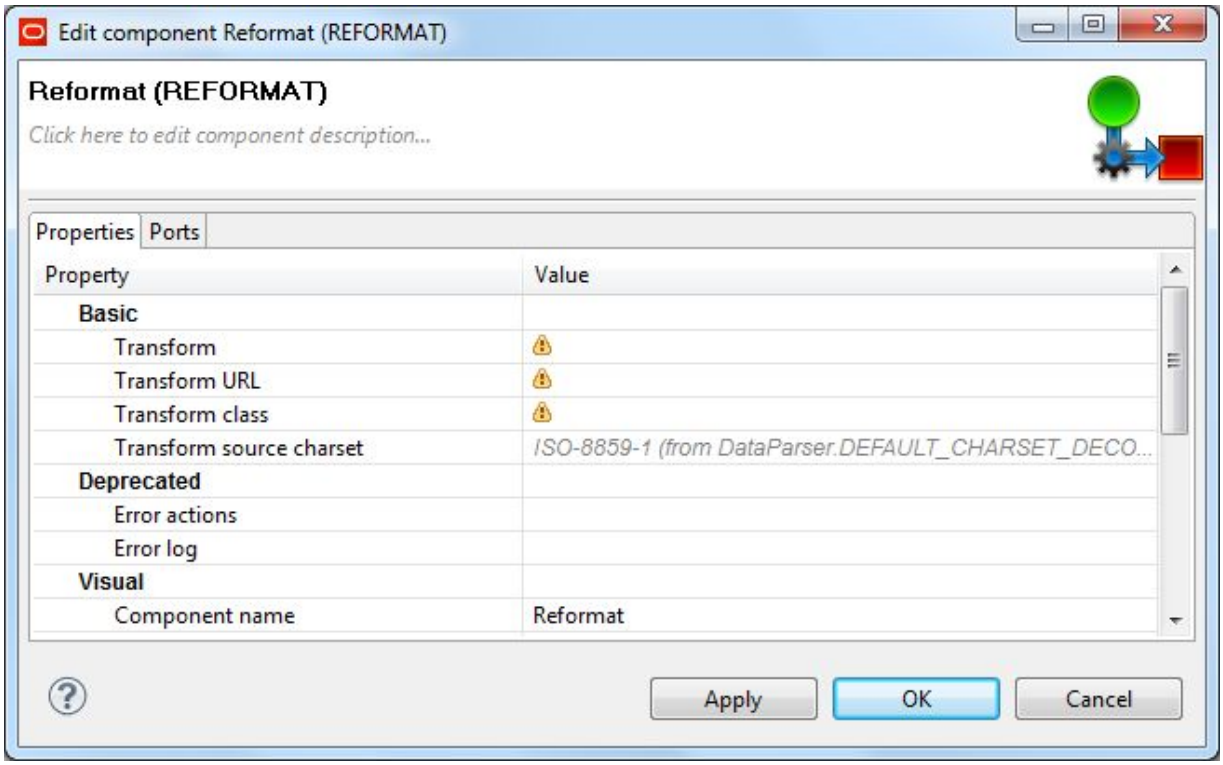
```
//#CTL2

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

```
// Transforms input record into output record.
function integer transform() {
   string maBool = "";
   string maRecord = "<mdex:record xmlns=\"\">";
   maRecord = maRecord + "<mdex-dimension_Key>" + $0.Key + "</mdex-dimension_Key>";

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

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

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

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

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

   return ALL;
}
```

The function builds each Managed Attribute Description Record using the configuration data in the input CSV file.

To configure the **Reformat** component in the managed attribute schema graph:

1. In the Graph window, double-click the **Reformat** component.
   The Reformat Edit Component dialog is displayed.



2. Single-click in the **Transform** field and then click the **...** button.
   The Transform editor is displayed.
3. Click the **Source** tab in the editor.
   The CTL template for the transform function is displayed.
4. Modify the CTL script so that it looks like the example above.
   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.
5. When you have finished your changes in the Transform editor, click **OK**.
   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.
6. Optionally, you can change the **Component name** field to provide a customized name (such as "Transform Attribute Metadata") for this component.
7. Click **OK** to apply your configuration changes.
8. Save the graph.

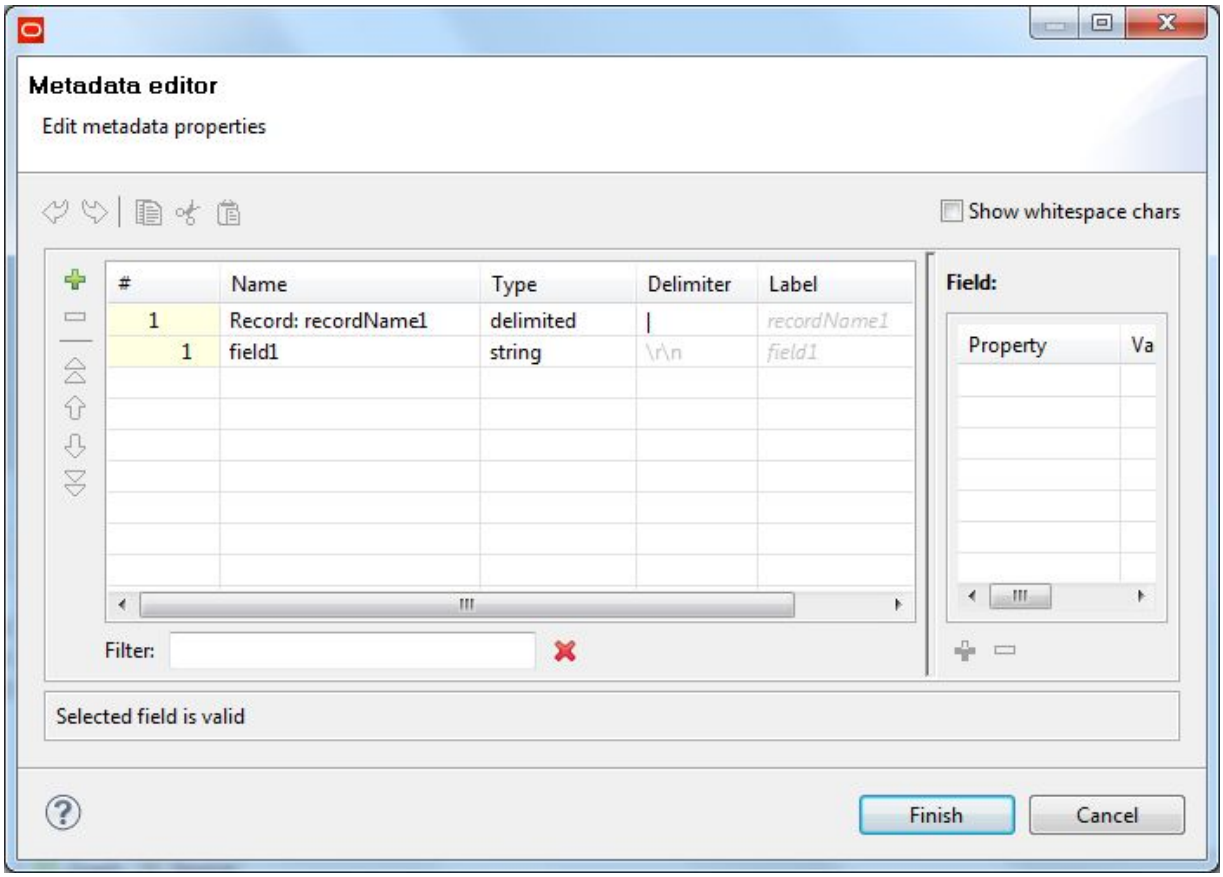
The two messages listed above should disappear once you configure the **Reformat** component Edge metadata.

Configuring the Reformat Edge

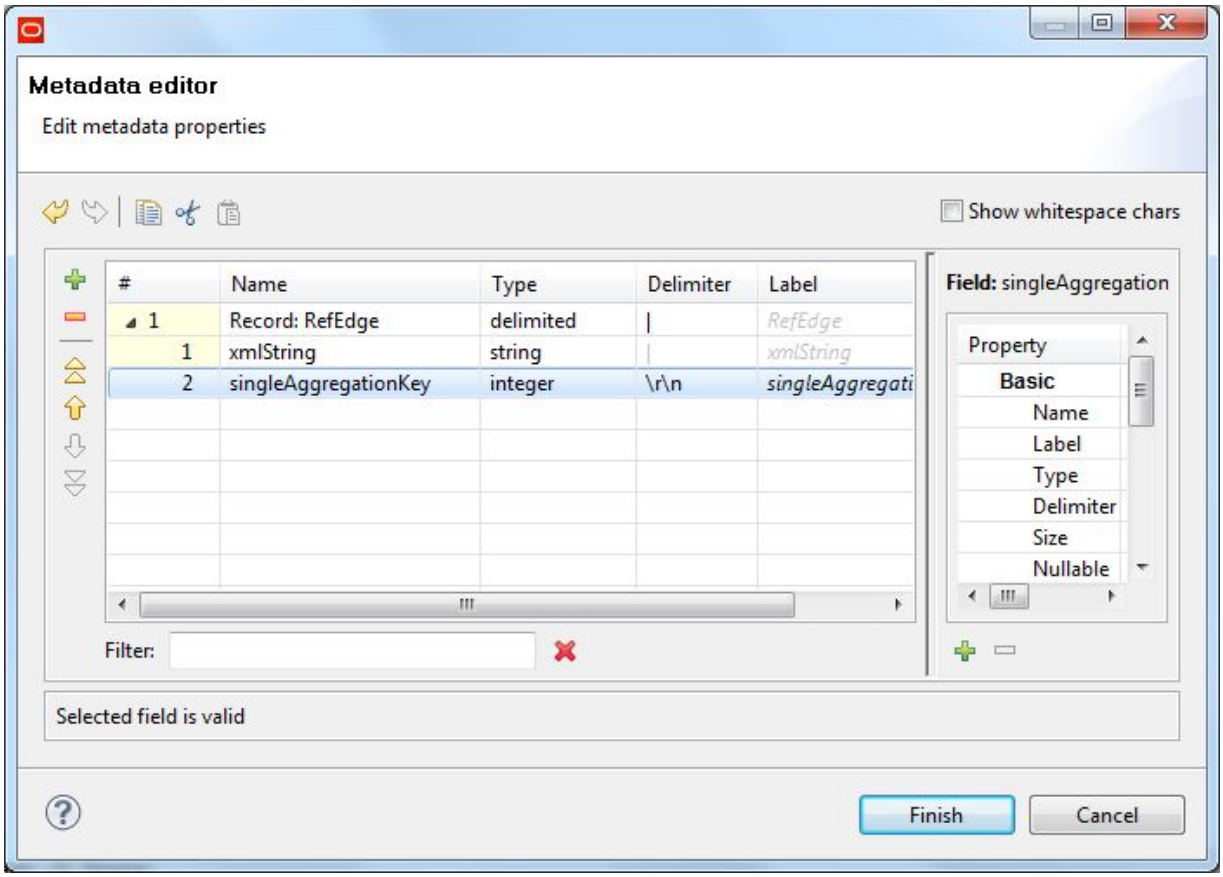
This task describes how to configure the Edge component that connects the **Reformat** and **Denormalizer** components.

To configure the **Reformat** component's Edge in the attribute schema graph:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. For the other fields:
   (a) Change the **field1** name to `xmlString` and leave its **Type** as `string`.
   (b) Add a new field by using the + (plus sign control). Name the field `singleAggregationKey` and set its **Type** as `integer`.
   At this point, the Metadata editor should look like this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the Denormalizer and the Edge for DDRs

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

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

1. To configure the **Denormalizer** component for managed attributes, use the same procedure as described in the topic *Configuring the Denormalizer component on page 61*.
2. To configure the Edge component, use the same procedure as described in the topic *Configuring the Denormalizer Edge on page 63*.
3. When you have finished your configuration, save the graph.

Configuring the WebServiceClient component for managed attributes

This topic describes how to configure the **WebServiceClient** component for loading managed attribute metadata.

This procedure assumes that you have created a graph and added the **WebServiceClient** component.

To configure the **WebServiceClient** connector for managed attribute metadata:

1. Follow steps 1-6 in the configuration procedure described in the topic *Configuring the WebServiceClient component for standard attributes on page 64*.
2. Replace Step 7 in that topic by adding this text to the **Generate request** field in the **Edit request structure** dialog:

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

3. Continue with Steps 8-11 of the topic.

Instead of running this graph directly, it is recommended that you create a transaction graph (with a **Transaction RunGraph** connector) with this LoadTaxonomySchema graph as its child graph, and then run the transaction graph.

Using a transaction graph to load the schemas

You can run the two schema graphs with a transaction graph.

A transaction graph uses a **Transaction RunGraph** connector to safely run one or more graphs within the outer transaction. This connector can start an outer transaction, run the set of graphs so that they succeed or fail as a unit, and finally commit the transaction (or roll it back upon failure).

To use a transaction graph to load the two attribute schemas:

1. Create a transaction graph as described in the chapter *Working with Outer Transaction Graphs on page 22*.
   Note that the chapter uses the two attribute schema graphs as examples.
2. Run the transaction graph as you would run any other graph.

The transaction graph will first run the standard attribute schema graph and then the managed attribute schema graph.

Chapter 8

Loading Configuration Files

This chapter describes how to load the Global Configuration Record and the configuration documents for an Endeca data store instance.

Types of configuration documents

Loading the configuration documents

Loading the GCR

Types of configuration documents

An Endeca data store offers a rich set of configuration documents that allow you to customize your Information Discovery implementation.

The Endeca data store configuration documents are the mechanism for implementing a number of Endeca features such as search and ranking. The configuration documents are created automatically by an Endeca Server `create-ds` command with a set of defaults that are described in the following topics. The configuration documents are stored in the Endeca data files and loaded into the Dgraph process at startup.

The configuration documents are as follows:

Configuration Document	Purpose
`dimsearch_config`	Configures attributes (both Standard Attributes and Managed Attributes) for value search.
`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 Dgraph.
`thesaurus`	The thesaurus allows the system to return matches for related concepts to words or phrases contained in user queries.

Configuration Document	Purpose
`en_word_forms_collection`	Sets the word forms (from one language) for the stemming feature. For information on replacing the current stemming file, see the chapter *Loading Stemming Files on page 96*.

Recommended order for loading the configuration documents

The recommended order of loading is:

1. Load the attribute schema (PDRs and DDRs) first. The schema should be loaded first because it creates the Endeca standard attributes and managed attributes that are 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. `stop_words` document
6. `thesaurus` document
7. `en_word_forms_collection` (only if you have changed the stemming language)

Global Configuration Record

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

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

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_EnableValueSearchWildcard>
  <mdex-config_MergePolicy>aggressive</mdex-config_MergePolicy>
  <mdex-config_SearchChars>+_</mdex-config_SearchChars>
  <mdex-config_SpellingRecordMinWordOccur>2</mdex-config_SpellingRecordMinWordOccur>
  <mdex-config_SpellingRecordMinWordLength>4</mdex-config_SpellingRecordMinWordLength>
  <mdex-config_SpellingRecordMaxWordLength>24</mdex-config_SpellingRecordMaxWordLength>
  <mdex-config_SpellingDValMinWordOccur>5</mdex-config_SpellingDValMinWordOccur>
```

```
  <mdex-config_SpellingDValMinWordLength>3</mdex-config_SpellingDValMinWordLength>
  <mdex-config_SpellingDValMaxWordLength>20</mdex-config_SpellingDValMaxWordLength>
</mdex:record>
```

This GCR:

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

You can create the file in a text editor.

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.

Sample dimsearch_config document

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

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

As mentioned above, the ProductRelRank strategy must have been configured previously with the `relrank_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:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
<config-service:OuterTransactionId>${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
<config-service:putConfigDocuments
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="dimsearch_config">
<DIMSEARCH_CONFIG>
  $xmlString
</DIMSEARCH_CONFIG>
</mdex:configDocument>
</config-service:putConfigDocuments>
</config-service:configTransaction>
```

The `name="dimsearch_config"` attribute references the `dimsearch_config` document. Note that the `OuterTransactionId` element is not needed if the operation is not running within an outer transaction.

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 "ProductRelRank" in DIMSEARCH_CONFIG element.
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT3 error details:
org.apache.axis2.AxisFault: Error applying updates: Invalid Relevance ranking strategy
"ProductRelRank" in DIMSEARCH_CONFIG element.
```

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

Sample recsearch_config document

The following example shows a `recsearch_config` document with three search interfaces:

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

The example creates the Surveys, Resellers, and Employees search interfaces. All the configured standard attributes (such as DimEmployee_FullName) must already exist in the Dgraph.

Note that if you include a relevance ranking strategy, it must have been configured previously with the `relrank_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:configTransaction
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
<config-service:OuterTransactionId=${OUTER_TRANSACTION_ID}</config-service:OuterTransactionId>
<config-service:putConfigDocuments
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="recsearch_config">
<RECSEARCH_CONFIG>
  $xmlString
</RECSEARCH_CONFIG>
</mdex:configDocument>
</config-service:putConfigDocuments>
</config-service:configTransaction>
```

The `$xmlString` variable contains the XML definition of the search interfaces and the `name="recsearch_config"` attribute references the `recsearch_config` document. Note that the `OuterTransactionId` element is not needed if the operation is not running within an outer transaction.

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 "ProductRelRank" referenced
in SEARCH_INTERFACE "ProductSearch"
ERROR [WatchDog] - Node WEB_SERVICE_CLIENT0 error details:
org.apache.axis2.AxisFault: Error applying updates: Invalid
Relevance Ranking Strategy "ProductRelRank" referenced
in SEARCH_INTERFACE "ProductSearch"
```

To correct this error, first use the `relrank_strategies` document to create the relevance ranking strategy (named ProductRelRank in this example) in the Dgraph before you attempt to load your `recsearch_config` document.

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

```
Error applying updates: No property with the name "ProductType" exists for search interface
"ProductSearch"
```

To correct this error, first load your standard attribute schema before loading the configuration documents.

relrank_strategies document

This document configures the relevance ranking strategies for an Information Discovery 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 ProductRelRank that consists of the RELRANK_INTERP and RELRANK_FIELD relevance ranking modules.

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

The `name="relrank_strategies"` attribute references the `relrank_strategies` document. Note that the `OuterTransactionId` element is not needed if the operation is not running within an outer transaction.

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 Dgraph.

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

```
<STOP_WORDS/>
```

Sample stop_words document

This example sets the stop words for an application.

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

Request structure text

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

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

The `name="stop_words"` attribute references the `stop_words` document. Note that the `OuterTransactionId` element is not needed if the operation is not running within an outer transaction.

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 thesaurus entries:

```
<THESAURUS/>
```

Sample thesaurus document

This example sets two thesaurus entries:

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

Request structure text

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

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

The `name="thesaurus"` attribute references the `thesaurus` document. Note that the `OuterTransactionId` element is not needed if the operation is not running within an outer transaction.

Loading the configuration documents

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

The procedure is basically the same for all 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, as in this example:

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

The example shows that the `recsearch_config` document is being loaded. The `$xmlString` variable holds the actual definition of the search interface.

The individual topics for the configuration documents in this chapter describe details of these request structures.

Graph components

The Quick Start Sample Application uses a graph named `LoadIndexingConfiguration` to create the search configuration (including creating the search interfaces). The sample graph in this chapter uses the same components that create the search interfaces.

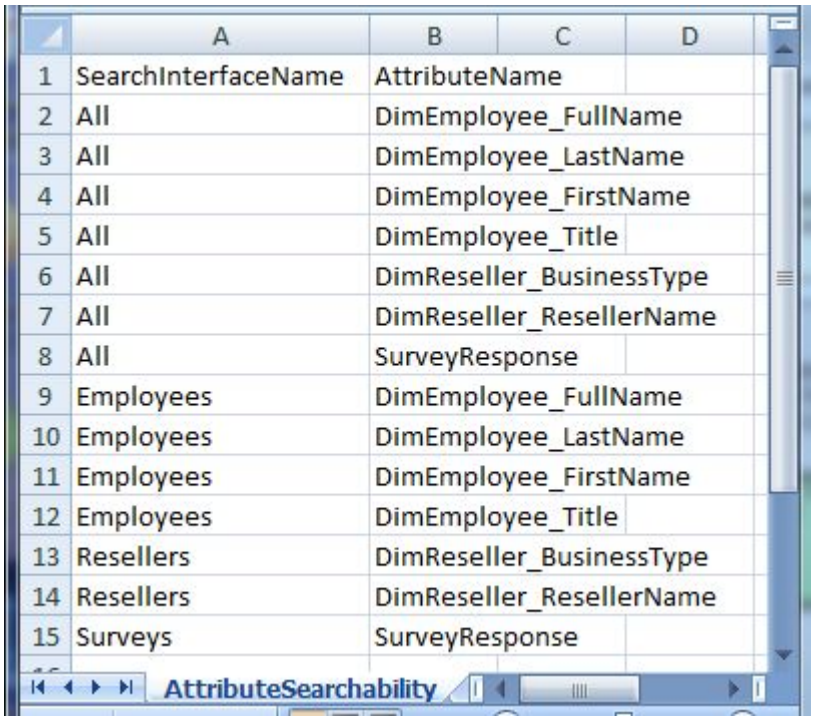
The sample graph in this chapter uses these components, in this order:

1. The **UniversalDataReader** component reads in the configuration document.
2. The **FastSort** transformer sorts the data before it is passed to the **Denormalizer** component (which requires sorted data).
3. The first **Denormalizer** component creates the search interface.
4. The second **Denormalizer** component creates a single output record for the whole group of input records.
5. The **WebServiceClient** component uses the Configuration Web Service's `config-service:putConfigDocuments` operation to load the configuration document into the Endeca data store. The request structure is shown in the `recsearch_config` document example above.

As noted above, the request structure in the **WebServiceClient** component will vary with each configuration document type.

Source file

The sample graph uses the same input CSV file used by the Quick Start project. The file, named `AttributeSearchability.csv`, looks like this:



	A	B	C	D
1	SearchInterfaceName	AttributeName		
2	All	DimEmployee_FullName		
3	All	DimEmployee_LastName		
4	All	DimEmployee_FirstName		
5	All	DimEmployee_Title		
6	All	DimReseller_BusinessType		
7	All	DimReseller_ResellerName		
8	All	SurveyResponse		
9	Employees	DimEmployee_FullName		
10	Employees	DimEmployee_LastName		
11	Employees	DimEmployee_FirstName		
12	Employees	DimEmployee_Title		
13	Resellers	DimReseller_BusinessType		
14	Resellers	DimReseller_ResellerName		
15	Surveys	SurveyResponse		

AttributeSearchability

The file provides inputs for three search interfaces:

- The Surveys search interface consists of only the SurveyResponse attribute.
- The Employees search interface consists of the DimEmployee_FullName, DimEmployee_LastName, DimEmployee_FirstName, and DimEmployee_Title attributes.
- The Resellers search interface consists of the DimReseller_BusinessType and DimReseller_ResellerName attributes.
- The All search interface consists of all the attributes.

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 an Integrator 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 **LoadConfigDocs** or **LoadSearchInterfaces**.
   (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

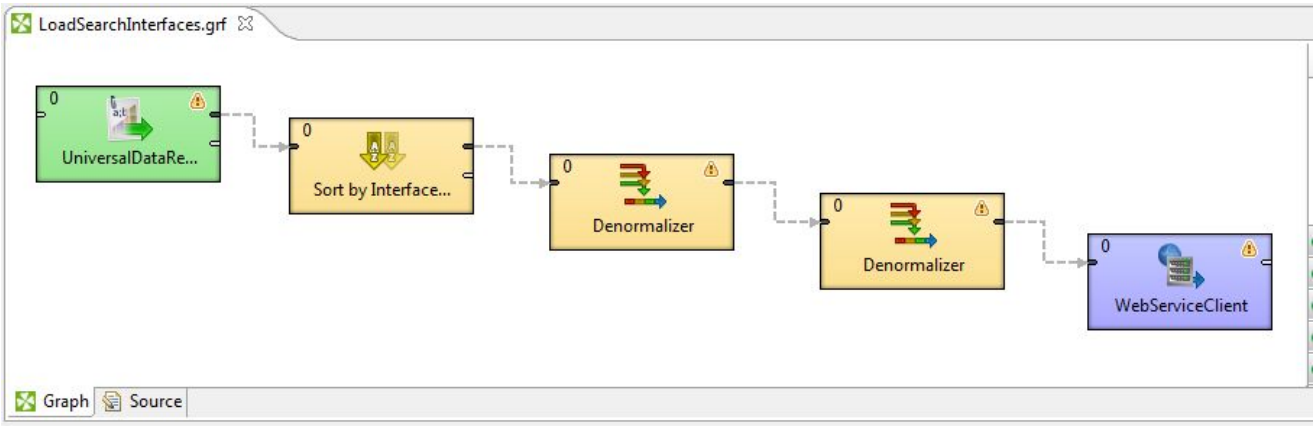
This tasks describes how to add the required components to the graph.

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

To add components to the graph:

1. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag the **UniversalDataReader** component from the **Readers** section.
   (b) Drag the **FastSort** component from the **Transformers** section.
   (c) Drag the **Denormalizer** component from the **Transformers** section.
   (d) Drag a second **Denormalizer** component from the **Transformers** section.
   (e) Drag the **WebServiceClient** component from the **Others** section.
2. In the Palette pane, click **Edge** and use it to connect the components.
3. From the File menu, click **Save** to save the graph.

At this point, the Graph Editor with the 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.

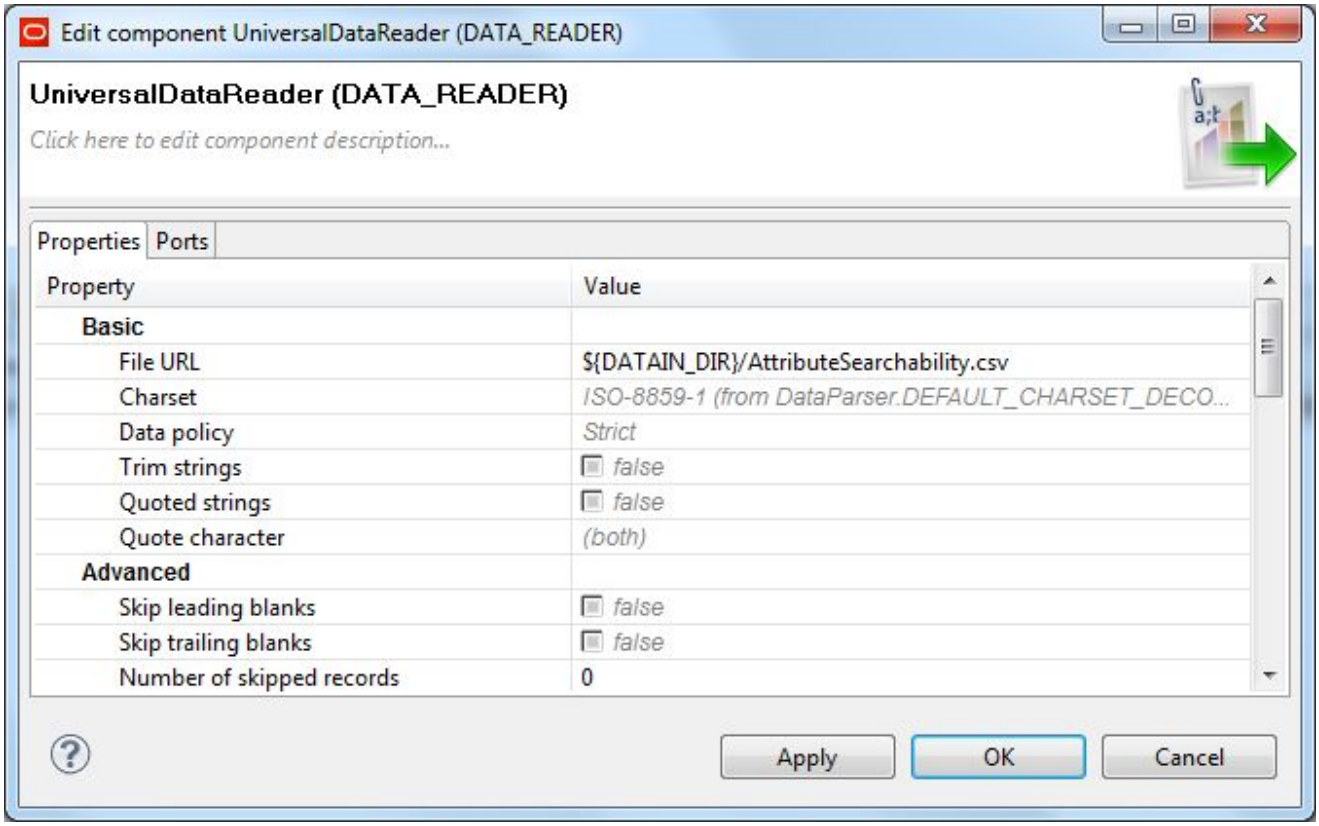
Important: The procedure also assumes that you have loaded your attribute schema (PDRs and DDRs) into the Endeca data store. This is because if the configuration document specifies an attribute to use, that attribute should already exist in the Dgraph; if it does not exist, the Dgraph may reject the configuration document and Integrator 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 input file folder (either **config-in** or **data-in**).
   (d) Select the source data file and click **OK**.
3. Leave the **Quoted strings** box to its default value of `false`.
4. Optionally, you can use the **Component name** field to provide a customized name (such as "Read Searchable Attributes") for this component.
5. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
6. Save the graph.

After step 5, the Reader Edit Component dialog should look like this example:



The next task is to configure the Reader's Edge.

Configuring metadata for the Reader Edge

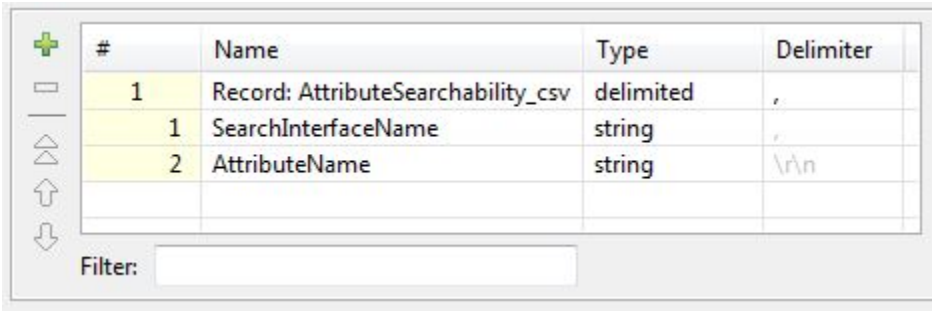
The Edge must be configured with a Metadata definition for loading a configuration document.

The prerequisite for this task is that an Edge component must connect the Reader and the following component.

Note: This procedure will configure metadata for loading the `recsearch_config` 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 input file.

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 input folder (such as **config-in**), 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.
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 displayed in the upper and middle panes of the Metadata Editor, which should look like this example:



6. In the Record pane, make these changes to the **Record** row:
   (a) Click the **Record** Name field and change its name to a more descriptive one, such as **SearchInterfaces**.
   (b) Leave the **Type** and **Delimiter** fields to their default settings.
7. Change the name of the **SearchInterfaceName** property to be **InterfaceName**.
8. When you have input all your changes in the Metadata Editor, click **Finish**.
9. Save the graph.

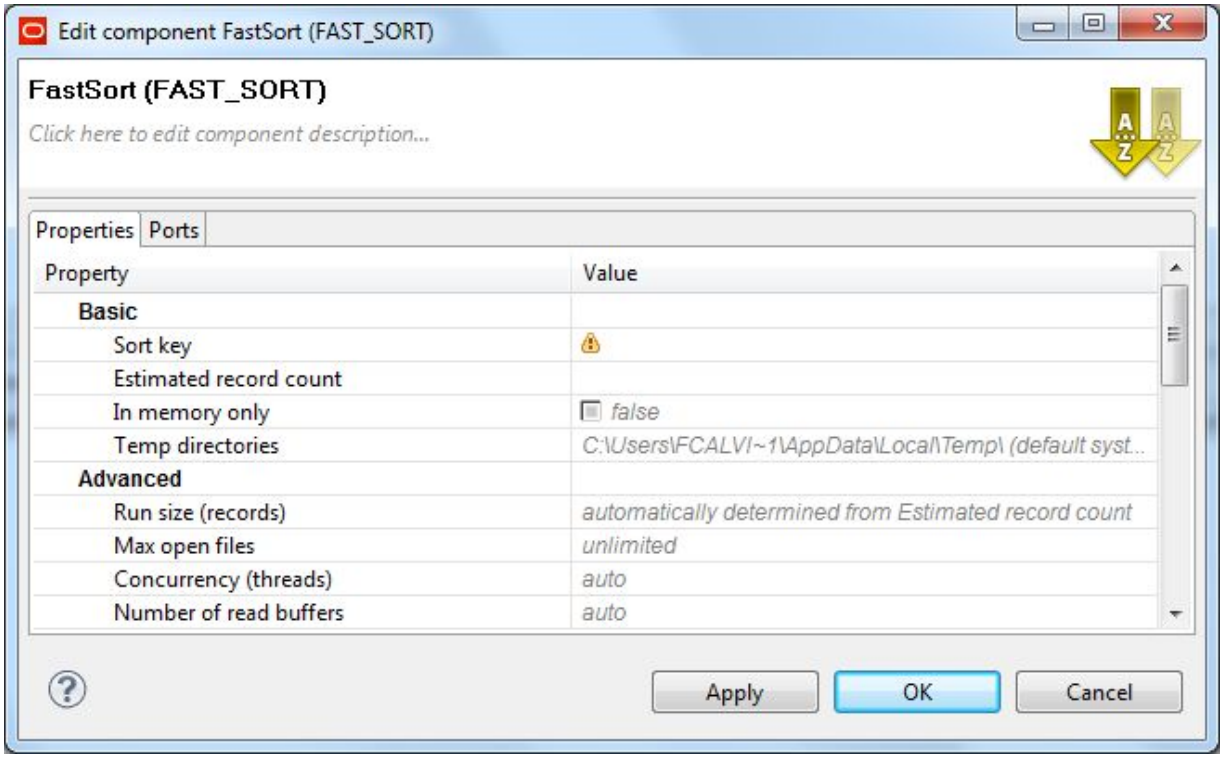
Configuring the FastSort component

The FastSort component takes input records and sorts them using a sorting key.

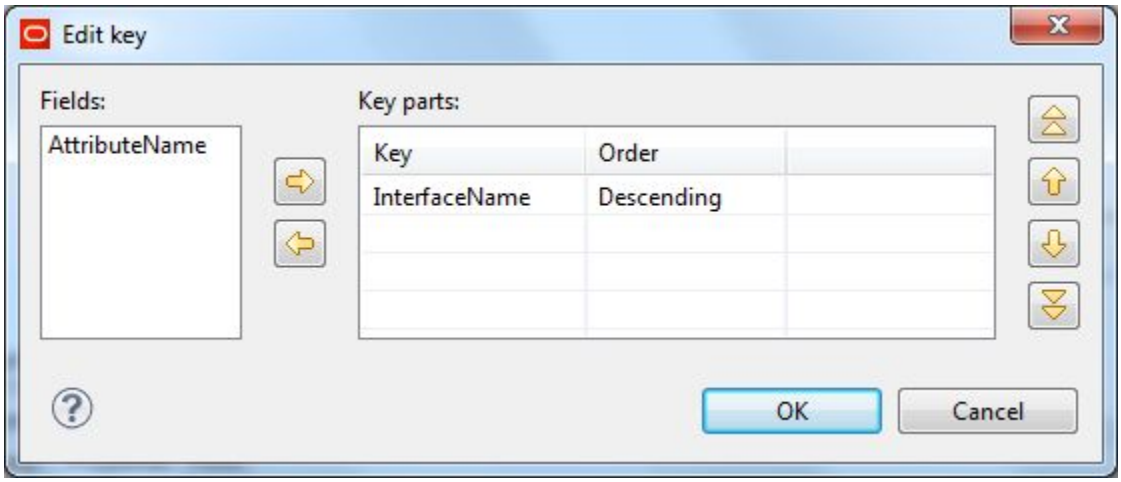
This component is necessary because the **Denormalizer** component (the next component in the graph) takes sorted data.

To configure the **FastSort** component:

1. In the Graph window, double-click the **FastSort** component.
   The FastSort Edit Component dialog is displayed.



2. Single-click in the **SortKey** field and then click the **...** button.
   The Edit Key editor is displayed.
3. In the Edit Key editor:
   (a) In the Fields pane, select the **InterfaceName** attribute.
   (b) Click the right-arrow button to move the SearchInterfaceName attribute to the Key Parts pane.
   (c) In the Key Parts pane, toggle the Order field to **Descending**. At this point, the Edit Key editor should look like this:



(d) Click **OK** to exit the Edit Key editor.

4. Optionally, you can use the **Component name** field to provide a customized name (such as "Sort by Interface Name") for this component.
5. In the FastSort Edit Component dialog, click **OK** to apply your changes and exit the component.
6. Save the graph.

Setting metadata for the FastSort component

The configuration of the Edge for the **FastSort** component is the same as for the **UniversalDataReader** component.

This procedure assumes that **SearchInterfaces** is the name of the metadata of the **UniversalDataReader** component.

To set the metadata for the **FastSort** Edge:

1. Right-click on the Edge and choose **Select metadata > SearchInterfaces**.
2. Save the graph.

Configuring the first Denormalizer component

The first **Denormalizer** component creates the search interface from the input data.

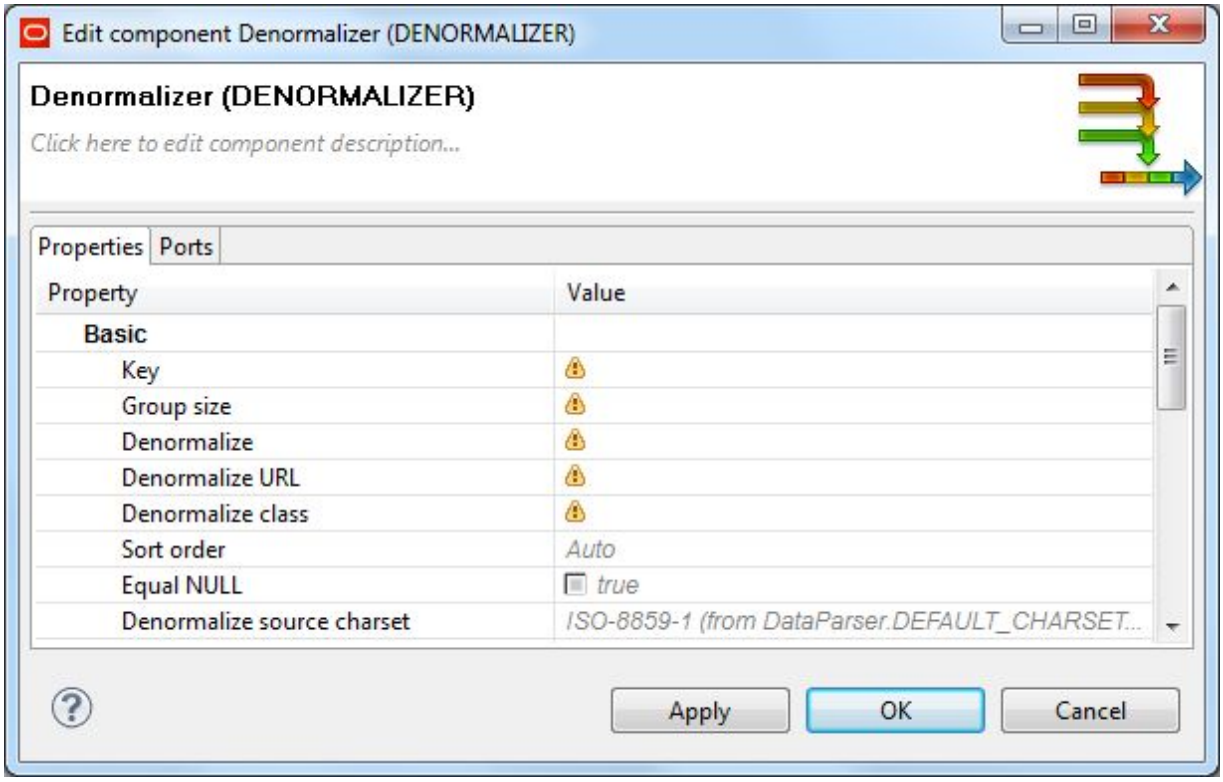
The transformation is done by these CTL functions in this **Denormalizer** component:

```
//#CTL2
// This transformation defines the way in which multiple input records
// (with the same key) are denormalized into one output record.
integer n = 0;
integer relRank = 1;
string value = "";
string nameOfInterface = "";

// This function is called for each input record from a group of records
// with the same key.
function integer append() {
  n++;
  value =
     value + "<MEMBER_NAME RELEVANCE_RANK='" + num2str(relRank) + "'>" +
```

```
    $0.AttributeName + "</MEMBER_NAME>";

  nameOfInterface = $0.InterfaceName;
  relRank++;

  return n;
}

// This function is called once after the append() function was called for all records
// of a group of input records defined by the key.
// It creates a single output record for the whole group.
function integer transform() {
  $0.xmlString = "<SEARCH_INTERFACE NAME=\"" + nameOfInterface + "\">"
       + value
       + "</SEARCH_INTERFACE>";
  $0.singleAggregationKey = 0;    // constant (aggregate everything into one request)

  value = "";
  nameOfInterface = "";
  relRank = 1;

  return OK;
}
```

To configure the first **Denormalizer** component in the graph:

1. In the Graph editor, double-click the first **Denormalizer** component.
   The Denormalizer Edit Component dialog is displayed.



2. Single-click in the **Key** field and then click the **...** button.
   The Edit Key dialog is displayed.
3. In the **Fields** pane of the Edit Key dialog, select **InterfaceName** and move it to the **Key parts** pane by clicking the right-arrow button. Click **OK** to apply your change.

4. Single-click in the **Denormalize** field and then click the **...** button.
   The Transform editor is displayed.
5. In the **Source** tab of the editor, modify the CTL script so that it looks like the example above.
   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.
6. When you have finished your edits, click **OK**.
   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.
7. Optionally, you can use the **Component name** field to provide a customized name for this component.
8. Click **OK** to apply your configuration changes.
9. Save the graph.

The two messages listed above should disappear once you configure the Edge for this **Denormalizer** component.

Configuring metadata for the first Denormalizer

This task describes how to configure the Edge component that connects the first and second **Denormalizer** components.

To configure the metadata for the first **Denormalizer** component:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.
2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a descriptive name. Our example will use **DenormEdge** as the name.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. For the other fields:
   (a) Change the **field1** name to `xmlString` and leave its **Type** as `string`.
   (b) Add a new field by using the + (plus sign control). Name the field **singleAggregationKey** and set its **Type** as `integer`.
4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Later, we will use this **DenormEdge** metadata for the second **Denormalizer** component.

Configuring the second Denormalizer component

The second **Denormalizer** component builds a single request.

The transformation is done by these CTL functions in this **Denormalizer** component:

```
//#CTL2
```

```
// This transformation defines the way in which multiple input records
// (with the same key) are denormalized into one output record.

// This function is called for each input record from a group of records
// with the same key.
integer n = 0;
string value = "";

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

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

To configure the second **Denormalizer** component in the graph:

1. In the Graph editor, double-click the second **Denormalizer** component.

   The Denormalizer Edit Component dialog is displayed.
2. Single-click in the **Key** field and then click the **...** button.

   The Edit Key dialog is displayed.
3. In the **Fields** pane of the Edit Key dialog, select **singleAggregationKey** and move it to the **Key parts** pane by clicking the right-arrow button. Click **OK** to apply your change.
4. Single-click in the **Denormalize** field and then click the **...** button.

   The Transform editor is displayed.
5. In the **Source** tab of the editor, modify the CTL script so that it looks like the example above.

   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.
6. When you have finished your edits, click **OK**.

   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.
7. Optionally, you can use the **Component name** field to provide a customized name for this component.
8. Click **OK** to apply your configuration changes.
9. Save the graph.

The two messages listed above should disappear once you configure this **Denormalizer** component's Edge.

The configuration of the Edge for this second **Denormalizer** component is the same as for the first **Denormalizer** component. In fact, you can use the same metadata that you created for the first one.

Setting metadata for the second Denormalizer

The configuration of the Edge for the second **Denormalizer** component is the same as for the first one.

This procedure assumes that **DenormEdge** is the name of the metadata of the first **Denormalizer** component. We will re-use that metadata for this second **Denormalizer** component.

To set the metadata for the Edge that connects the second **Denormalizer** component and the **WebServiceClient** component:

1. Right-click on the Edge and choose **Select metadata>DenormEdge**.
2. Save the graph.

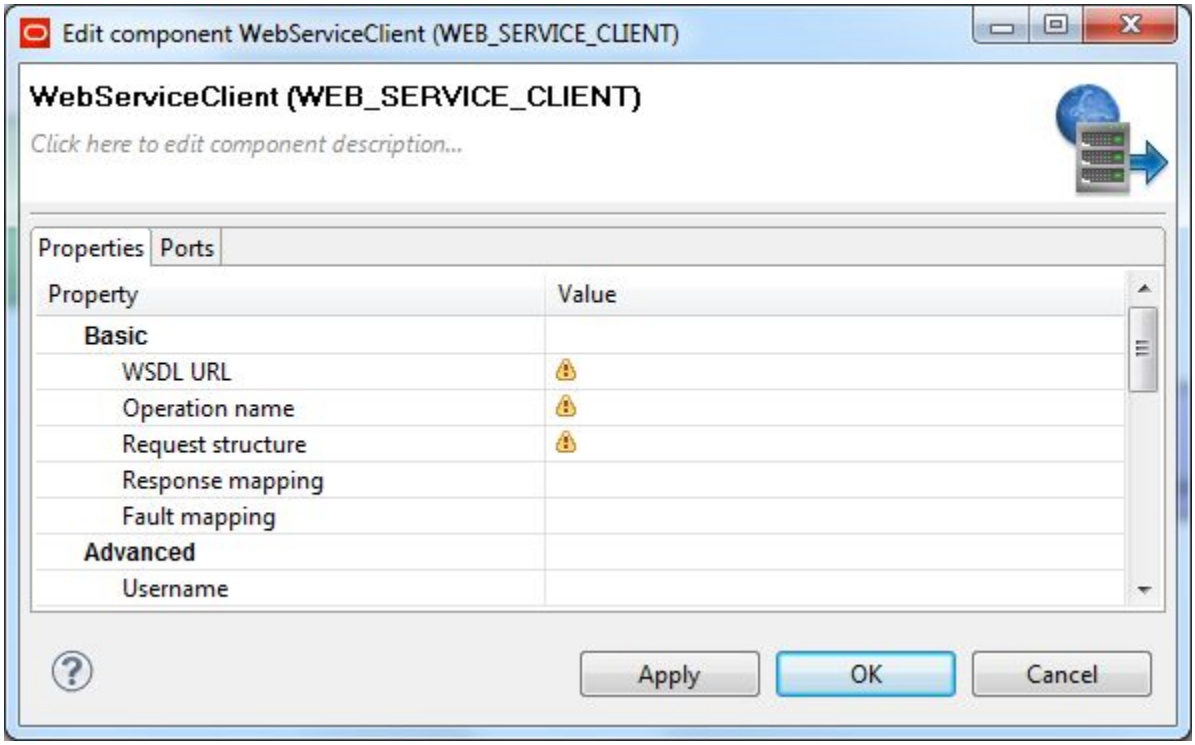
Configuring the WebServiceClient component

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

This procedure will configure metadata for loading the `recsearch_config` 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="recsearch_config">
```

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



To configure the **WebServiceClient** component:

1. Make sure that the Endeca data store instance is running and its Configuration Web service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

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

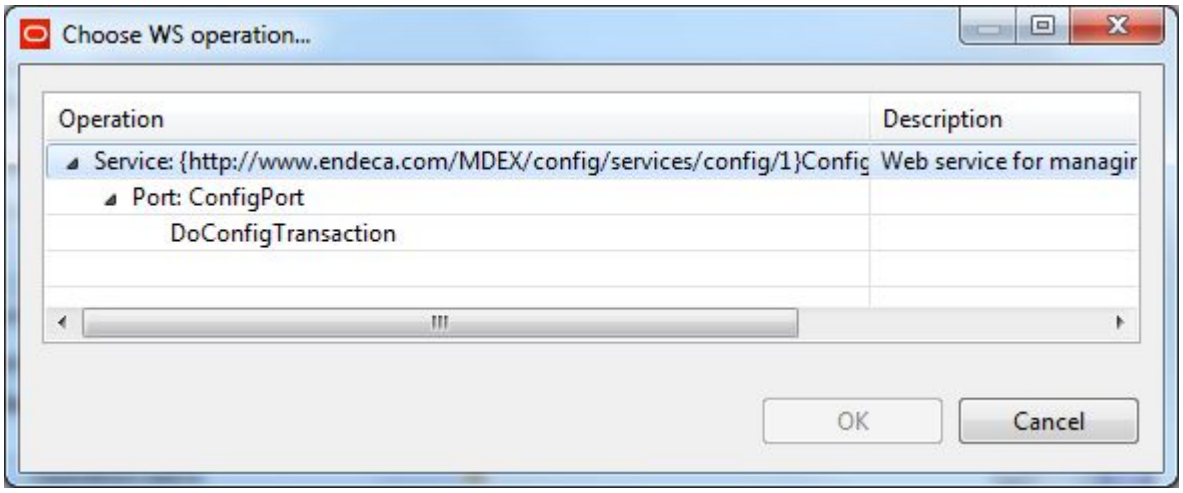
   The URL command returns the WSDL of the Configuration Web service.

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.

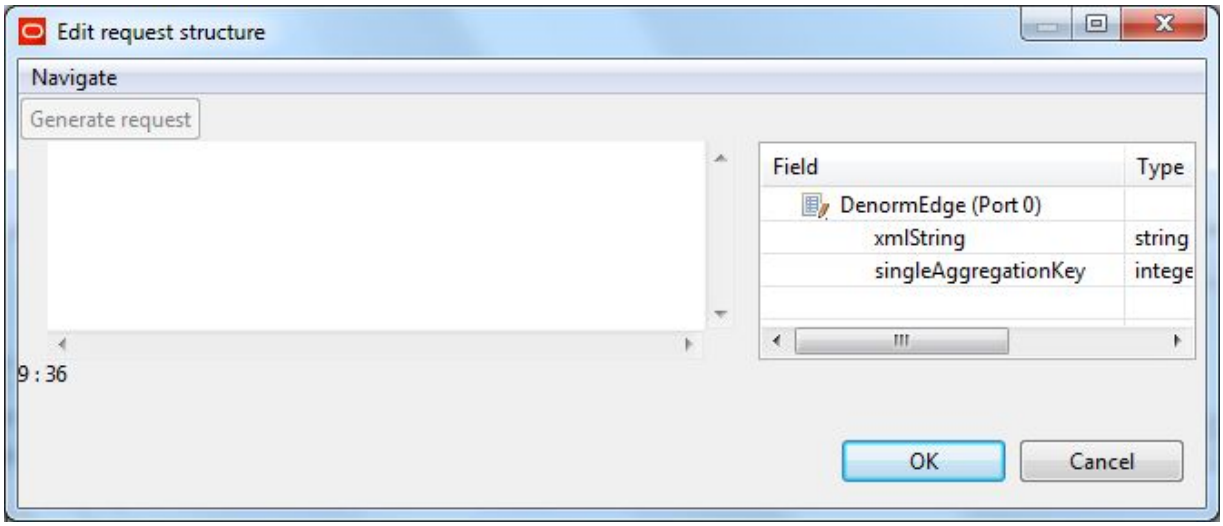
4. In the **Operation name** field, click the **...** browse button, which displays the **Choose WS operation** dialog:

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.

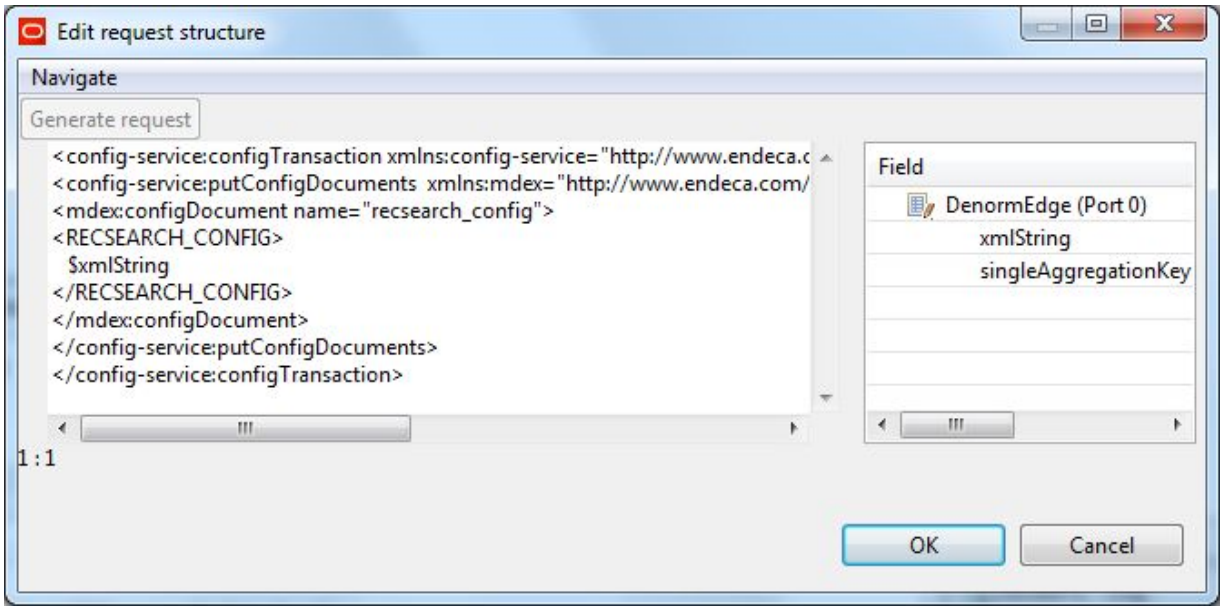
6. Click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the browse button to display the **Edit request structure** dialog:



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

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

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



8. After adding the request text in the **Edit request structure** dialog, click **OK**.

9. Optionally, you can use the **Component name** field to provide a customized name (such as "Load Configs") for this component.
10. When you have entered all your changes in the Edit Component dialog, click **OK**.
11. Save the graph.

Instead of running this graph directly, you may want to create a transaction graph (with a **Transaction RunGraph** connector) with this LoadSearchInterfaces graph as its child graph, and then run the transaction graph. For details on transaction graphs, see *Working with Outer Transaction Graphs on page 22*.

Loading the GCR

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

Loading the Global Configuration Record (GCR) into the Dgraph 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 5 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. You can build the GCR output xmlString with **Reformat** and **Denormalizer** components, similar to the graph that loads the standard attribute schema (described in Chapter 7, "Loading the Attribute Schema").
4. Configure the Reader and Transformation components and their metadata.
5. Configure the **WebServiceClient** component, as described in the topic *Configuring the WebServiceClient component on page 92*. The only difference is that you add this text to the **Generate request** field:

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

6. Make sure you save the graph.

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

Chapter 9

Loading Stemming Files

This chapter describes how to replace the stemming file in an Endeca data store with another language version.

About stemming files

Configuring the Reader in the stemming graph

Configuring the Edge in the stemming graph

Configuring the stemming WebServiceClient component

About stemming files

An Endeca data store's stemming feature allows the system to consider alternate forms of individual words as equivalent for the purpose of search query matching.

The stemming feature is described in detail in the *Oracle Endeca Server Developer's Guide*. As explained in that guide, the default language for the stemming feature is English.

You can, however, overwrite the default English stemming file with one from these other languages:

- Dutch (`nl_word_forms_collection.xml`)
- French (`fr_word_forms_collection.xml`)
- German (`de_word_forms_collection.xml`)
- Italian (`it_word_forms_collection.xml`)
- Portuguese (`pt_word_forms_collection.xml`)
- Spanish (`es_word_forms_collection.xml`)

These files, along with the English `es_word_forms_collection.xml` file, are shipped in the `lang/stemming` directory in the Integrator root installation.

After the new stemming file is loaded, the Dgraph rebuilds the stemming dictionary with the new word forms.

If you intend to use a stemming file from another language, keep the following limitations in mind:

- The Dgraph supports only one stemming language at a time. This means that while you can overwrite the current language (i.e., the stemming word forms in that language), you cannot load multiple languages and expect them to run concurrently.
- When loading a non-English stemming (word forms collection) file via the Configuration Web Service's `putConfigDocuments` operation, you must use `en_word_forms_collection` as the name in the `<mdex:configDocument>` element of the request, regardless of the file's language:

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

Using another name (such as `de_word_forms_collection`) will cause the request to fail. The reason is that the Configuration Web Service's WSDL is hard-coded to use the `en_word_forms_collection` name.

Configuring the Reader in the stemming graph

This task describes how to configure the **UniversalDataReader** component to read the stemming file.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. The procedure also assumes that you added a stemming file (`fr_word_forms_collection.xml` in our example) to the project's **data-in** folder. This reader will use that file as its input file.

To configure the Reader component in the stemming graph:

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, which brings up the **URL Dialog** screen.
3. In the **URL Dialog**:
   (a) Click the **Workspace view** tab and then double-click the **data-in** folder.
   (b) Select the `fr_word_forms_collection.xml` file and click **OK**.
4. In the Reader Edit Component dialog, click **OK**.
5. Save the graph.

Configuring the Edge in the stemming graph

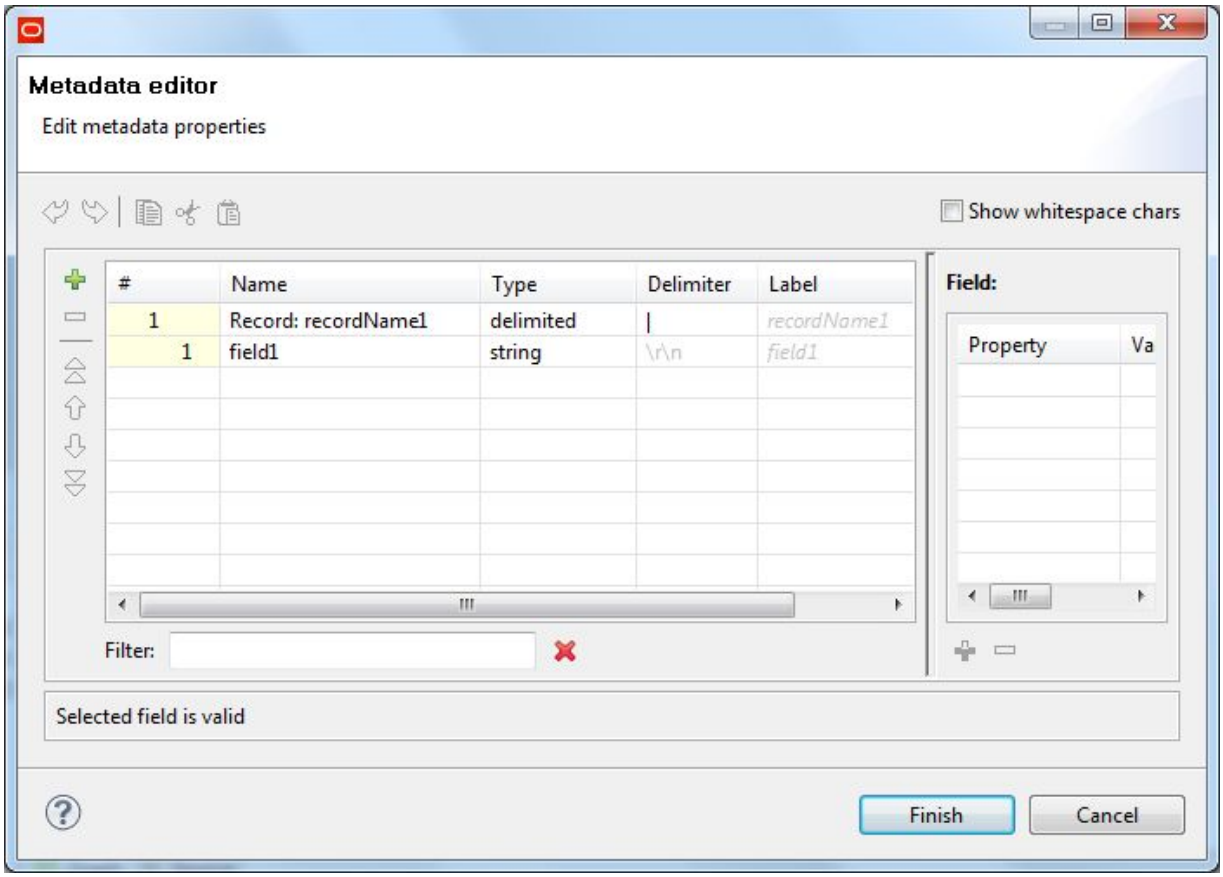
This topic describes how to configure the Edge metadata.

This procedure assumes that you have created a graph and added the **UniversalDataReader** and **WebServiceClient** components and connected them with an Edge.

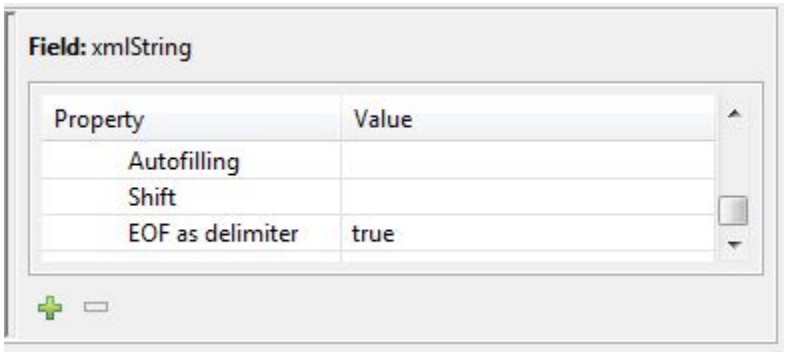
The contents of the stemming file will be read in as one long XML string. Therefore, the metadata must be configured to have only one string field and no record or field delimiters. This means that you must manually modify the Edge's metadata to remove the default record and field delimiters from the metadata. This will leave the **EOF as delimiter** property as the sole delimiter.

To configure the Edge Metadata definition for a stemming file:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **field1** default name to **xmlString**.
   (b) Leave the **Type** field set to **String**.
   (c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as in this example:



3. When you have input all your changes in the Metadata Editor, click **Finish**.
4. Now you must manually remove the record and field delimiters from the metadata:
   (a) In the Graph Editor, 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, as shown in this example:

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

(c) Delete the **fieldDelimiter** and **recordDelimiter** attributes, so that the Record element now looks like this:

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

(d) While still within the Source view, right-click and select **Save** to save the graph.

5. Click the **Graph** icon to return to the Graph Editor.

Configuring the stemming WebServiceClient component

This topic describes how to configure the **WebServiceClient** component to communicate with the Configuration Web service.

Keep in mind that regardless of which language stemming file you are loading, you must use `en_word_forms_collection` as the document name specified in the `<mdex:configDocument>` element of the request:

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

Using any other name (such as `fr_word_forms_collection`) will cause the request to fail.

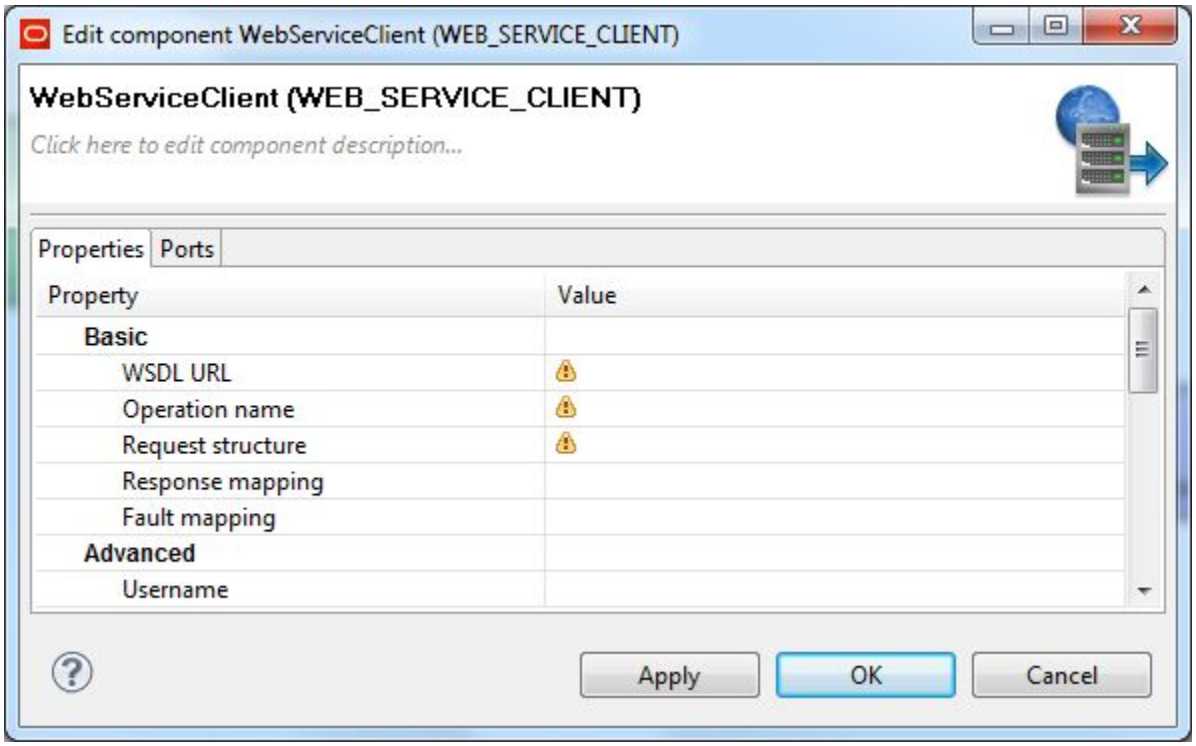
To configure the **WebServiceClient** component in the stemming graph:

1. Make sure that the Endeca data store instance is running and its Configuration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

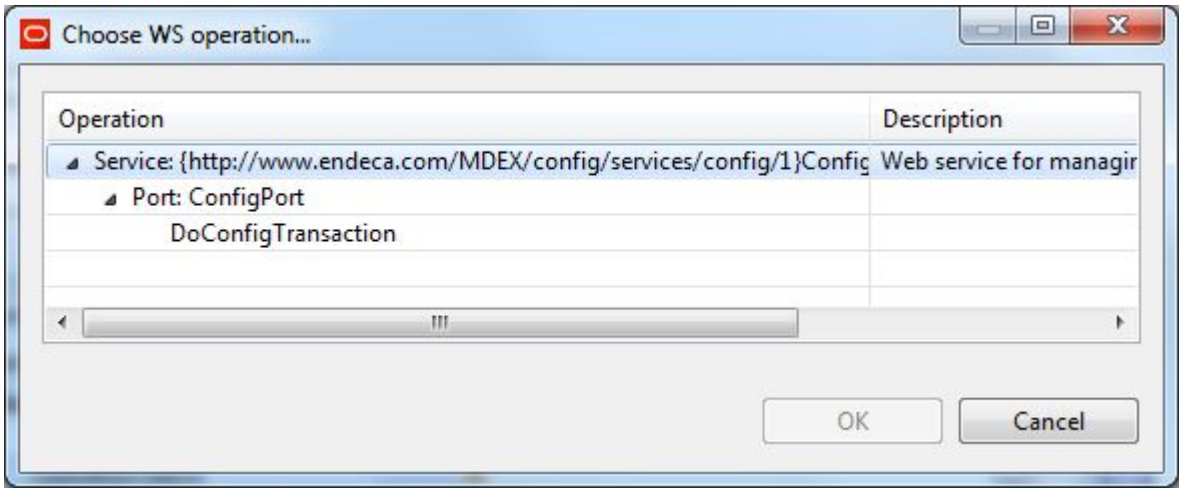
   ```
   http://localhost:7770/ws/config/bikes?wsdl
   ```

   The URL command returns the WSDL of the Configuration Web service.

2. In the Graph window, double-click the **WebServiceClient** component.
   The 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:

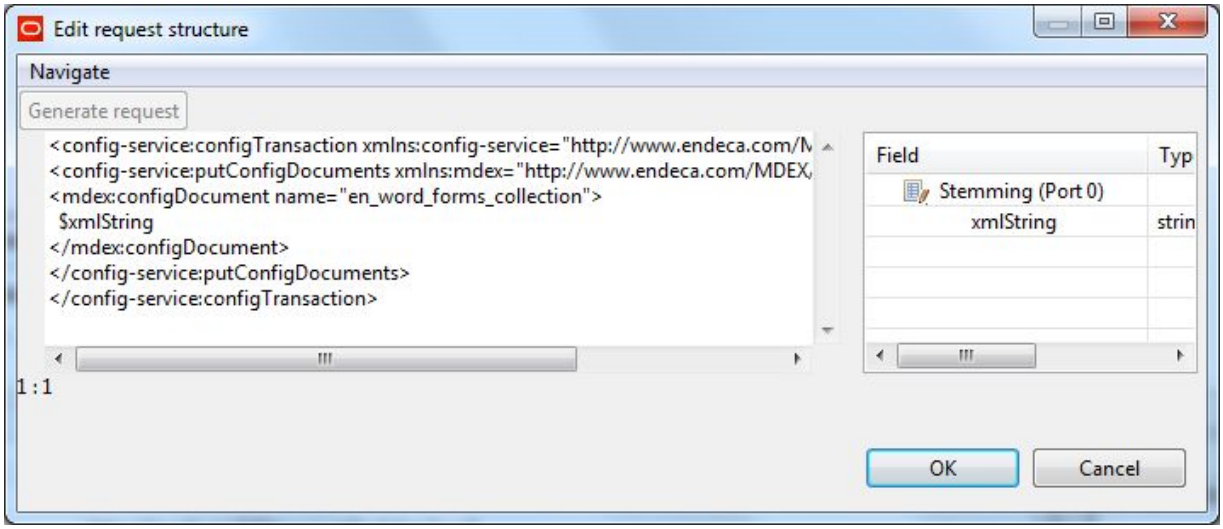


5. In the **Choose WS operation** dialog, select **DoConfigTransaction** and then click **OK**.
   The name of the Web service operation is entered in the **Operation name** field.
6. Click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the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

```
<config-service:configTransaction
```

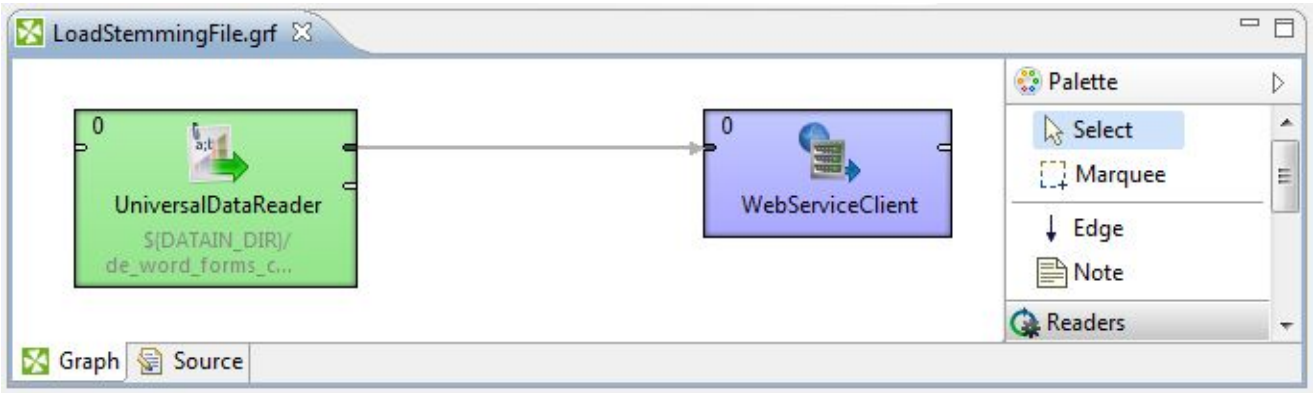
```
  xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0">
<config-service:putConfigDocuments
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
<mdex:configDocument name="en_word_forms_collection">
  $xmlString
</mdex:configDocument>
</config-service:putConfigDocuments>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

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



8. After adding the request text in the **Edit request structure** dialog, click **OK**.
9. Optionally, you can use the **Component name** field to provide a customized name (such as "Load Stemming File") for this component.
10. When you have entered all your changes in the Edit Component dialog, click **OK**.
11. Save the graph.

The final graph will look similar to this example:



When you run the graph, the new stemming file will overwrite the current one in the Endeca data store.

Chapter 10

Loading Precedence Rules

This chapter describes how to load your precedence rules into the Dgraph.

About precedence rules

Schema for precedence rules

Format of the precedence rules input file

Adding components to the precedence rules graph

Configuring the precedence rules Reader

Configuring the Reformat component for precedence rules

Configuring the precedence rules WebServiceClient component

Deleting precedence rules

About precedence rules

A precedence rule 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.

Precedence rules allow your application to delay the display of Endeca standard or managed attributes that the user triggers. 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.

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.

The standard and/or managed attributes referenced in precedence rules do not have to exist in the Dgraph 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.

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

Schema for precedence rules

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

The `config-service:putPrecedenceRules` operation creates each of the given precedence rules or updates them if they already exist. Each `precedenceRule` element uses this schema syntax:

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

The meanings of the `precedenceRule` attributes are as follows:

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

precedenceRule attribute	Meaning
isLeafTrigger	If the trigger is a managed attribute, isLeafTrigger specifies a Boolean value (that must be in lower case) that denotes the type of the trigger attribute value: • If true, the trigger attribute is a leaf type, which means that the precedence rule will fire only if a leaf value is selected. That is, querying any leaf managed value from the trigger managed attribute will cause the target managed value to be displayed (many triggers, one target). • If false (the default), the trigger attribute is a non-leaf type, which means that the precedence rule will fire when any value is selected. That is, if the managed value specified as the trigger or any of its descendants are in the navigation state, then the target is presented (one trigger, one target). Note that isLeafTrigger does not apply to Endeca standard attributes. You must specify it when you create a precedence rule, but whichever value you use is ignored by the Dgraph when the precedence rule is run.

Precedence rule example

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

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

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

Format of the precedence rules input file

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

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

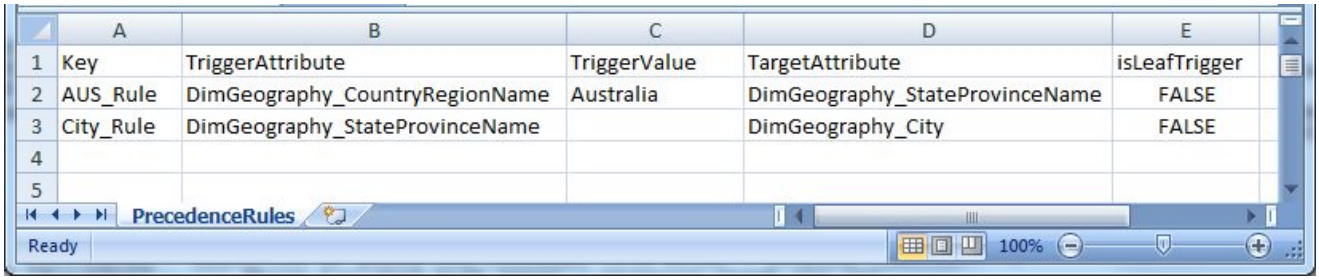
```
Key|TriggerAttribute|TriggerValue|TargetAttribute|isLeafTrigger
```

The actual names of the header properties in your input file can be different from the names used here (for example, you can use RuleName instead of Key). The properties are delimited (for example, by the comma in a CSV file or the pipe character in a text file).

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

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

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



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

Adding components to the precedence rules graph

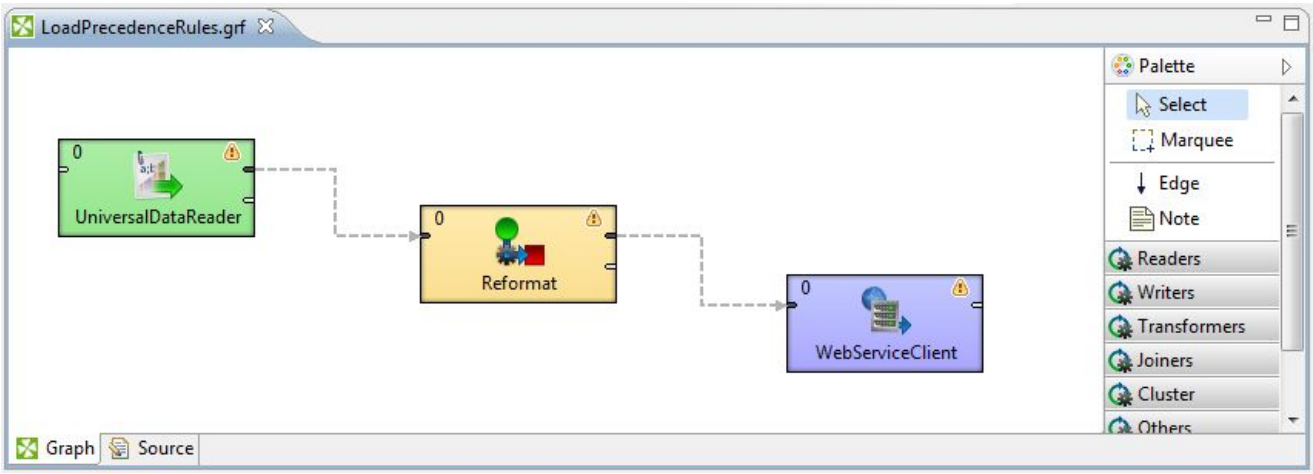
You must add the **UniversalDataReader**, **Reformat**, and **WebServiceClient** components to the graph.

This procedure assumes that you have created an empty graph.

To add components to the graph:

1. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag the **UniversalDataReader** component from the **Readers** section.
   (b) Drag the **Reformat** component from the **Transformers** section.
   (c) Drag the **WebServiceClient** component from the **Others** section.
2. In the Palette pane, click **Edge** and use it to connect the components.
3. From the File menu, click **Save** to save the graph.

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



The next tasks are to configure these components.

Configuring the precedence rules Reader

This task describes how to configure the **UniversalDataReader** component to read in the configuration file for creating precedence rules.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the precedence rule configuration file to the project's **data-in** (or **config-in**) folder.

To configure the **UniversalDataReader** component for the precedence rules configuration 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 input file folder (either **config-in** or **data-in**).
   (d) Select the configuration input file and click **OK**.
3. Optionally, you can use the **Component name** field to provide a customized name (such as "Read Rules Metadata") for this component.
4. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
5. Save the graph.

The next task is to configure the Reader's Edge.

Configuring the Reader Edge

This task describes how to configure the Reader Edge component for the Metadata definition.

To configure the Reader Edge component:

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:
   (a) Double-click the input folder.
   (b) Select the configuration source file and click **OK**.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed.
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 Record pane, you should change the **recordName1** default value to a name that is appropriate for your data.
7. When you have input all your changes, click **Finish**.
8. Save the graph.

Configuring the Reformat component for precedence rules

A **Reformat** component is used to transform incoming configuration data into a `precedenceRule` record.

The transformation is done by this CTL function in the **Reformat** component:

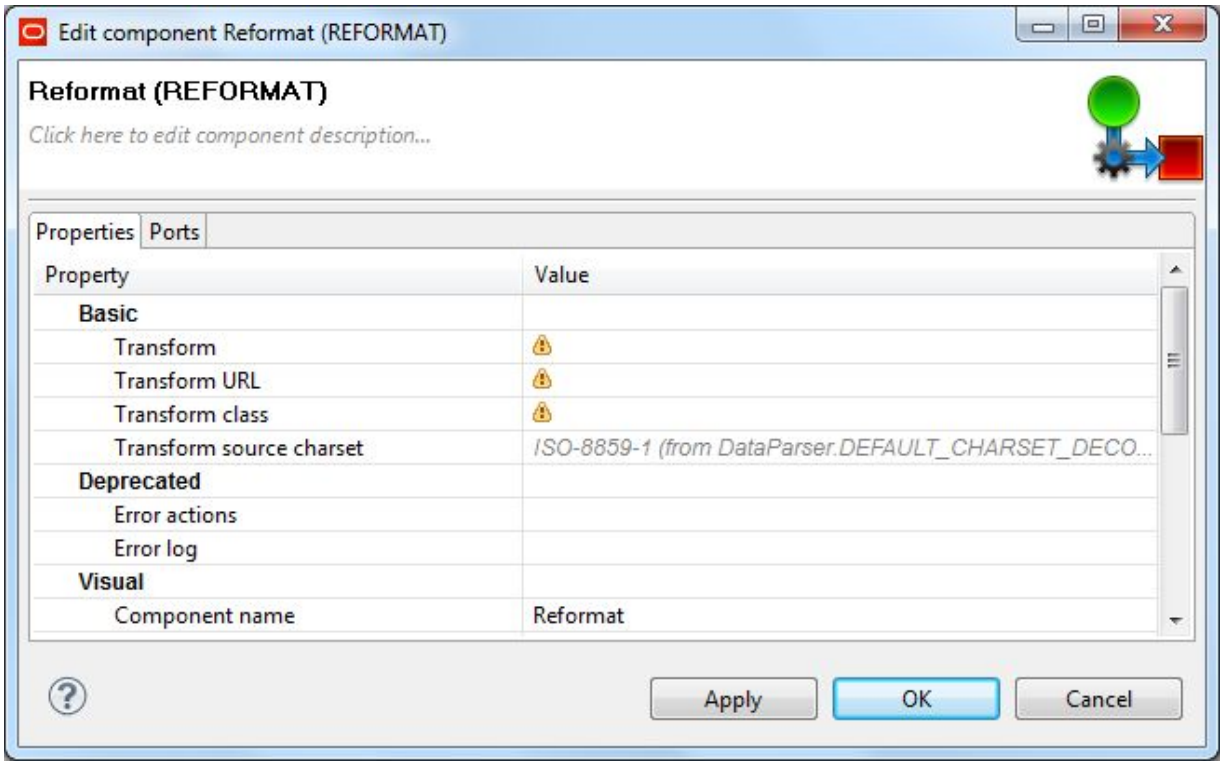
```
function integer transform() {
   string prRecord = "";
   string isLeaf = "";

   // Begin building the precedenceRule record
   prRecord = "<mdex:precedenceRule ";
   // Add the name of the rule.
   prRecord = prRecord + "key='" + $0.Key + "' ";
   // Add the name of the trigger attribute
   prRecord = prRecord + "triggerAttributeKey='" + $0.TriggerAttribute + "' ";

   // Add mval or pval trigger value only if present in the input file
   if ($0.TriggerValue != null && !$0.TriggerValue.isBlank()) {
      prRecord = prRecord + "triggerAttributeValue='" + $0.TriggerValue + "' ";
   }

   // Add the name of the target attribute
   prRecord = prRecord + "targetAttributeKey='" + $0.TargetAttribute + "' ";

   // Add the boolean that specifies if the trigger is a leaf
   // Lower case the boolean in the CSV file
   isLeaf = lowerCase($0.isLeafTrigger);
   prRecord = prRecord + "isLeafTrigger='" + isLeaf + "'/>";
```

```
   // Append the record to the xmlString variable, which stores all the rules
   $0.xmlString = prRecord;

   return ALL;
}
```

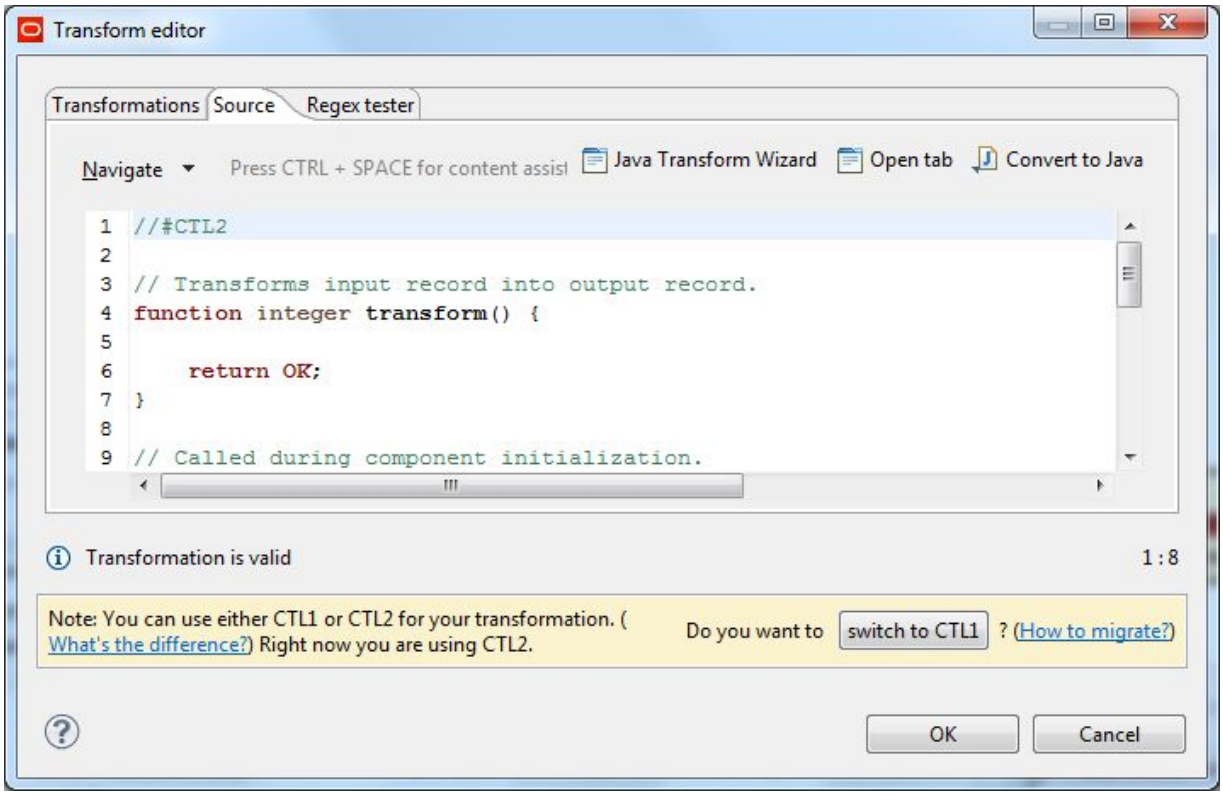
When it runs, the component will build one or more `precedenceRule` elements and send them in the `xmlString` property to the **WebServiceClient** component in the graph.

To configure the **Reformat** component in the precedence rules graph:

1. In the Graph window, double-click the **Reformat** component.
   The Reformat Edit Component dialog is displayed.



2. Single-click in the **Transform** field and then click the **...** button.
   The Transform editor is displayed.
3. Click the **Source** tab in the editor.
   The CTL template for the transform function is shown.



4. Modify the CTL script so that it looks like the example above.
   You may see the message "Cannot write to output port '0'" at the bottom of the editor. Assuming you have not made any coding errors, you may disregard the message for now.
5. When you have finished your edits, click **OK**.
   If you see the error "Transformation contains syntax errors! Accept it anyway?" in a pop-up message, click **Yes**.
6. Optionally, you can use the **Component name** field to provide a customized name (such as "Transform Precedence Rules") for this component.
7. Click **OK** to apply your configuration changes.
8. Save the graph.

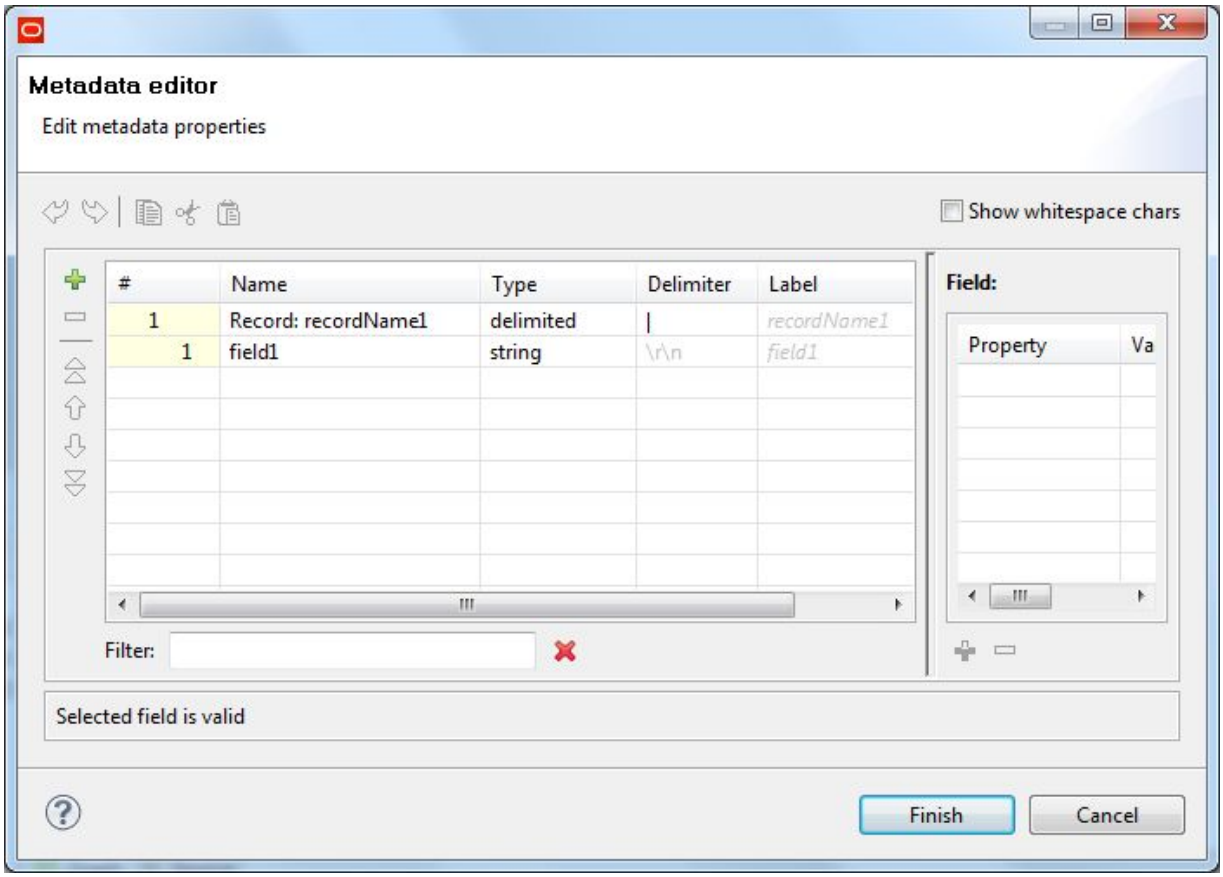
The two messages listed above should disappear once you configure the **Reformat** component Edge metadata.

Configuring the precedence rules Reformat Edge

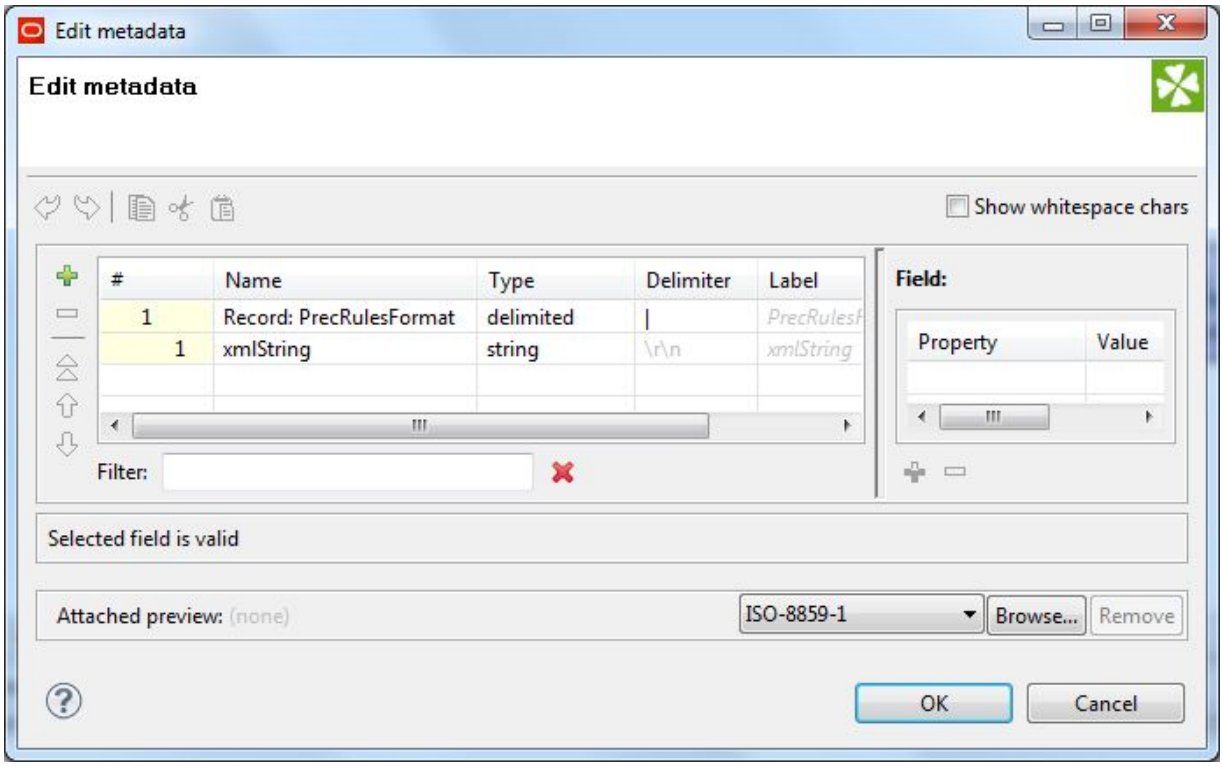
This task describes how to configure the Edge component that connects the **Reformat** and **WebServiceClient** components.

To configure the **Reformat** component's Edge in the precedence rules graph:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file.
3. Change the **field1** name to `xmlString` and leave its **Type** as `string`.
   You can leave the **Delimiter** field unchanged.
   At this point, the Metadata editor should look like this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the precedence rules WebServiceClient component

The **WebServiceClient** component must be configured to communicate with the Configuration Web Service.

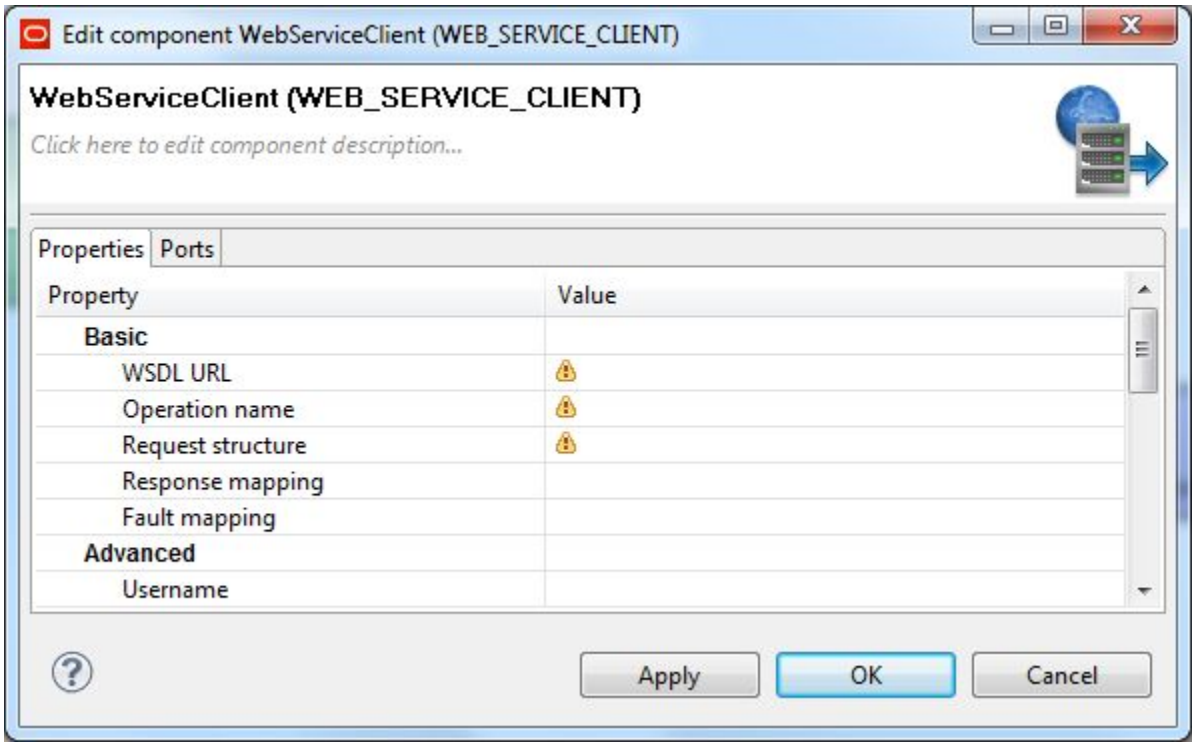
In addition, you must add a `config-service:putPrecedenceRules` operation to the request structure of the component.

To configure the **WebServiceClient** component in the precedence rules graph:

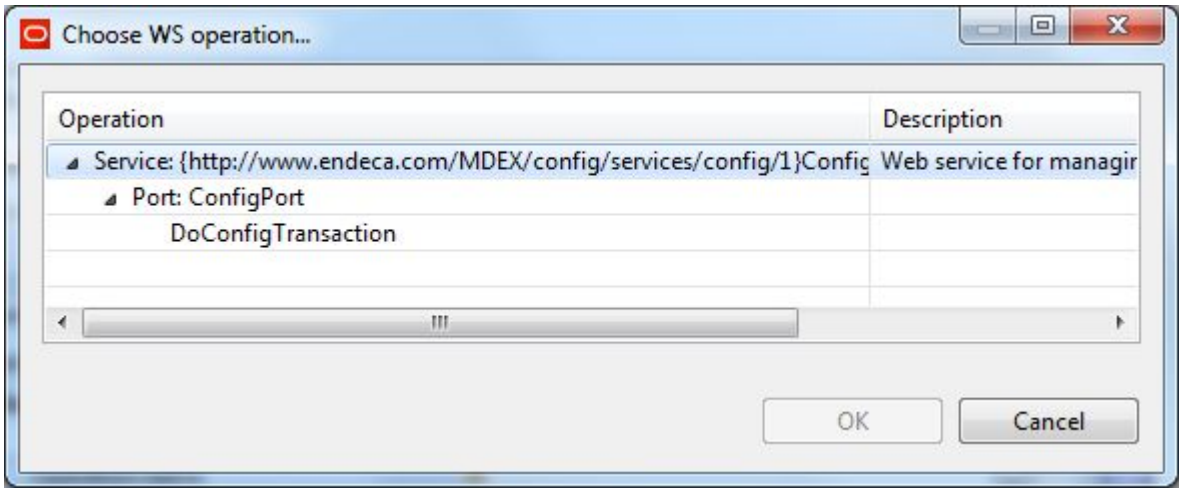
1. Make sure that the Endeca data store instance is running and its Configuration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

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

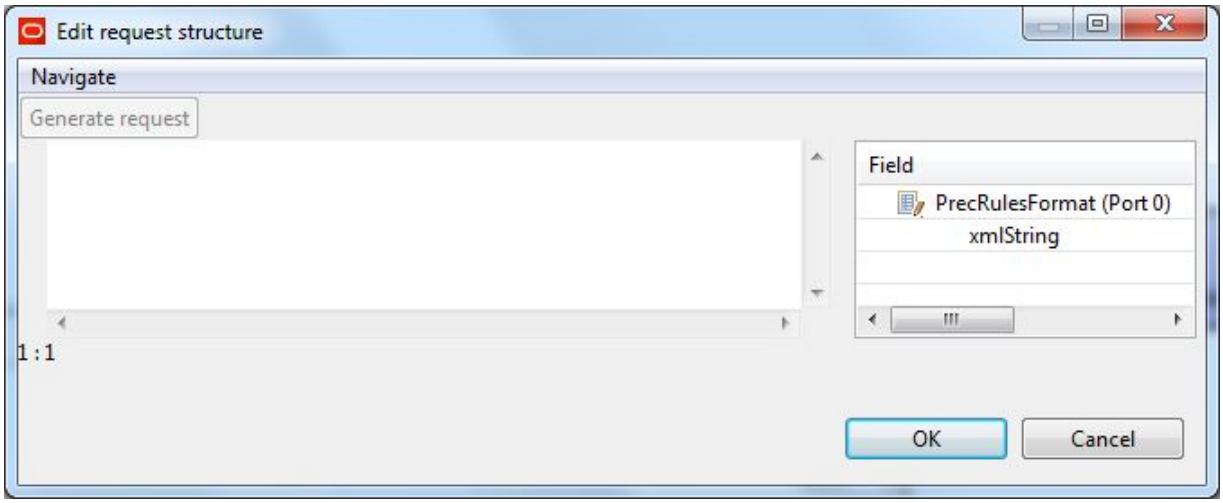
   The URL command returns the WSDL of the Configuration Web service.
2. In the Graph editor, double-click the **WebServiceClient** component.
   The Edit Component dialog is displayed.



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:



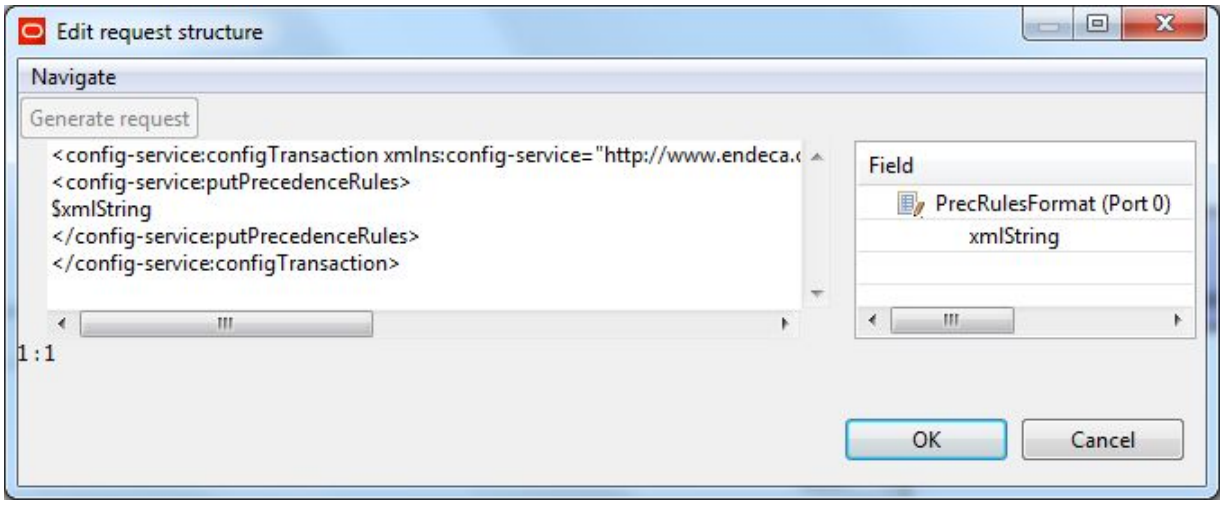
5. In the **Choose WS operation** dialog, select **DoConfigTransaction** and then click **OK**.
   The name of the Web service operation is entered in the **Operation name** field.
6. Click inside the **Request structure** field, which causes the **...** browse button to be displayed. Then click the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



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

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

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



8. After adding the request text in the **Edit request structure** dialog, click **OK**.
9. Optionally, you can use the **Component name** field to provide a customized name (such as "Load Precedence Rules") for this component.
10. When you have entered all your changes in the Edit Component dialog, click **OK**.
11. Save the graph.

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

Deleting precedence rules

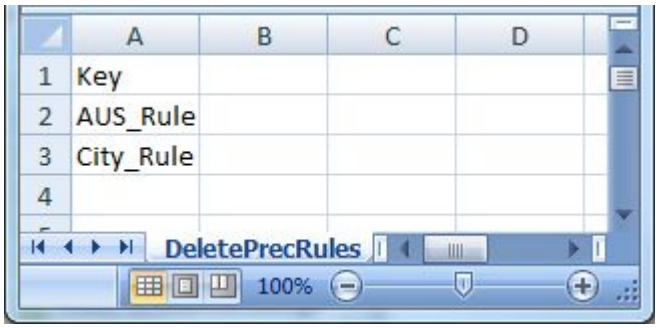
The `config-service:deletePrecedenceRules` operation lets you remove an existing precedence rule from the Dgraph.

The Configuration Web Service's `deletePrecedenceRules` operation takes one or more `precedenceRule` elements that will be deleted. Because precedence rules are stored as records in the Dgraph, you need to specify only the `key` attribute of the precedence rule, as in this example that deletes a precedence rule named "ProvinceRule":

```
<config-service:configTransaction
     xmlns:config-service="http://www.endeca.com/MDEX/config/services/types/1/0"
     xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <config-service:deletePrecedenceRules>
      <mdex:precedenceRule key="ProvinceRule"/>
   </config-service:deletePrecedenceRules>
</config-service:configTransaction>
```

To delete precedence rules from an Endeca data store:

1. Create an input file that contains one column, with a **Key** header name and with one or more rows of precedence rule names, as in this simple CSV example:



2. Create a graph and add the components described in the topic *Adding components to the precedence rules graph on page 105*.
3. Configure the **UniversalDataReader** component as described in the topic *Configuring the precedence rules Reader on page 106*.
   Make sure you use the file created in Step 1 as the input file and that the **Number of skipped records per source** field is set to 1.
4. To configure the Reader Edge, use the procedure in the topic *Configuring the Reader Edge on page 107*.
   Note that the **Record** field's Delimiter field will be empty, as there is only one column.
5. Configure the **Reformat** component so that the CTL in the **Source** tab looks like this:

```
function integer transform() {
   string prRecord = "";

   prRecord = "<mdex:precedenceRule key='" + $0.Key + "'/>";

   $0.xmlString = prRecord;

   return ALL;
}
```

6. To configure the Reformat Edge, use the procedure in the topic *Configuring the precedence rules Reformat Edge on page 109*.

Note that the **Record** field's Delimiter field will be empty, as there is only one column.

7. To configure the **WebServiceClient** component, use the procedure in the topic *Configuring the precedence rules WebServiceClient component on page 111*. The major difference is that you will add this text to the **Generate request** field:

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

8. Save the graph.

After creating the graph and configuring the components, run the graph to delete the precedence rules listed in the input file.

Chapter 11

Adding Key-Value Pairs

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

About key-value pair data

Format of the KVP input file

Configuring the Reader for the KVP input file

Configuring the Add KVPs connector

Configuring KVP metadata

Running the KVPs graph

About key-value pair data

The **Add KVPs** connector can add key-value pair data to records in an Endeca data store.

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.

Schema property	Meaning
`kvpKey`	The name (key) of the Endeca standard attribute to be added to the record. If the standard attribute does not exist in the Dgraph, it is automatically created by DIWS with system default values. For the default values, see *Standard attribute default values on page 9*.
`kvpValue`	The value of the standard attribute to be added to the record.
`mdexType`	Specifies the `mdex` type (such as `mdex:int` or `mdex:dateTime`) for the `kvpKey` standard attribute. This parameter is intended for use when you want to create a new standard attribute and want to specify its property type. If a new PDR for the standard attribute is created and `mdexType` is not specified, then the type of the new standard attribute will be `mdex:string`. If the standard attribute already exists, you can specify an empty value for `mdexType`. For a list of valid data types, see *Supported data types on page 8*.

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

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

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

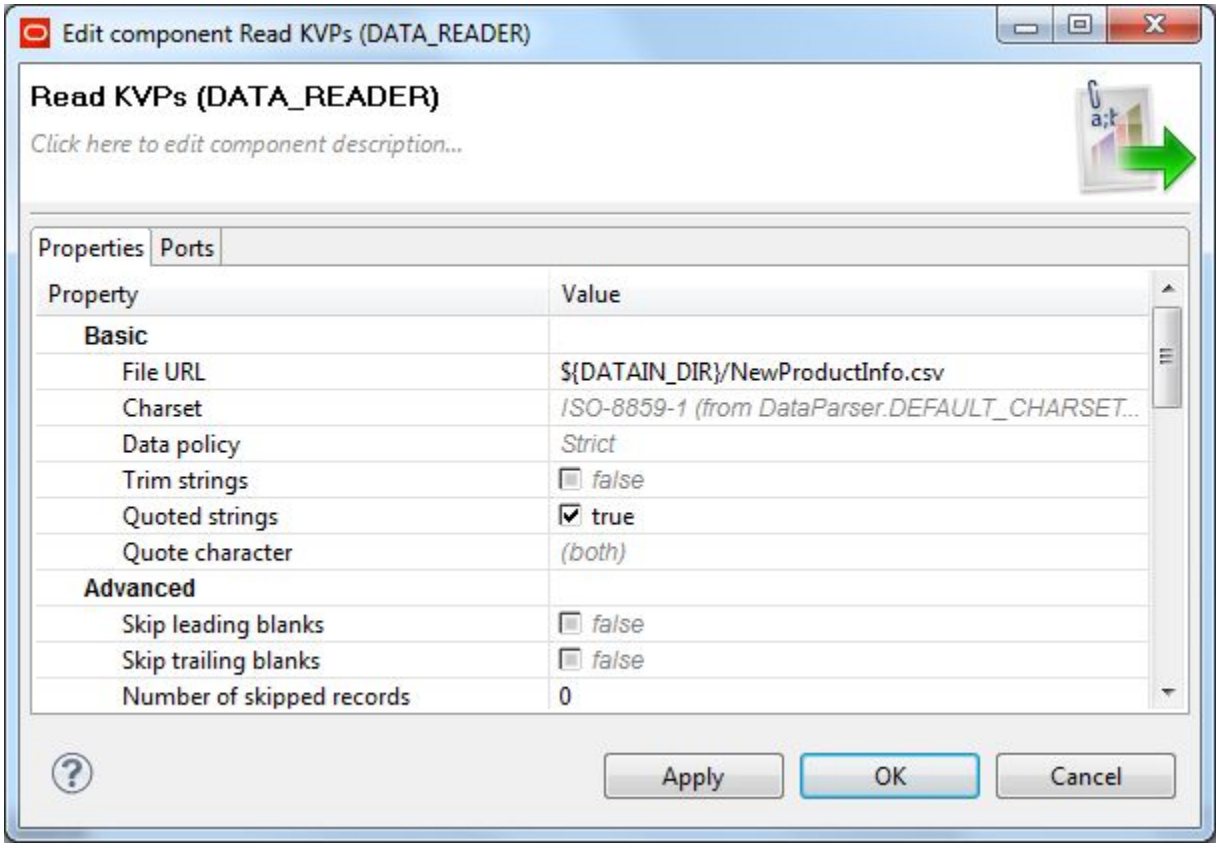
This procedure assumes that you have created a graph for the KVP components and that you have copied the input file (named `NewProductInfo.csv` in our example) into the **data-in** folder in the Navigation pane of the project. The procedure also assumes that you will be using the **UniversalDataReader** 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 0.
6. Optionally, you can use the **Component name** field to provide a customized name (such as "Read KVPs") for this component.
7. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
8. Save the graph.

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



Configuring the Add KVPs connector

You must configure the **Add KVPs** connector to properly connect to your Endeca data store.

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 **Discovery** 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) **Endeca Server Host**: The host name of the machine on which the Endeca Server is running. You can specify **${ENDECA_SERVER_HOST}** if you have that variable defined in the `workspace.prm` file for your project.

(b) **Endeca Server Port**: The port on which the Endeca Server is listening. You can specify **${ENDECA_SERVER_PORT}** if you have that variable defined in the `workspace.prm` file for your project.

(c) **Data Store Name**: The name of the Endeca data store that contains the records to be added or modified.

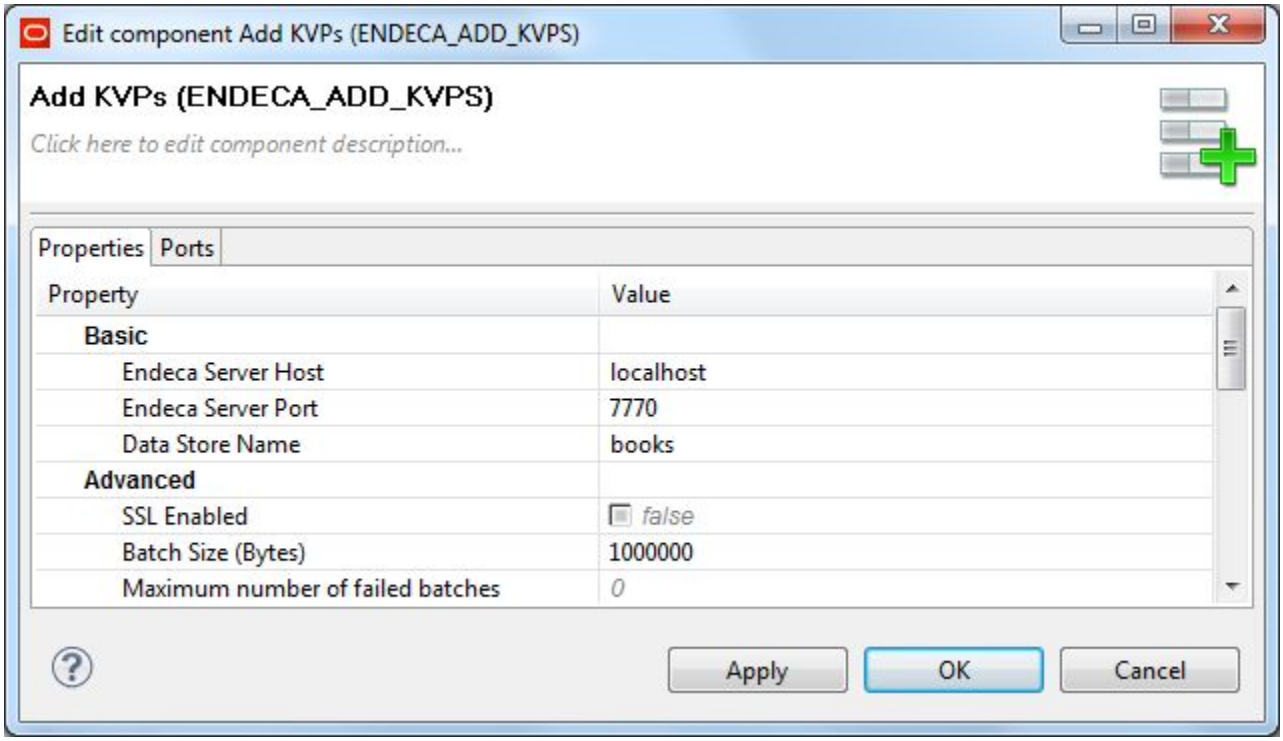
(d) **SSL Enabled**: Toggle this field to `true` if the Endeca Server is SSL-enabled.

(e) **Batch Size (Bytes)**: To change the default batch size, enter a positive integer. Specifying 0 or a negative number will disable batching.

(f) **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:



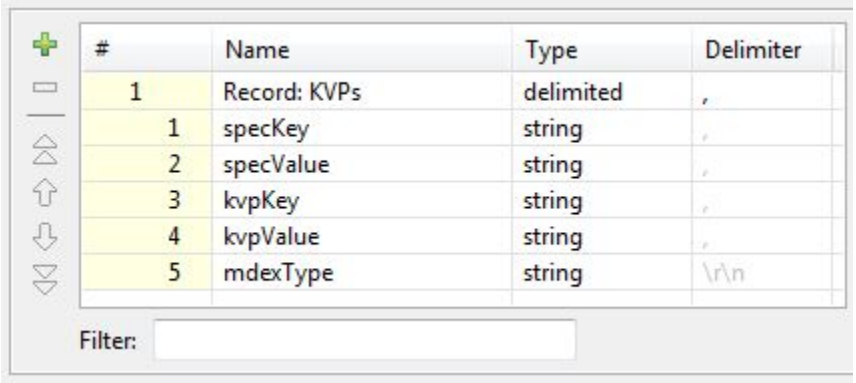
Configuring KVP metadata

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 key-value pair 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**.
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** Name field and change the 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 comma character in our example).
   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**.
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 Endeca data store.

To run the graph to add key-value pairs to an Endeca data store:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **Add KVPs** connector and that the Endeca data store has been started.
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 running Endeca data store:

```
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: 04/06/12 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 12

Loading Taxonomies

This chapter describes how to load an externally managed taxonomy (EMT) into an Endeca data store.

Overview of loading a taxonomy

Format of the taxonomy input file

Creating a graph for the taxonomy

Adding components to the taxonomy graph

Configuring the Reader for the taxonomy input file

Configuring the Add Managed Values connector

Configuring taxonomy metadata

Running the taxonomy graph

Overview of loading a taxonomy

This chapter will walk you through the various tasks in creating a graph that can load a taxonomy into an Endeca data store.

The **Add Managed Values** connector allows you to load an externally managed taxonomy (EMT) into an Endeca data store. 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 a reader component (such as the **UniversalDataReader** component) to the graph.

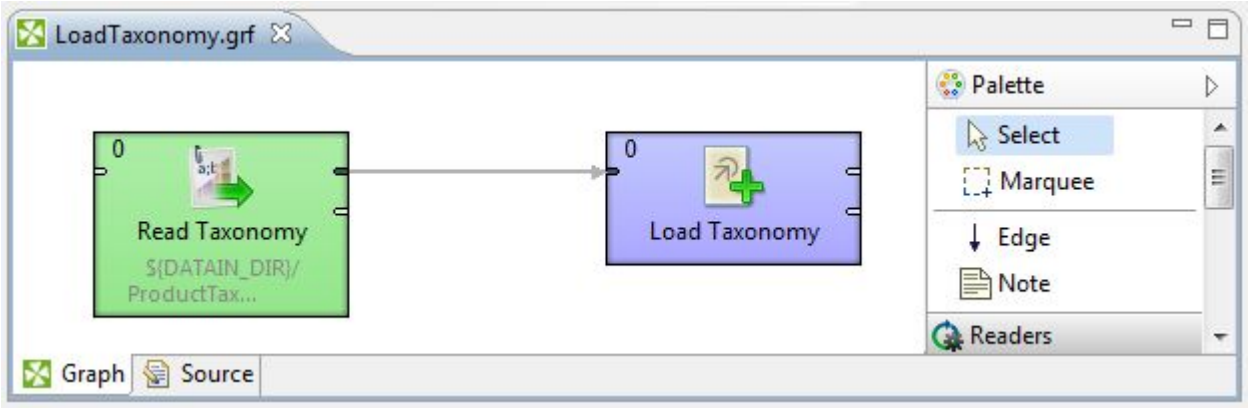
Keep the following two items in mind when adding a taxonomy:

- 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 Dgraph, 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 *Default values for new attributes on page 9*.

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 Integrator Reader components. The format of the source data is explained in a following topic.

Sample taxonomy graph

The following two-component graph is used as the example for the load-taxonomy procedure:



The graph reads in a CSV file (named `ProductTaxonomy.csv`) and uses an **Add Managed Values** connector to load the data into the Endeca data store.

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:

```
MvalSpec|Displayname|ParentKey|Synonym
```

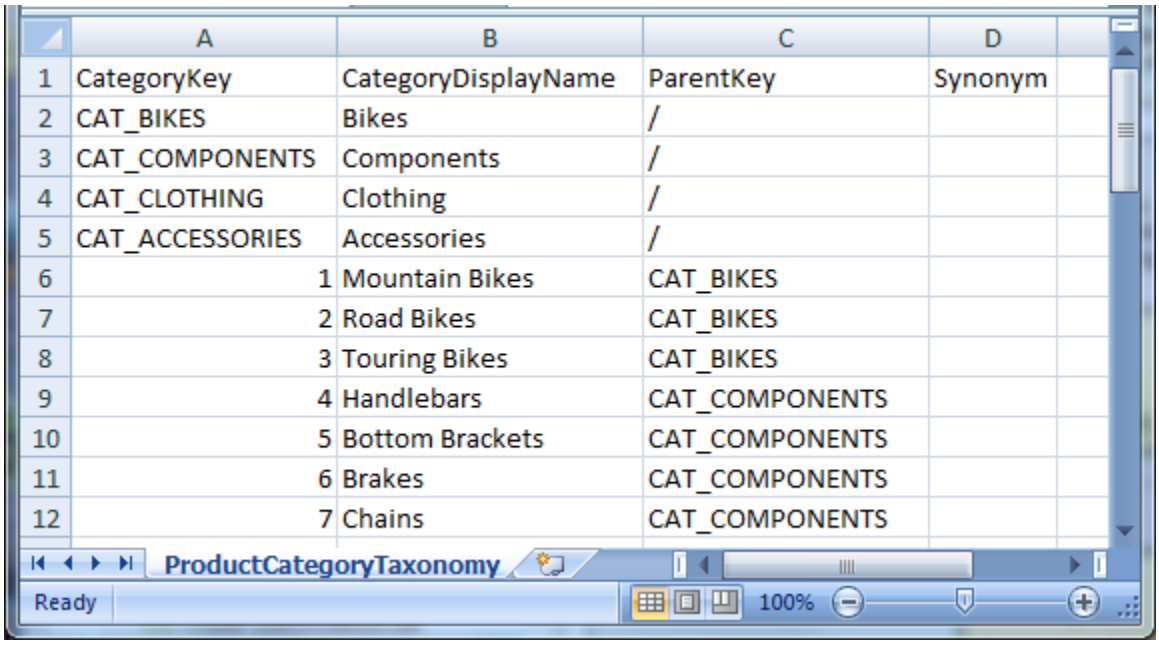
The actual names of the header properties in your input file can be different from the names used here (for example, you can use `CategoryKey` instead of `MvalSpec`). However, the order (positions) of the header properties and their values is crucial. For example, the third position signifies the managed value's parent ID, regardless of the name used for that header property.

The meanings of these header properties are as follows:

Property	Purpose
`MvalSpec`	A unique string identifier for the managed value. This is the managed value spec.
`Displayname`	The name for the managed value.
`ParentKey`	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.

After the header row, the second and following rows in the input file contain managed value data for the managed value properties.

The following image shows the beginning lines of the `ProductCategoryTaxonomy.csv` input file for the taxonomy graph:



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

ProductCategoryTaxonomy

Ready 100%

In this example:

- Four root managed values are created. Their managed value specs are CAT_BIKES, CAT_COMPONENTS, CAT_CLOTHING, and CAT_ACCESSORIES and they all have a `ParentKey` of a forward slash (/) because they are root managed values. Their `CategoryDisplayName` values set the names that will be displayed in the application UI.
- Seven child managed values are created. Three are children of the CAT_BIKES managed value and the other four are children of the CAT_COMPONENTS managed value.

Note that more child managed values are created from the `ProductTaxonomy.csv` specifications. In our example, the file is stored in the **data-in** folder 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

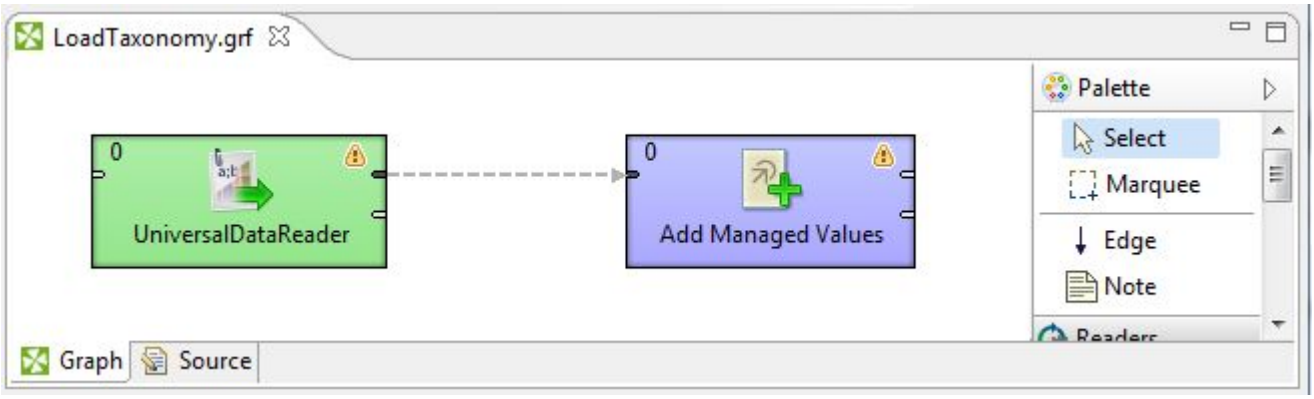
The process requires that you add the **UniversalDataReader** component and the **Add Managed Values** connector to the graph.

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 **Discovery** 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

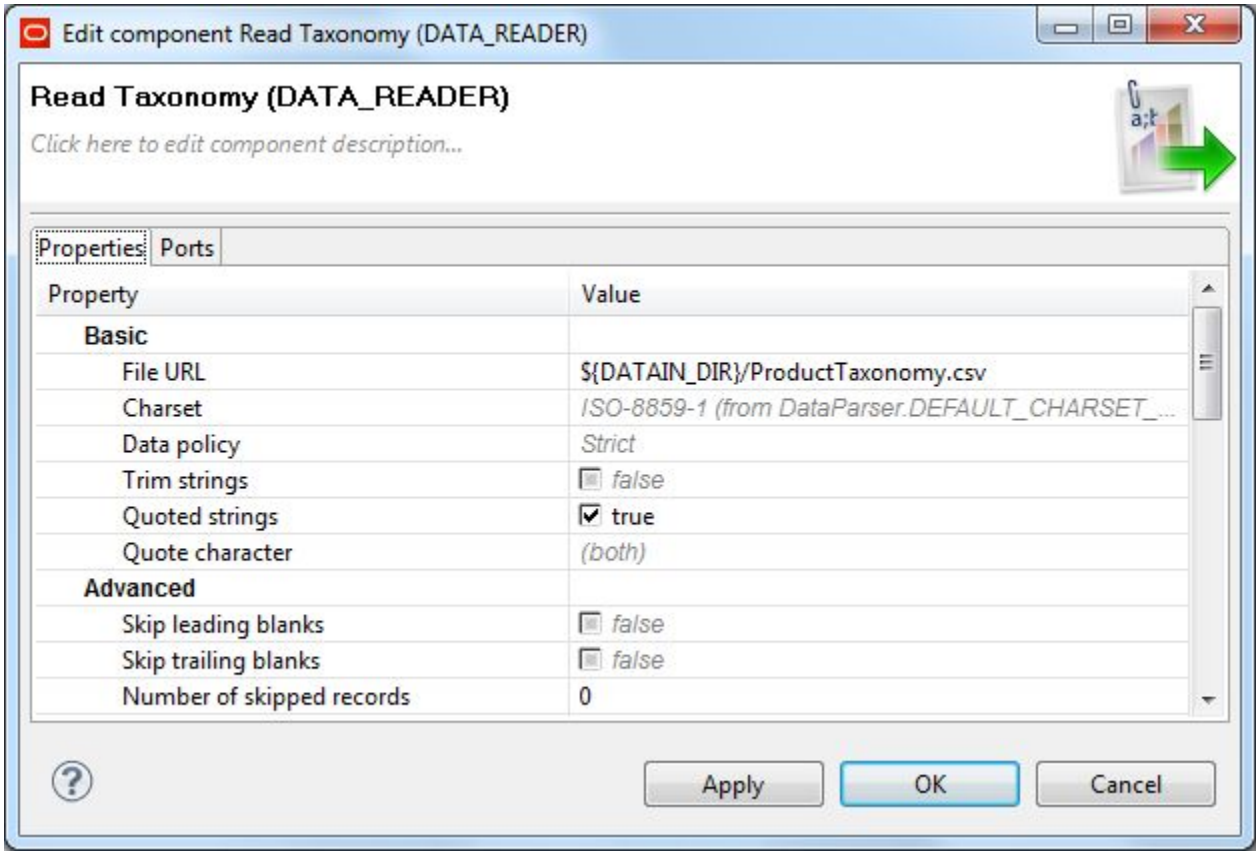
This task describes how to configure the **UniversalDataReader** component to read in the taxonomy data.

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 **config-in** folder (or alternatively, to the **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 **config-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 to its default of 0. (It will automatically be changed to 1 when you configure the metadata for the Reader's Edge.)
5. Optionally, you can use the **Component name** field to provide a customized name (such as "Read Taxonomy") for this component.
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:



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

Configuring the Add Managed Values connector

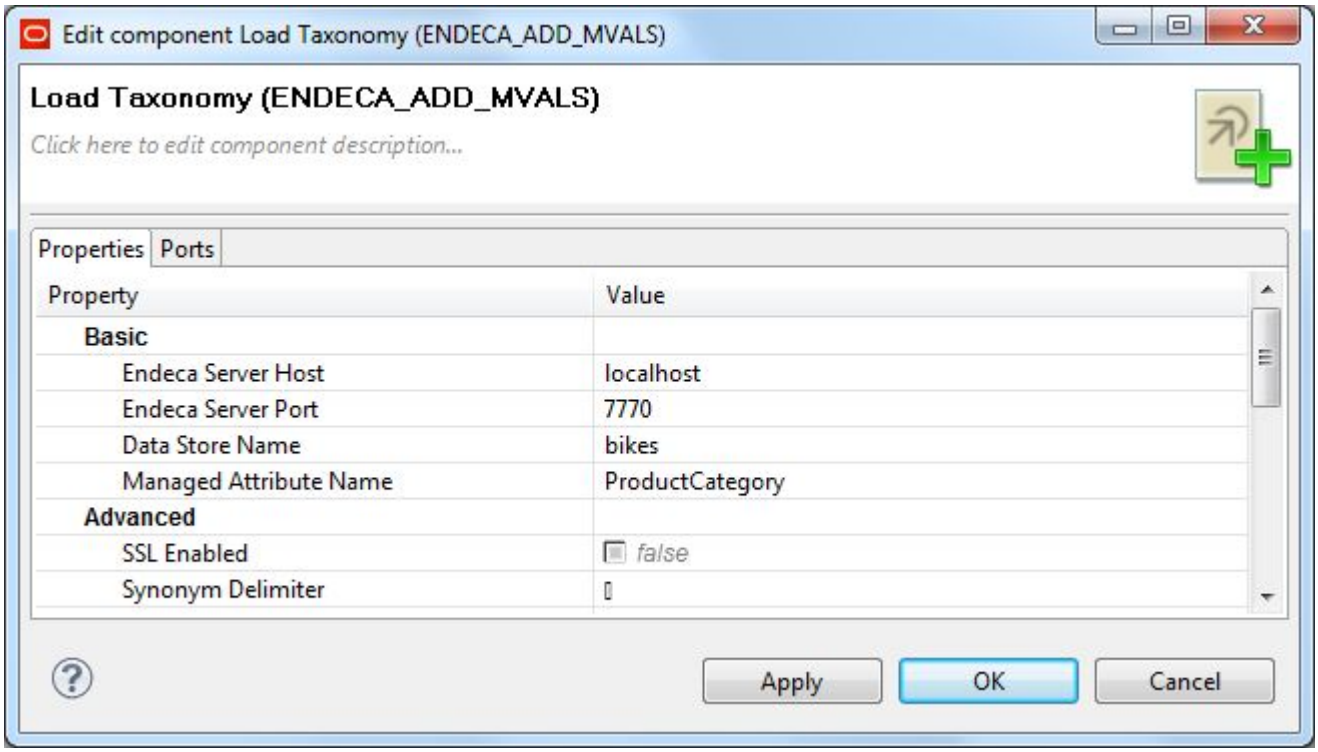
This topic describes how to configure the **Add Managed Values** component.

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 Edit Component dialog is displayed.
2. In the Writer Edit Component dialog, enter these settings:
   (a) **Endeca Server Host**: The host name of the machine on which the Endeca Server is running.
   (b) **Endeca Server Port**: The port on which the Endeca Server is listening.
   (c) **Data Store Name**: The name of the Endeca data store to which the managed values will be added.
   (d) **Managed Attribute Name**: The name of the Endeca managed attribute to which the managed values will be added.
   (e) **SSL Enabled**: Toggle this field to `true` if the Endeca Server is SSL-enabled.
   (f) **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.
   (g) Optionally, you can use the **Component name** field to provide a customized name (such as "Load Taxonomy") for this component.
3. When you have input all your changes, click **OK**.
4. Save the graph.

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



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

Configuring taxonomy metadata

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

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

To configure the Metadata definition for the taxonomy 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 taxonomy source 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**.
   The taxonomy data is loaded into the Metadata Editor.
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** Name field and change the default value to a name such as `Taxonomy`.
   (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 comma in our example).
   (d) When you have input all your changes, click **Finish**.
7. Save the graph.

The Metadata definition for the Edge is now set.

Running the taxonomy graph

After creating the graph and configuring the components, you can run the graph to add the taxonomy to the Endeca data store.

To run the graph to load a taxonomy:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **Add Managed Values** connector and that the Endeca data store has been started.
2. Run the transaction 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 output lists the number of records that were read in by the **UniversalDataReader** component and the number of records that were sent to the Dgraph by the **Add Managed Values** connector.

If you want to run the taxonomy graph within a transaction graph, you can create the transaction graph as described in the topic *Working with Outer Transaction Graphs on page 22*.

Chapter 13

Implementing Text Enrichment

This chapter describes now to add a text enrichment module to your projects.

About Text Enrichment

Text Enrichment prerequisites

Text Enrichment properties file

Creating a Text Enrichment project and graph

Configuring the Text Enrichment Reader

Configuring the Text Enrichment component

Configuring the Add/Update Records connector

About Text Enrichment

The Text Enrichment component provides information extraction and summarization capabilities.

Extracted information include entities (such as people, places, and organizations), quotations, and themes. The Text Enrichment component utilizes the Salience Engine from Lexalytics. Depending on the version of the Salience Engine that you purchased, the engine also provides the ability to extract sentiment from documents at the document, entity, and theme levels.

Supported Text Enrichment features

The following table lists the features supported by the Text Enrichment component. Note that while the Salience Engine supports a larger number of extraction features, only the following ones are supported by the Text Enrichment component.

Text Enrichment feature	Resulting information in the output record
Sentiment Analysis	An overall sentiment score for the current document (computed only if the Sentiment Analysis feature has been enabled).

Text Enrichment feature	Resulting information in the output record
Named Entities	A list of named entities in the current document (computed only if the Named Entities feature has been enabled). The user specifies which types of entities will be extracted. Supported entity types are: • Company (i.e., businesses) • Person • Place (i.e., geographical locations) • Product • Sports • Title • List (for user-defined entities) The output record will have one column per type and each column can have multiple values. Additionally, if the user has enabled Sentiment Analysis, the entities will be added to different groups based on their sentiment scores. The user has to specify the different ranges for the entity sentiment scores. The output record will have one column per range and each column can have multiple values.
Themes	A list of themes in the document (computed only if the Themes feature has been enabled). All meta-themes are added to the output record (the user has to specify the name of the field/property for meta-themes). For any theme that is not a meta-theme, if the theme score is higher than a user-specified threshold, then: • If Sentiment Analysis is enabled, the theme is added to a group based on its sentiment score. The user must specify the different ranges for the sentiment scores. The output record will have one column per range and each column can have multiple values. • Regardless of whether Sentiment Analysis is enabled or disabled, the theme is added to another (i.e., not meta theme) user-specified field.
Quotations	A list of quotes, with their speakers, in the document (computed only if the Quotations feature has been enabled). The user can specify the maximum length of quotes and the name of the field/property in the output record.
Document Summary	A shortened version of the input content so as to best represent the whole content in a limited number of words.

Lexalytics information sources

The Lexalytics Support Web site provides two sources of information on the Salience Engine:

- The Documentation Wiki is at: *http://dev.lexalytics.com/wiki/pmwiki.php*

- The Developer Blog is at: *http://dev.lexalytics.com/blog*

Although both sources are aimed at a developer audience, they can provide useful information for Integrator users who are implementing the Text Enrichment feature.

Text Enrichment prerequisites

This topic lists the prerequisites for your Text Enrichment project.

Salience Engine installation

The Text Enrichment component requires that Version 5 of the Lexalytics Salience Engine be installed on the machine. We recommend that you use Version 5.0.6242 or higher. Use the Lexalytics installation instructions to install the engine.

When configuring the Text Enrichment component, you must specify the paths of the Salience Engine `data` directory and the Salience Engine license file. Make sure that you note the locations of these two items.

User-supplied files for the graph

The user building the Text Enrichment graph first has to supply two files:

- A source input (which can be in a file or a database) that contains the text that is to be analyzed by the Salience Engine. The CSV file used for the sample graph described in this chapter is named `SurveyResponses.csv` and is shipped with the Quick Start project. The sample graph assumes the file is stored in the project's **data-in** folder.
- A Text Enrichment properties file that specifies the configuration used by the Salience Engine. The properties file used for the sample graph is named `textenrichment.properties` (you can use another name for your file) and is stored in the project's root.

The Text Enrichment properties file is described in greater detail in the following section of this chapter.

Text Enrichment properties file

The Text Enrichment properties file specifies the configuration used by the Salience Engine to analyze a document and extract information.

The Text Enrichment properties file enables/disables the extraction features (such as whether quotations are extracted), specifies the field names in the input file that contain the data to be analyzed, and provides other configuration settings. The major configuration properties specify the following:

- Whether a specific feature should be enabled or disabled. For example, Named Entity extraction will not be performed unless this feature is enabled.
- Scoring thresholds.
- The field names to be used in the output.

Note that the spelling and case of the configuration properties must be used as shown in the table below.

The meanings of the configuration properties are described in the following tables. Note that some of the setting values are **user-variable** (which means the user can customize the name or setting), while others must use the value as listed in the table.

Global Sentiment Analysis property	Meaning
`te.sentiment-analysis.enabled`	If set to `true` (the default), Sentiment Analysis is enabled on a global basis for document, entity, and theme types. Note that to use this feature, you must have installed the version of the Salience Engine that supports Sentiment Analysis.

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

Named Entity property	Meaning
`te.entity.enabled`	If set to `true` (the default), Named Entity Extraction is enabled. If set to `false`, Named Entity Extraction is disabled.
`te.entity.types`	Sets the types of named entities to extract. Supported types are: • `Company` • `Person` • `Place` • `Product` • `Sports` • `Title` • `List` (must be used for if you have user-defined entities) The default types are `Person`, `Company`, and `Product`. Each configured entity type is written to a target field whose name is made up of the entity type (such as `Person`) prefixed by the `te.entity-sentiment.field` value.

Named Entity property	Meaning
`te.entity.field-prefix`	Sets the prefix name that is used to determine the final field names for the named entities. The field name is user-variable, and `Entities` is the default. For example, if you set this value to **Entities** and you configure `Person` and `Company` entity types to be extracted, then **EntitiesPerson** and **EntitiesCompany** will be the two target field names in the output records. If you have user-defined entities, then all the user-defined entities are put in the **EntitiesList** target field.
`te.entity-sentiment.enabled`	If set to `true` (the default), Sentiment Analysis is enabled for entities and a sentiment score is computed for the entities. If set to `false`, Entity Sentiment Analysis is disabled.
`te.entity-sentiment.cut1.label`	Sets the value for this `cut1` label. This will be the column name for the negative entities to be extracted. The field name is user-variable, and `EntitiesNegative` is the default. Negative entities are entities with a score less than the negative threshold.
`te.entity-sentiment.cut1.value`	Sets the `EntitiesNegative` threshold. The value is user-variable (for example, a value of `-0.1` can be used). Entity-sentiment scores that are less than this value are written to the `EntitiesNegative` record field.
`te.entity-sentiment.cut2.label`	Sets the value for this `cut2` label. This will be the column name for the neutral type of entity sentiments to be extracted. Neutral entities are entities with a sentiment score between the negative and positive thresholds. The field name is user-variable, and `EntitiesNeutral` is the default.
`te.entity-sentiment.cut2.value`	Sets the `EntitiesPositive` threshold. The value is user-variable (for example, a value of `0.1` can be used). Entity-sentiment scores that are greater than this value are written to the `EntitiesPositive` record field.
`te.entity-sentiment.cut3.label`	Sets the value for this `cut3` label. This is the column name for the positive type of entity sentiments to be extracted. Positive entities are entities with a score greater than the positive threshold. The field name is user-variable, and `EntitiesPositive` is the default.

Theme property	Meaning
`te.theme.enabled`	If set to `true` (the default), Theme Extraction is enabled. If set to `false`, Theme Extraction is disabled.

Theme property	Meaning
`te.theme.field`	Sets the target field name in the output records in which kept theme names are written. The field name is user-variable, and `Themes` is the default. Kept themes are those themes whose score is higher than the `te.theme.score.threshold` setting and have made the `te.theme.keep-max` cut-off list.
`te.theme.score.threshold`	Sets a score threshold for keeping themes. That is, only keep themes with a score greater than this threshold. The value is user-variable, and `1.0` is the default.
`te.theme.keep-max`	Sets a threshold for keeping the best themes. That is, of those themes that are above the `te.theme.score.threshold` setting, only keep the themes with the best scores. The value is user-variable, and the default is `100`.
`te.theme-sentiment.enabled`	If set to `true` (the default), Sentiment Analysis is enabled for themes and a sentiment score is computed for the themes. If set to `false`, Theme Sentiment Analysis is disabled.
`te.theme-sentiment.cut1.label`	Sets the value for this `cut1` label. The field name is user-variable, and `ThemesNegative` is the default. Negative themes are themes with a score less than the negative threshold.
`te.theme-sentiment.cut1.value`	Sets the `ThemesNegative` threshold. The value is user-variable (for example, a value of `-0.1` can be used).
`te.theme-sentiment.cut2.label`	Sets the value for the `cut2` label. The field name is user-variable, and `ThemesNeutral` is the default. Neutral themes are themes with a sentiment score between the negative and positive thresholds.
`te.theme-sentiment.cut2.value`	Sets the `ThemesPositive` threshold. The value is user-variable (for example, a value of `0.1` can be used). ThemesPositive are themes with a score greater than the positive threshold.
`te.theme-sentiment.cut3.label`	Sets the value for this `cut3` label. The field name is user-variable, and `ThemesPositive` is the default. Positive themes are themes with a score greater than the positive threshold.
`te.meta-theme.field`	Sets the target field name in the output records in which the meta-themes are written. The field name is user-variable, and `ThemesMeta` is the default. Meta-themes are a list of themes in the document.

Theme property	Meaning
`te.meta-theme.frequency.threshold`	Sets a score threshold for keeping meta-themes. That is, only keep meta-themes with a score greater than this threshold. The value is user-variable, and `1.0` is the default.

Quotation property	Meaning
`te.quotation.enabled`	If set to `true` (the default), Quoted Context Extraction is enabled. If set to `false`, Quoted Context Extraction is disabled.
`te.quotation.field`	Sets the target field name in the output records in which quoted content is written. The field name is user-variable, and the default is `Quotes`.
`te.quotation.max-length`	Sets the maximum length (in characters) of a quotation. The default length is 200. Note that if the quotation in the source field is longer than this setting, the source quotation is not written to the target field.

Document Summary property	Meaning
`te.summary.field`	Sets the column name in the output file in which the summarization of the input content is written. The field name is user-variable, and the default is `Summary`.
`te.summary.length`	Sets the document summary length in sentences. The default length is 3 sentences.

Basic Custom properties	Meaning
`te.salience.userdataDirectory`	Takes an absolute path to a directory that contains a user-created data dictionary.
`te.sentiment.setSentimentDictionary`	Takes an absolute path to a user-created dictionary that will be used as the sentiment dictionary for the Salience Engine (that is, this dictionary overrides the default Salience sentiment dictionary).

Basic Custom properties	Meaning
`te.sentiment.addSentimentDictionary`	Takes an absolute path to a user-created dictionary that will be used in addition to the current sentiment Analysis dictionary: • If `te.sentiment.setSentimentDictionary` has been used, then the additional dictionary is added to the first user-created sentiment dictionary. • If `te.sentiment.setSentimentDictionary` has not been used, then the additional dictionary is added to the default Salience sentiment dictionary.

Advanced Custom options

Besides the `te.*` configuration properties listed above, you can set other options by using Salience API methods. These advanced options will come from these classes:

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

where *xxx* is the name of the method, such as `Salience.Options.Base.setFailLongSentence`.

Information on these classes is available in the Lexalytics Salience 5 Javadoc:

http://dev.lexalytics.com/doc/java-se5.0

When using these API extension points, be aware that they are not parsed by the Endeca Text Enrichment software, but instead are passed as-is to the Salience Engine.

Sentiment activator interaction

There are four configuration activation properties that control Sentiment Analysis:

- `te.sentiment-analysis.enabled`: Enables or disables Sentiment Analysis on a global basis.
- `te.document-sentiment.enabled`: Enables or disables Document Sentiment Analysis.
- `te.entity-sentiment.enabled`: Enables or disables Entity Sentiment Analysis.
- `te.theme-sentiment.enabled`: Enables or disables Theme Sentiment Analysis.

If `te.sentiment-analysis.enabled` is set to `true`, then Sentiment Analysis is enabled on a global basis. If set to `false`, then the document, entity, and theme sentiment activators will also be treated as `false`, regardless of their actual settings. In other words, a setting of `false` means that the Salience Engine will not perform Sentimental Analysis of any type.

If `te.sentiment-analysis.enabled` is set to `true`, then you can disable Document, Entity, and Theme Sentiment Analysis in any combination. For example, if you are not interested in Entity Sentiment Analysis, you can enable Document and Theme Sentiment Analysis but disable Entity Sentiment Analysis. This activation scheme thus lets you customize the Sentiment Analysis feature for your specific needs.

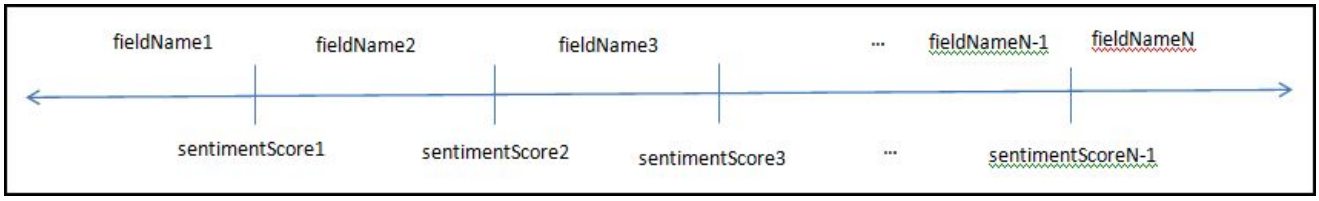
Customizing the theme and/or entity cuts

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

Named-entities are added to different user-configured fields based on their sentiment scores. You can configure the various output fields by specifying range-thresholds and field-names as follows (names in bold-face are user-supplied names):

```
te.entity-sentiment.cut1.label = fieldName1
te.entity-sentiment.cut1.value = sentimentScore1
te.entity-sentiment.cut2.label = fieldName2
te.entity-sentiment.cut2.value = sentimentScore2
te.entity-sentiment.cut3.label = fieldName3
te.entity-sentiment.cut3.value = sentimentScore3
...
te.entity-sentiment.cut-1.label = fieldNameN-1
te.entity-sentiment.cut-1.value = sentimentScoreN-1
te.entity-sentiment.cutN.label = fieldNameN
```

This field schema can be represented graphically by this illustration:



The above configuration specifies **N** different fields into which the named-entities will be mapped based on their sentiment-scores. Any entity whose sentiment-score is between `MIN_FLOAT` and **sentimentScore1** will be placed in **fieldName1**. Then, any entity whose sentiment-score is between **sentimentScore1** and **sentimentScore2** will be placed in **fieldName2**, and so on. Finally, any entity whose sentiment score is between **sentimentScoreN-1** and `MAX_FLOAT` will be placed in **fieldNameN**.

The label can be any string that is allowed to be a field-name (e.g., `EntitiesBucket1`). The value can be any floating-point number.

Note: There are no default values for the above-mentioned properties in the Text Enrichment component. Therefore, a property will not be used unless you add it to the properties file, with a named label and a floating-point value.

The following is an example configuration:

```
te.entity-sentiment.cut1.label = EntitiesNegative
te.entity-sentiment.cut1.value = -0.1
te.entity-sentiment.cut2.label = EntitiesNeutral
te.entity-sentiment.cut2.value = 0.1
te.entity-sentiment.cut3.label = EntitiesPositive
```

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

Sample Text Enrichment properties file

```
# Enable Sentiment Analysis on global basis
te.sentiment-analysis.enabled = true

# Enable Document Sentiment
```

```
te.document-sentiment.enabled = true
te.document-sentiment.field = DocumentSentiment

# Enable Entity extraction
te.entity.enabled = true
# Entity types to allow and their prefix
te.entity.types = Person, Company, Product, Place
te.entity.field-prefix = Entities
# Entity sentiment goes -0.1 < s < 0.1
te.entity-sentiment.enabled = true
te.entity-sentiment.cut1.label = EntitiesNegative
te.entity-sentiment.cut1.value = -0.1
te.entity-sentiment.cut2.label = EntitiesNeutral
te.entity-sentiment.cut2.value = 0.1
te.entity-sentiment.cut3.label = EntitiesPositive

# Enable Theme extraction
te.theme.enabled = true
te.theme.field = Themes
# Only keep themes with score greater than the threshold
te.theme.score.threshold = 0.0
# Of those that are above the threshold, only keep the best 50
te.theme.keep-max = 50
```

```
# Theme sentiment goes -0.1 &lt; s &lt; 0.1
te.theme-sentiment.cut1.label = ThemesNegative
te.theme-sentiment.cut1.value = -0.1
te.theme-sentiment.cut2.label = ThemesNeutral
te.theme-sentiment.cut2.value = 0.1
te.theme-sentiment.cut3.label = ThemesPositive
# Set meta-theme field and only keep those above0.1
te.meta-theme.field = ThemesMeta
te.meta-theme.frequency.threshold = 0.1

# Enable Quotation extraction
te.quotation.enabled = true
te.quotation.field = Quotes
# Max length of a quotation, in characters
te.quotation.max-length = 400

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

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

Creating a Text Enrichment project and graph

This topic describes how to create a graph for text enrichment.

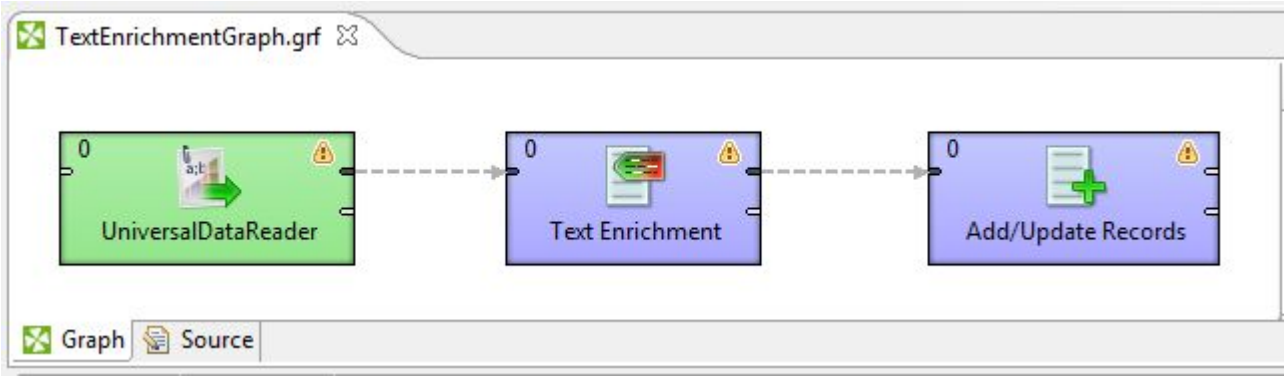
This procedure assumes that you have created a Text Enrichment properties file.

This sample graph uses the **Add/Update Records** connector to load the tagged records into an Endeca data store. However, during development, you can use a **UniversalDataWriter** component to write the output to a disk file. After you are satisfied with the output, you can replace the **UniversalDataWriter** component with the **Add/Update Records** connector or any other Endeca connector that writes to an Endeca data store. These Endeca connectors are described elsewhere in this guide.

To create a Text Enrichment project and a corresponding graph:

1. Create an empty project with the **File>New>ETL Project** command.
   In our example, **TextEnrichment** is the name of the project.
2. In the **TextEnrichment** project, create an empty graph with the **File>New>ETL Graph** command.
3. Copy the Text Enrichment input file into the project's **data-in** folder.
4. Copy the Text Enrichment properties file into the project folder.
5. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag the **UniversalDataReader** component from the **Readers** section.
   (b) Drag the **Text Enrichment** component from the **Discovery** section.
   (c) Drag the **Add/Update Records** component from the **Discovery** section.
6. In the Palette pane, click **Edge** and use it to connect the components.
7. From the File menu, click **Save** to save the graph.

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



The next tasks are to configure these components.

Configuring the Text Enrichment Reader

This task describes how to configure the **UniversalDataReader** component to read in the Text Enrichment source file.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the Text Enrichment source file (named `SurveyResponses.csv` in our example) to the project's **data-in** folder.

To configure the **UniversalDataReader** component for the text enrichment source 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) In the URL Dialog box, click the **Workspace view** tab and then double-click the **data-in** input file folder.

(d) Select the configuration input file and click **OK**.

3. Set the **Number of skipped records per source** field to 1 (so that the source file's header property field names are not read in).
4. Optionally, you can use the **Component name** field to provide a customized name for this component.
5. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
6. Save the graph.

The next task is to configure the Reader's Edge.

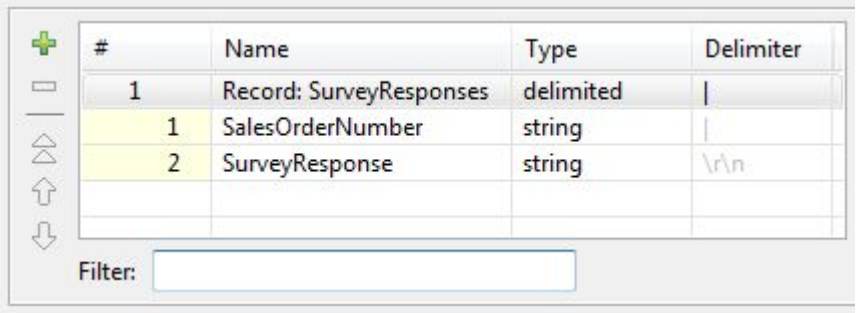
Configuring the Reader Edge

This task describes how to configure the Edge metadata for the Reader component.

This procedure assumes that you are using the `SurveyResponses.csv` sample file as your text enrichment source file. This sample file has two columns of data: `SalesOrderNumber` and `SurveyResponse`.

To configure the Reader Edge component:

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:
   (a) Double-click the input folder.
   (b) Select the text enrichment source file and click **OK**.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed.
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 Record pane, you should change the **recordName1** default value to a name that is appropriate for your data.
   At this point, the Record pane should look like this:



7. When you have input all your changes, click **Finish**.
8. Save the graph.

Configuring the Text Enrichment component

This topic describes how to configure the **Text Enrichment** component.

This procedure assumes that you have created a graph and added the **Text Enrichment** component.

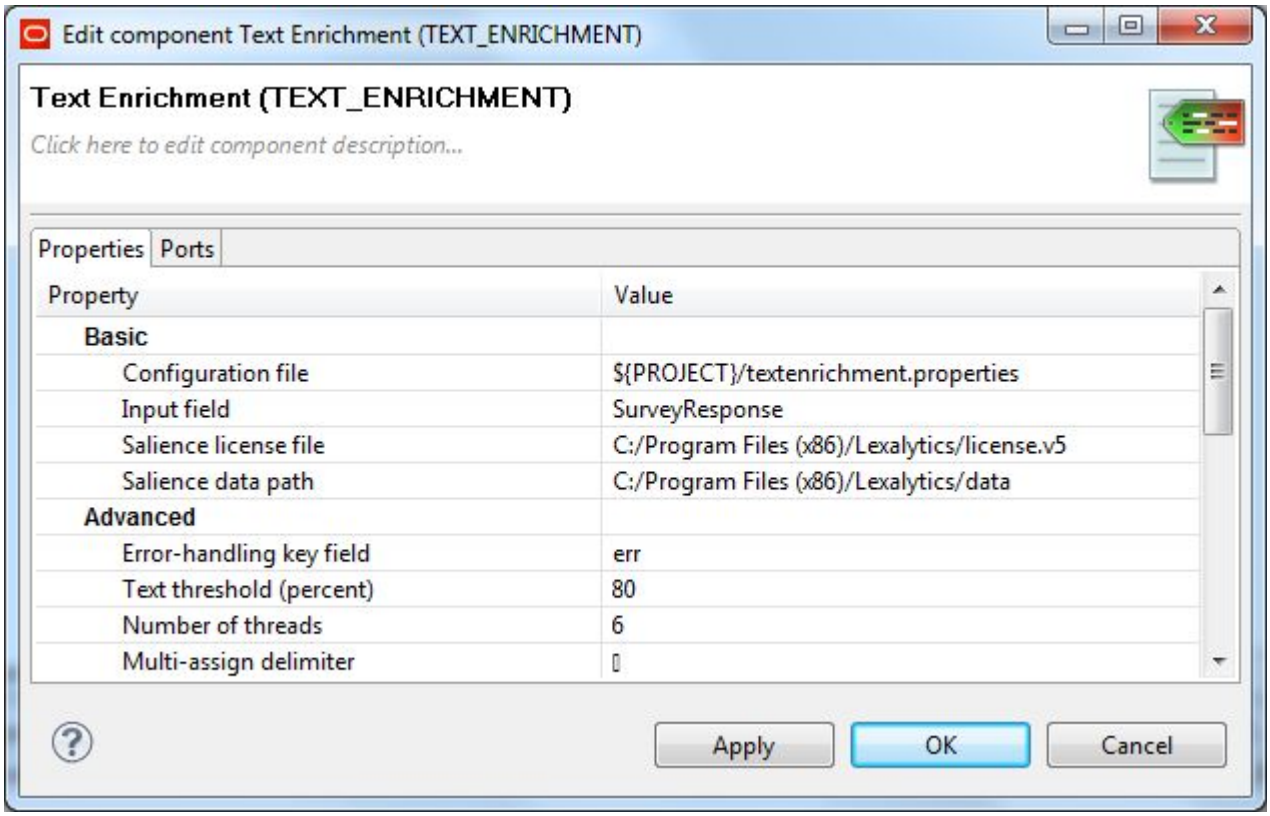
Note: The sorting of the output data from the **Text Enrichment** component is non-deterministic. Therefore, if your project requires a consistently-sorted output, you should add an **ExtSort** component downstream from the **Text Enrichment** component.

To configure the **Text Enrichment** component:

1. In the Graph window, double-click the **Text Enrichment** component.
   The Edit Component dialog is displayed.
2. In the Text Enrichment Edit Component dialog, enter these mandatory settings:
   (a) **Configuration file**: The absolute path of the Text Enrichment properties file.
   (b) **Input field**: The name of the source field (in the data source record file) from whose data entities and sentiment will be extracted.
   (c) **Salience license file**: The absolute path of the Lexalytics Salience license file.
   (d) **Salience data path**: The absolute path of the Lexalytics `data` directory.
   Note that for fields that require a pathname, you can click inside the Value field to display a **...** browse button. You can then browse to the required location.
3. Still in the Text Enrichment Edit Component dialog, you can set these optional properties:
   (a) **Error-handling key field**: A field for error-handling output. Specifying a field name is mandatory. If you do not have a specific error field, you can specify the record specifier (primary key) field name (note that this record specifier field name must exist in the input metadata).
   (b) **Text threshold (percent)**: The minimum percentage of alpha-numeric characters that the input field has to have to be processed. If no threshold is provided, the default value for processing is `80`.
   (c) **Number of threads**: The number of threads the component should run on. If no thread count is provided, the default value is 1.
   (d) **Multi-assign delimiter**: The character that separates multiple values inside data fields. The default is the Unicode DELETE character (\U007F).

4. Optionally, you can use the **Component name** field to provide a customized name for this component.
5. When you have input all your changes, click **OK**.
6. Save the graph.

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



In this example, the data in the input source record's SurveyResponse field will be used for text enrichment.

Configuring the Text Enrichment Edge

This task describes how to configure the Edge metadata for the Text Enrichment component.

The requirement for the configuration is that this Metadata must contain all the fields that were present in the Metadata on the Reader Edge, plus all the fields that the user expects the Text Enrichment component to add (which is dependent on the configuration of the Text Enrichment component).

In this sample Text Enrichment graph, the following properties will be created in the Metadata editor:

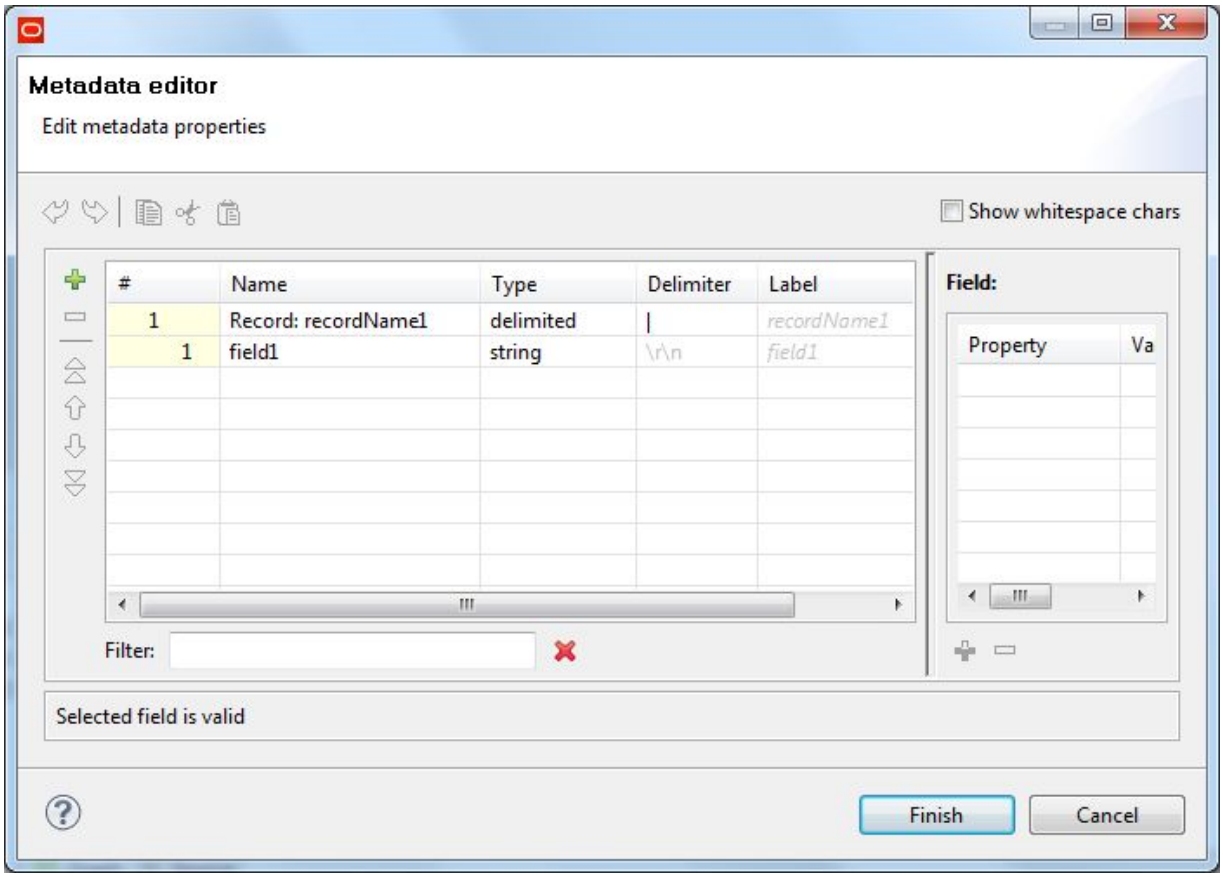
- `DocumentSentiment`
- `SalesOrderNumber`
- `EntitiesPerson`
- `EntitiesProduct`
- `EntitiesCompany`

- `ThemesMeta`
- `ThemesNegative`
- `ThemesPositive`
- `EntitiesNegative`
- `EntitiesNeutral`
- `EntitiesPositive`
- `SurveyResponse`
- `Summary`
- `Quotes`

All of these properties are of Type `string`, except for DocumentSentiment, which is a `decimal`.

To configure the **Text Enrichment** component's Edge:

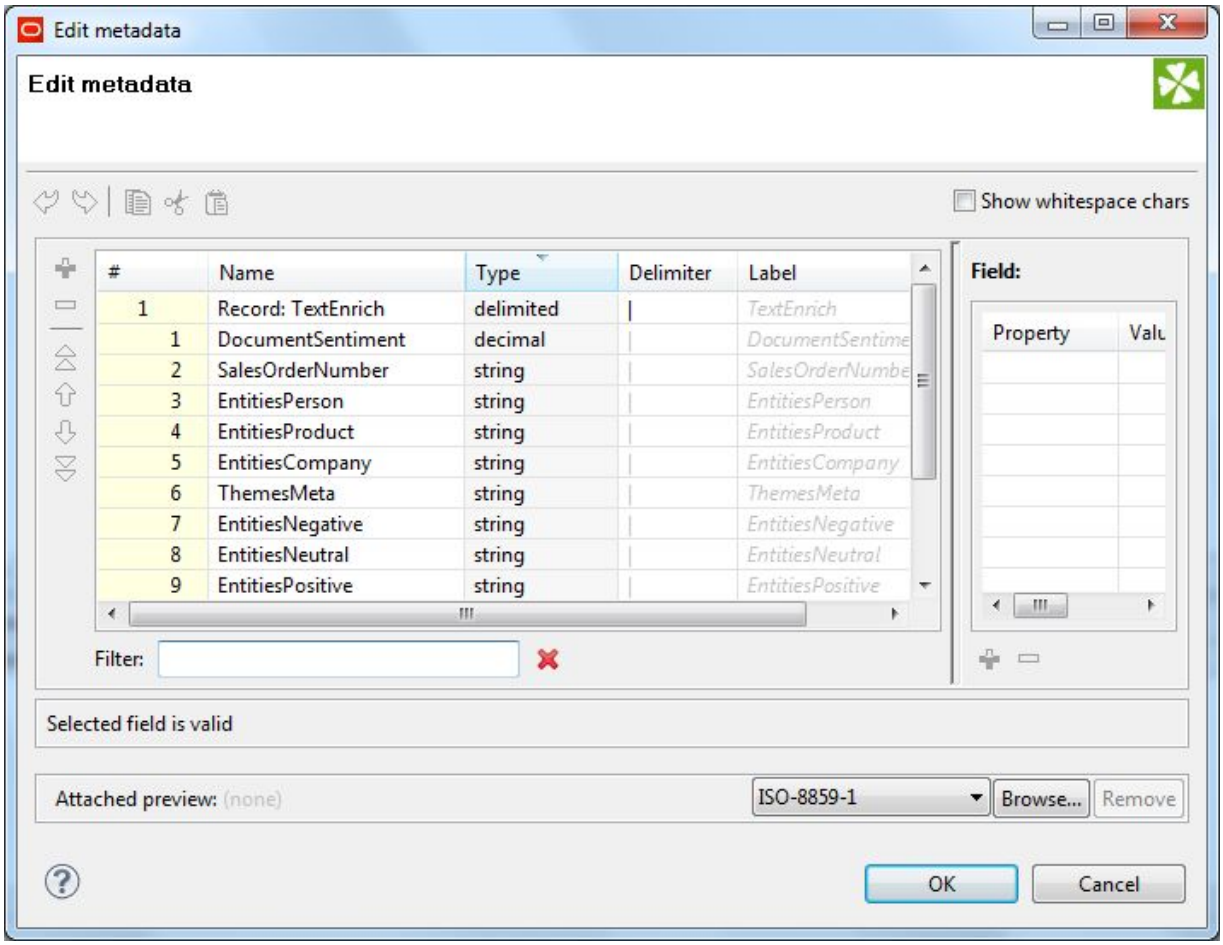
1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field named **field1**.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default name to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file.

3. For the other fields:
   (a) Change the **field1** name to `DocumentSentiment` and change its **Type** to `decimal`.
   (b) Add a new field by using the + (plus sign) control. Name the field `SalesOrderNumber` and leave its **Type** as `string`.
   (c) Repeat step 3b for the rest of the properties listed above.

   At this point, the Metadata editor should look similar to this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the Add/Update Records connector

This task describes how to configure the **Add/Update Records** connector to load the extracted text into an Endeca data store.

To configure the connector, use the procedure described in the topic *Configuring the Add/Update Records connector on page 50*. In our sample graph, we are using the **SalesOrderNumber** attribute as the value for the component's **Spec Attribute** configuration property.

After creating the graph and configuring the components, you can run the graph to extract the entities. You can run the graph by clicking the green circle with white triangle icon in the Tool bar: 

Chapter 14

Using the Text Tagger

This chapter describes how to use the Text Tagger components in your projects.

About the Text Tagger components

Building a Text Tagger Regex graph

Building a Text Tagger Whitelist graph

About the Text Tagger components

The Integrator provides two Text Tagger components.

The two Text Tagger components are:

- The **Text Tagger Regex** component, which uses a regex (regular expression) to match text in a specified text field on incoming records.
- The **Text Tagger Whitelist** component, which uses a tags-rule file to define which terms to match in a specified text field on incoming records. The tags-rule file also specifies a string value to tag on the records when a match is found.

Each component can operate independently of the other (that is, each component can be used by itself in a graph). Alternatively, you can use both components in the same graph.

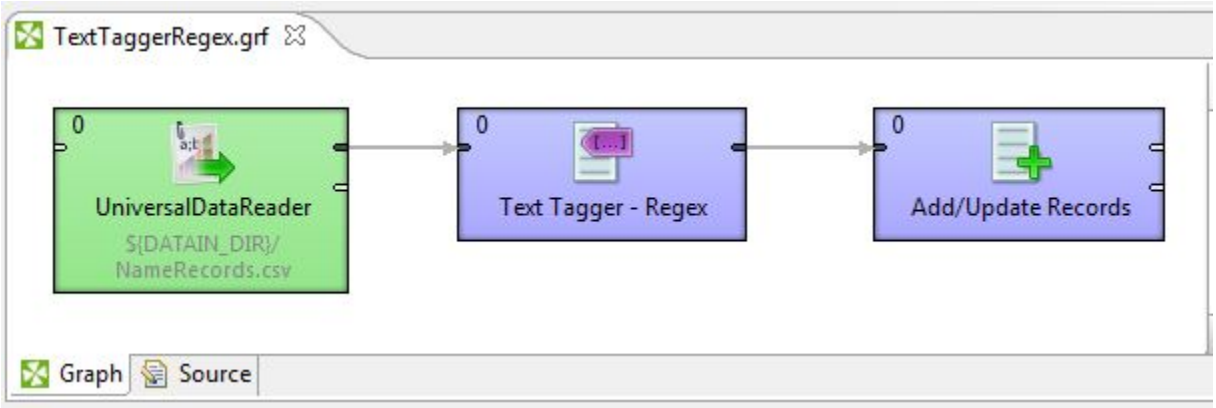
Note: This chapter documents each component separately. Each of the two sample graphs will have only one of the components.

Building a Text Tagger Regex graph

This section describes how to build a graph that uses the **Text Tagger Regex** component.

The graph described in this section is intended to illustrate how to deploy a **Text Tagger Regex** component. The graph is therefore a simple one that reads in a list of people's names, tags those names using a regex search pattern, and renders the results to a CSV file. Note that the names in the input file are in a FirstName-LastName format, but the render pattern will specify that the names are output in a LastName-FirstName format.

The sample three-component graph will look like this when configured:



Multi-assign delimiter and overwritten text

The **Text Tagger Regex** component has a **Multi-assign delimiter** configuration property that specifies the character used as a delimiter in the output text.

The component's **Overwrite Target Field** property determines how the matched text is written to the target field:

- If set to `true`, the matched text overwrites any existing content in the target field. If the returned matched text consists of only one string, then the multi-assign delimiter is not used in the output written to the target field. However, if the returned text consists of multiple matched strings, then each string is delimited by the multi-assign delimiter.
- If set to `false` (which is the default), the matched text output is appended to any existing content in the target field, with each set of data delimited by the multi-assign delimiter. In addition, if a set of data has multiple matched strings, those strings will also be delimited by the multi-assign delimiter.

You can load the multi-assign records into the Endeca data store with an Information Discovery ingest connector, such as the **Add/Update Records** connector. The connector has a **Multi-assign delimiter** property whose value should be set to the same character as the **Multi-assign delimiter** property in the **Text Tagger Regex** component.

Using regular expressions

The **Text Tagger Regex** component uses a regex (regular expression) for its search and render patterns.

The component has two configuration properties that are used for matching character sequences against patterns specified by regular expressions:

- The **Search Pattern** property specifies a regex that a term (in the source field) must match.
- The **Render Pattern** property specifies a regex that is used to render (write out) the matched term into the target field.

The component implements Oracle's `java.util.regex` package to parse and match the pattern of the regular expression. Therefore, the supported regular-expression constructs are the same as those in the documentation page for the `java.util.regex.Pattern` class at this URL:

http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

This means that among the valid constructs you can use are:

- Escaped characters, such **\t** for the tab character.
- Character classes (simple, negation, range, intersection, subtraction). For example, **[^abc]** means match any character except a, b, or c, while **[a-zA-Z]** means match any upper- or lower-case letter.
- Predefined character classes, such as **\d** for a digit or **\s** for a whitespace character.
- POSIX character classes (US-ASCII only), such as **\p{Alpha}** for an alphabetic character, **\p{Alnum}** for an alphanumeric character, and **\p{Punct}** for punctuation.
- Boundary matchers, such as **^** for the beginning of a line, **$** for the end of a line, and **\b** for a word boundary.
- Logical operators, such as **X|Y** for either X or Y.

For a full list of valid constructs, see the `Pattern` class documentation page at the URL listed above.

The following is an example of a useful regular expression that uses the POSIX **\p{Alnum}** construct:

```
^\
p{Alnum}[\p{Alnum}\.\-' ]+$
```

When used with the **Search Pattern** property, this regular expression will match only terms that have at least two characters, start with an alphanumeric character, and contain only alphanumeric, period, dash, apostrophe, and space characters. (The apostrophe is to retain terms such as `O'Malley`).

Creating a Text Tagger Regex graph

This topic describes how to create a graph that uses the **Text Tagger Regex** component.

This sample graph uses a **UniversalDataWriter** component to write the output of the **Text Tagger Regex** component to a disk file.

To create a Text Tagger project and a corresponding **Text Tagger Regex** graph:

1. Create an empty project with the **File>New>ETL Project** command.
   In our example, **TextTagger** is the name of the project.
2. In the project, create an empty graph with the **File>New>ETL Graph** command.
3. Copy the source input file into the project's **data-in** folder.
4. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag the **UniversalDataReader** component from the **Readers** section.
   (b) Drag the **Text Tagger - Regex** component from the **Discovery** section.
   (c) Drag the **Add/Update Records** connector from the **Discovery** section.
5. In the Palette pane, click **Edge** and use it to connect the components.
6. From the File menu, click **Save** to save the graph.

Configuring the Sample Data Reader

This task describes how to configure the **UniversalDataReader** component to read in the Text Tagger Regex source file.

This procedure assumes that you have created a graph and added the **UniversalDataReader** component. It also assumes that you have added the Text Tagger Regex source file (`NameRecords.csv` in our example) to the project's **data-in** folder.

To configure the **UniversalDataReader** component for the Text Tagger Regex source 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) In the URL Dialog box, click the **Workspace view** tab and then double-click the **data-in** input file folder.
   (d) Select the configuration input file and click **OK**.
3. Set the **Quoted strings** field to **true**.
4. Set the **Number of skipped records per source** field to **1** (so that the source file's header property field names are not read in).
5. Optionally, you can use the **Component name** field to provide a customized name for this component.
6. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
7. Save the graph.

Configuring the Reader Edge

This task describes how to configure the Edge metadata for the Reader component.

This procedure assumes that sample input file has two columns of data: `Id` (which are identifiers for the names) and `FirstNameLastName` (which are names of people, in a format where their first names are followed by their last names).

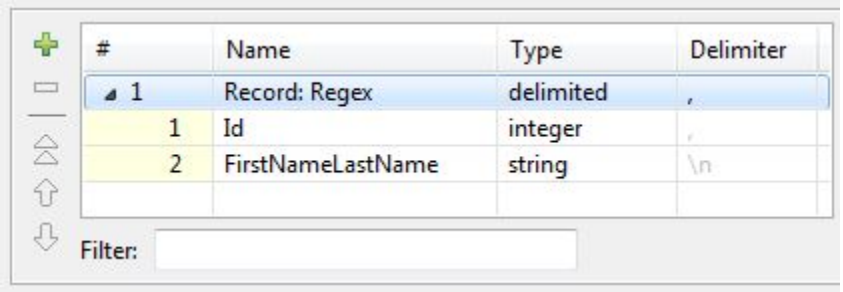
To configure the Reader Edge component for the Text Tagger Regex graph:

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:
   (a) Double-click the input folder.
   (b) Select the Text Tagger source file and click **OK**.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed.
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 Record pane, you can change the **recordName1** default value to a name that is appropriate for your data.
   At this point, the Record pane should look like this:



7. When you have input all your changes, click **Finish**.
8. Save the graph.

Configuring the Text Tagger Regex component

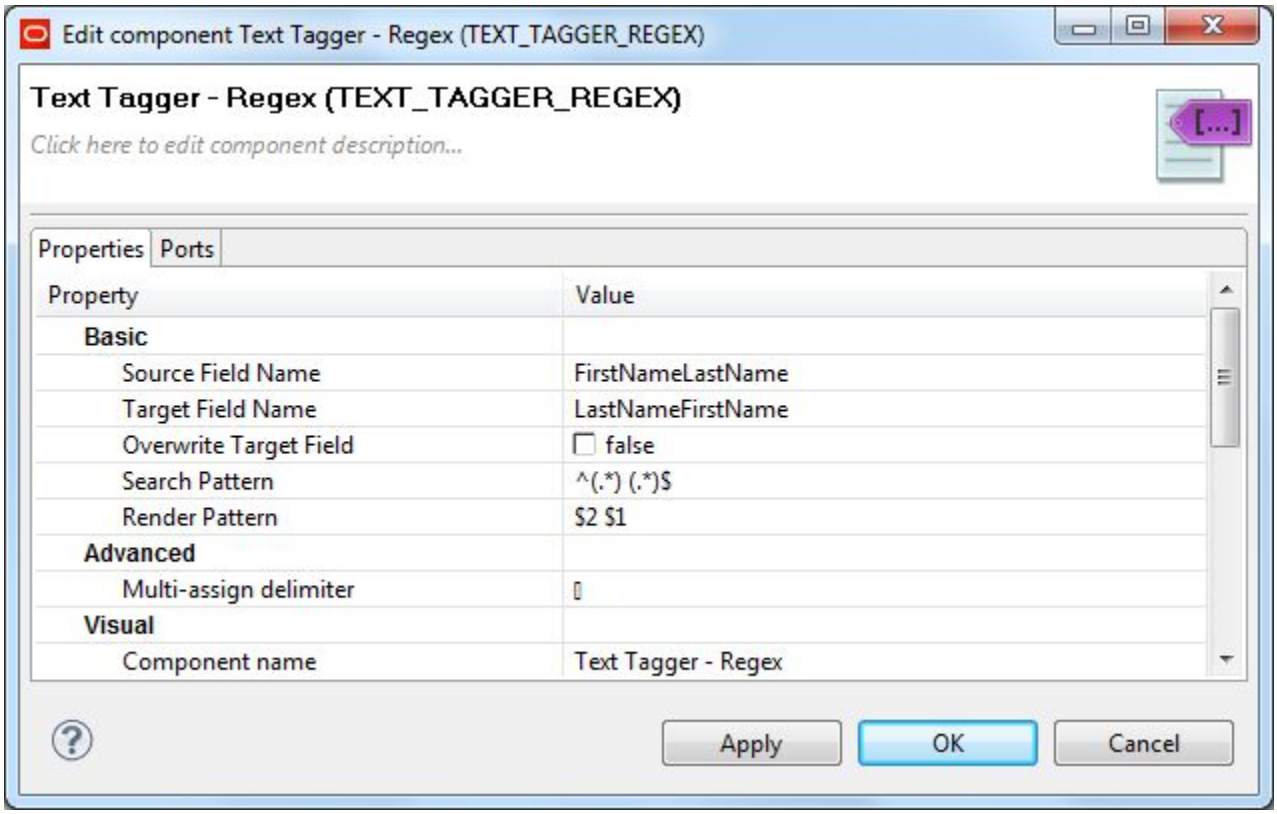
This topic describes how to configure the **Text Tagger Regex** component.

This procedure assumes that you have created a graph and added the **Text Tagger Regex** component.

To configure the **Text Tagger Regex** component:

1. In the Graph window, double-click the **Text Tagger Regex** component.
   The Edit Component dialog is displayed.
2. In the Edit Component dialog, enter these settings:
   (a) **Source Field Name**: The name of the field (in the input records) in which the regex will search for matches.
   (b) **Target Field Name**: The name of the field (in the output records) into which the matched output is written.
   (c) **Overwrite Target Field**: If set to `true`, the content of the target field will be overwritten by the matched output. If set to `false` (the default), the matched output will be appended to the existing content of the target field, with the new content separated from the previous content by the multi-assign delimiter.
   (d) **Search Pattern**: The regex pattern to use when searching for matched text in the source field.
   (e) **Render Pattern**: The regex pattern to use when rendering (writing) the matched output in the target (output) field.
   (f) **Multi-assign delimiter**: The character that separates multiple matched values in the target field.
3. Optionally, you can use the **Component name** field to provide a customized name for this component.
4. When you have input all your changes, click **OK**.
5. Save the graph.

After the component is configured, the Edit Component dialog should look similar to this example:



Configuring the Edge for the Text Tagger Regex component

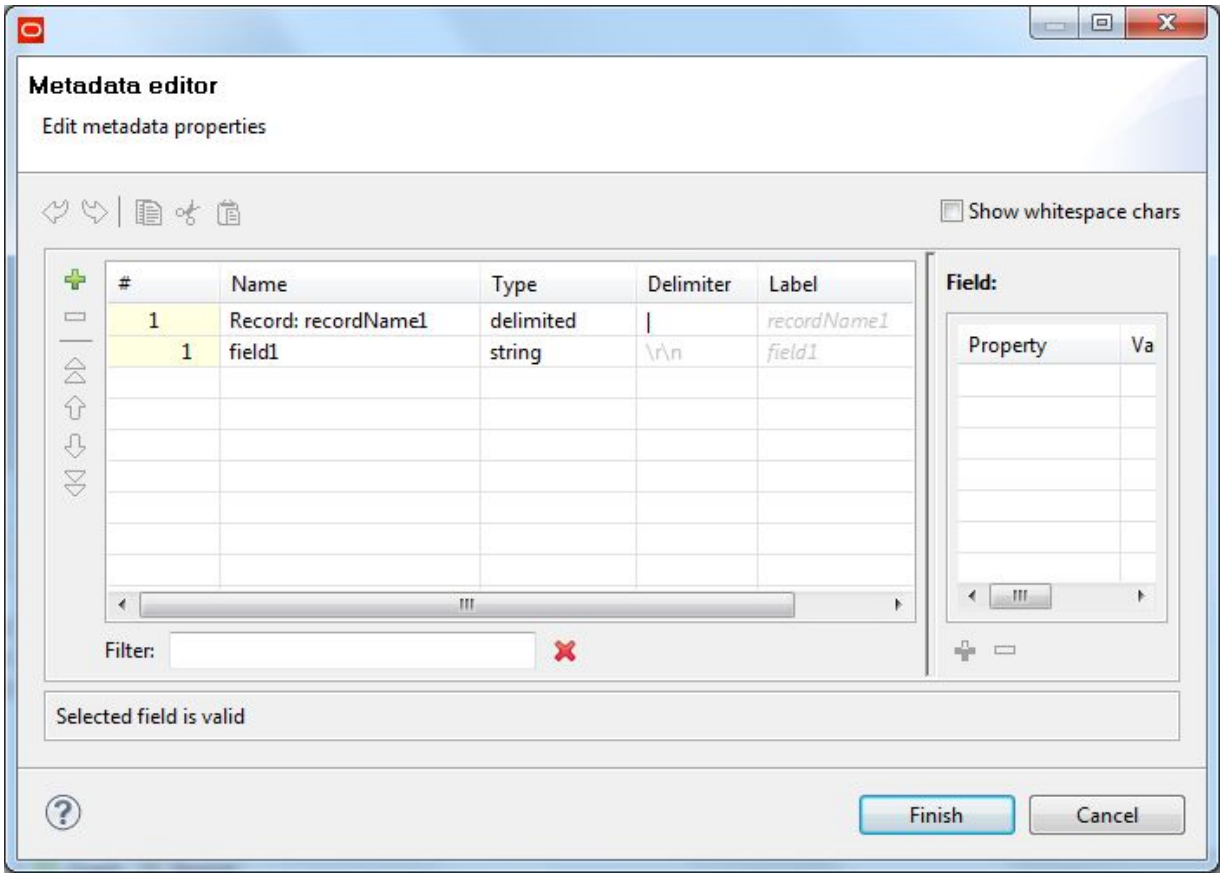
This task describes how to configure the Edge metadata for the Text Tagger Regex component.

In this sample Text Tagger Regex graph, the following properties will be created in the Metadata editor:

- `Id` (identifier for the user names)
- `FirstNameLastName` (the user name in a FirstName-LastName format)
- `LastNameFirstName` (the user name in a LastName-FirstName format, as output by the render pattern)

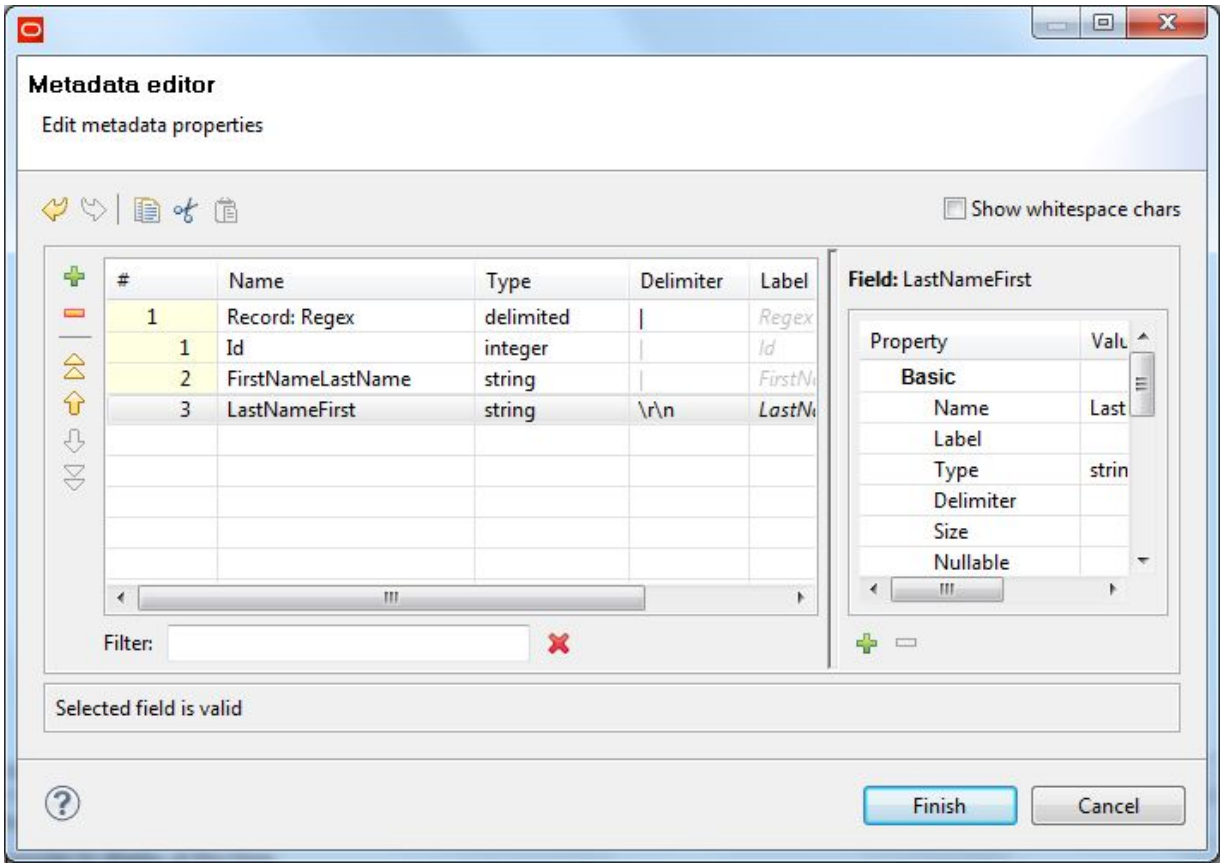
To configure the **Text Tagger Regex** component's Edge:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field named **field1**.



2. In the **Record:recordName1** field:
   (a) Optionally, you can change the **recordName1** default name to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.
   (c) Set the **Delimiter** field to the delimiter character in your input file (which is the comma in our example).
3. For the other fields:
   (a) Change the **field1** name to `Id` and change its **Type** to `integer`.
   (b) Add a new field by using the + (plus sign) control. Name the field `FirstNameLastName` and leave its **Type** as `string`.
   (c) Add another new field by using the + control. Name the field `LastNameFirst` and leave its **Type** as `string`.

   At this point, the Metadata editor should look like this:



4. When you have input all your changes in the Metadata editor, click **Finish**.
5. Save the graph.

Configuring the Add/Update Records connector

This task describes how to configure the **Add/Update Records** connector to load the extracted text into an Endeca data store.

To configure the connector, use the procedure described in the topic *Configuring the Add/Update Records connector on page 50*. In our sample graph, we are using the **SalesOrderNumber** attribute as the value for the component's **Spec Attribute** configuration property.

After creating the graph and configuring the components, you can run the graph to extract the entities. You can run the graph by clicking the green circle with white triangle icon in the Tool bar:

Building a Text Tagger Whitelist graph

The **Text Tagger Whitelist** component tags words and phrases based on a whitelist of terms.

The **Text Tagger Whitelist** component takes a list of search-term/tag-value pairs and searches for the specified terms in the specified unstructured text property (the source field) on records flowing through the graph pipeline. If a term is found on a record, the tag value is tagged on the target property (the target field)

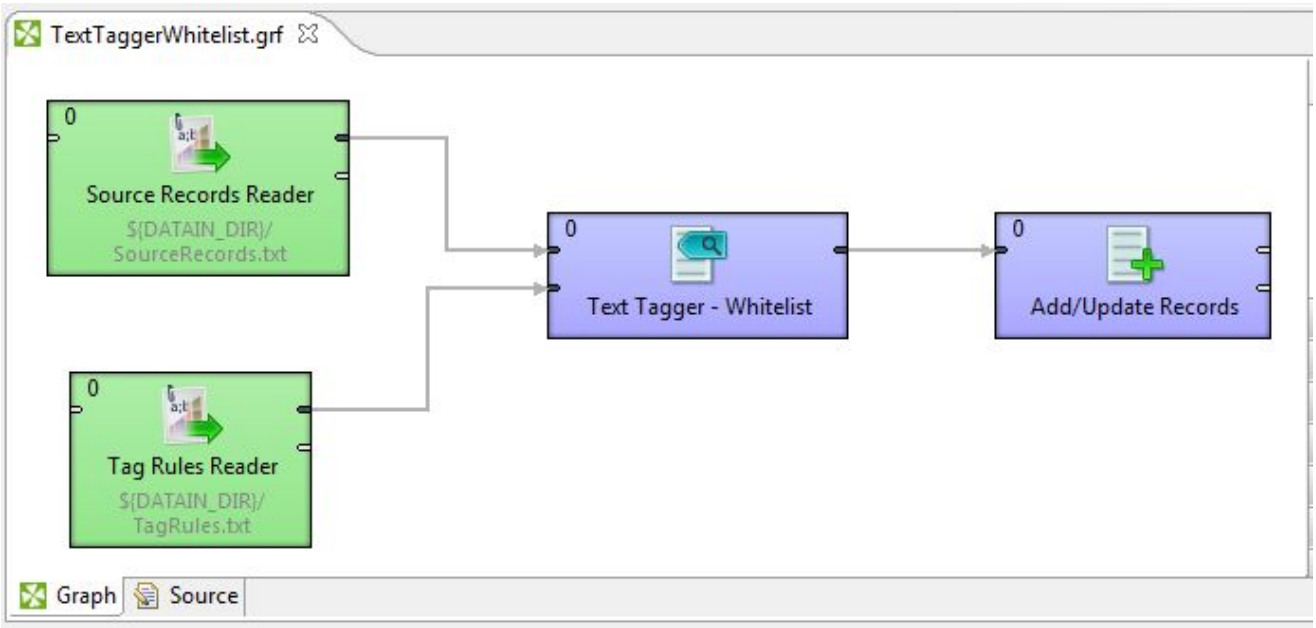
on the record. The list of terms to be tagged is kept in a tag-rules file, which is explained in the following section. You can specify only one tag-rules file as an input for the **Text Tagger Whitelist** component.

The source records are read into the graph via a standard Integrator Reader, such as the **UniversalDataReader** component used in our sample graph. Source records can come from a database, delimited files, and so on. Any restrictions on the record source format are those of the Reader. The major consideration to keep in mind is that the Text Tagger iterates over the properties on each record, expecting that the content of each property is text that can be parsed. The behavior of the Text Tagger is unspecified if the content of a property configured for tagging is not valid (i.e., cannot be parsed).

Note that the **Text Tagger Whitelist** component will use both of its input ports. Port 0 will input the source records (which contain the source property to be searched), while Port 1 will input the tag-rules file.

Sample Text Tagger Whitelist graph

The sample graph described in this section is intended to illustrate how to deploy a **Text Tagger Whitelist** component:



The graph works as follows:

1. The data source records are read in via the **UniversalDataReader** component named "Source Records Reader". This component passes its records into Port 0 of the **Text Tagger Whitelist** component.
2. The tag-rules file is read in via a second **UniversalDataReader** component named "Tag Rules Reader". This component is connected to Port 1 of the **Text Tagger Whitelist** component.
3. The **Text Tagger Whitelist** component processes the incoming data source records and tags them the rules in the tags-rules file.
4. The tagged records are loaded into an Endeca data store via the **Add/Update Records** connector.

The configuration of the components is described in greater detail in this chapter.

Format of the tag-rules source

The metadata schema for the tags-rule source is fixed and uses a specific ordering.

The **Text Tagger Whitelist** component expects the metadata of the tags-rule source to have two properties, in the following order:

- **SearchTerm** (which contains the terms to be searched)
- **TagValue** (which contains the value to be tagged if the search term is found)

Both properties are of type **String** and both must be spelled as listed.

The first row of the tag-rules source is the record header row and must use the same schema as the metadata, as shown in this pipe delimited example:

```
SearchTerm|TagValue
```

The following is a simple example of a tag-rules source file:

```
SearchTerm|TagValue
United States|American topic
France|French topic
Japan|Japanese topic
```

In the example, if the string "United States" is found in the source property, then the target property is tagged with the "American topic" value. Likewise, the "French topic" value is tagged if "France" is found in the source property and "Japanese topic" is tagged if the string "Japan" is found.

If you are using a file (such as a CSV file), the format of the tag-rules file can be any format that is supported by the Integrator Reader component that you are using in your graph. For example, you can use a CSV or plain-text file and read it in with the **UniversalDataReader** component (as is the case in our sample graph).

Note, however, that the tag rules are read into an input port of the **Text Tagger Whitelist** component. By using an input port, the tag rules are not limited to being stored in a flat file. They can come from any source, such as a database. In this case, they can be read in with a database-reader component, such as the **DBInputTable** component.

Search Term lengths

The component's **Search Term Maximum Characters Length** configuration setting specifies the maximum length, in characters, of terms in the tag-rules file. The value of this setting must be larger than the longest search term in the tag-rules file. For example, if the longest search term has 50 characters, then you must set the **Search Term Maximum Characters Length** setting to 51 or greater.

The reason is that as the **Text Tagger - Whitelist** component traverses a source record, it grabs text in chunks that are the size of the **Search Term Maximum Characters Length** setting. For example, if the setting is 50, then the component grabs 50 characters of text at a time. If the search term is longer than the setting, then the search term will never be found because it will be too long to match the text that the component is currently working with.

Creating a Text Tagger Whitelist graph

This topic describes how to create a graph that uses the **Text Tagger Whitelist** component.

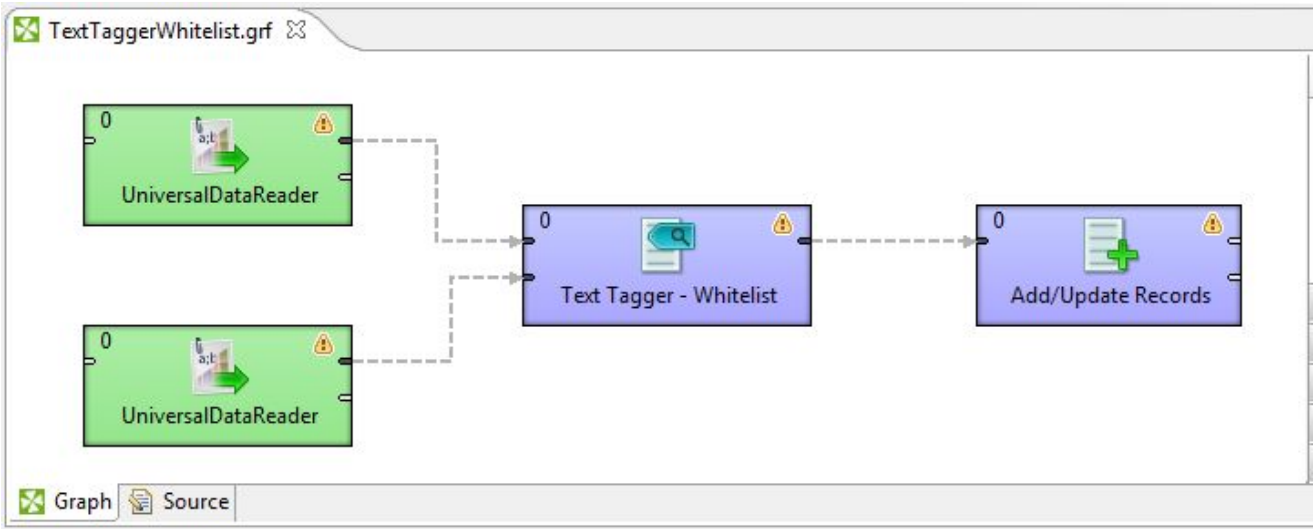
This procedure assumes that you have created a Text Tagger project.

This sample graph uses the **Add/Update Records** connector to load the tagged records into an Endeca data store. However, during development, you can use a **UniversalDataWriter** component to write the output to a disk file. After you are satisfied with the output, you can replace the **UniversalDataWriter** component with the **Add/Update Records** connector or any other Information Discovery connector that writes to an Endeca data store. These Information Discovery connectors are described elsewhere in this guide.

To create a Text Tagger Whitelist graph:

1. In the project, create an empty graph with the **File>New>ETL Graph** command.
2. Copy the source records input file into the project's **data-in** folder.
3. Copy the tag-rules file into the project's **data-in** folder.
4. In the Palette pane, drag the following components into the Graph Editor:
   (a) Drag a **UniversalDataReader** component from the **Readers** section. This first Reader will read in the source records input file.
   (b) Drag a second **UniversalDataReader** component into the Graph Editor. This second Reader will read in the tag-rules file.
   (c) Drag the **Text Tagger - Whitelist** component from the **Discovery** section.
   (d) Drag the **Add/Update Records** connector from the **Discovery** section.
5. In the Palette pane, click **Edge** and use it to connect the components as follows:
   (a) Connect Port 0 (output) of the first **UniversalDataReader** component to Port 0 (input) of the **Text Tagger Whitelist** component.
   (b) Connect Port 0 (output) of the second **UniversalDataReader** component to Port 1 (input) of the **Text Tagger Whitelist** component.
   (c) Connect Port 0 (output) of the **Text Tagger Whitelist** component to Port 0 (input) of the **Add/Update Records** component.
6. From the File menu, click **Save** to save the graph.

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



Configuring the source data Readers

This topic describes how to configure both **UniversalDataReader** components.

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

The configuration of both Reader components is the same, with the exception of the input file for the Reader.

To configure the **UniversalDataReader** components for the Text Tagger Whitelist input files:

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) In the URL Dialog box, click the **Workspace view** tab and then double-click the **data-in** folder.
   (d) Select the input file and click **OK**.
3. Set the **Quoted strings** field to be appropriate to your data. For example, if your source records have quoted strings, set the field to **true**.
4. Set the **Number of skipped records per source** field to **1** (so that the source file's header property field names are not read in).
5. Optionally, you can use the **Component name** field to provide a customized name for the components, such as "Source Records Reader" and "Tag Rules Reader".
6. Click **OK** to apply your configuration changes to the **UniversalDataReader** component.
7. Save the graph.

Configuring the Reader Edges

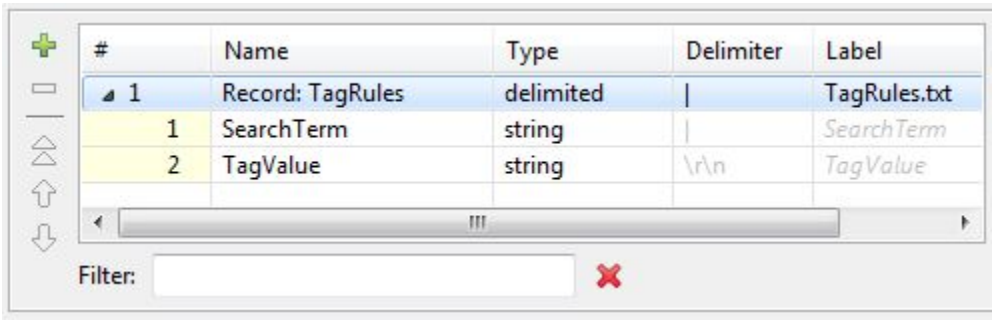
This task describes how to configure the Edge metadata for the Reader components.

To configure the Reader Edge component:

1. Right-click on the Edge for the tag-rules Reader component 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:
   (a) Double-click the input folder.
   (b) Select the Text Tagger tag-rules file and click **OK**.
4. In the Flat File dialog, click **Next**.
   The Metadata Editor is displayed.
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 Record pane, you can change the **recordName1** default value to a name that is appropriate for your data.
   At this point, the Record pane should look like this:



7. When you have input all your changes, click **Finish**.
8. Save the graph.
9. Repeat the above steps for the source records Reader component. Note that at step 3b, you will select the source records file as the input file for this Reader component.

Configuring the Text Tagger Whitelist component

This topic describes how to configure the **Text Tagger Whitelist** component.

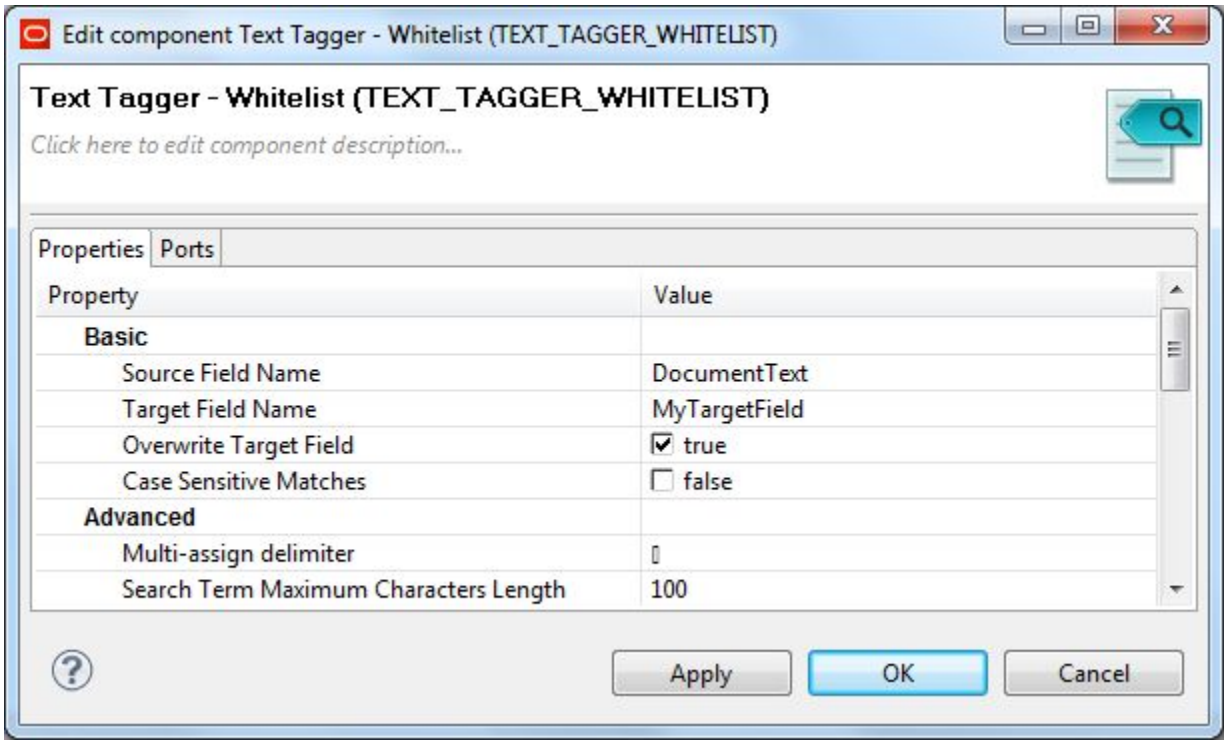
This procedure assumes that you have created a graph and added the **Text Tagger Whitelist** component.

To configure the **Text Tagger Whitelist** component:

1. In the Graph window, double-click the **Text Tagger - Whitelist** component.
   The Edit Component dialog is displayed.
2. In the Edit Component dialog, enter these settings:
   (a) **Source Field Name**: The name of the field (in the input records) that should be matched against the terms in the tag-rules file.
   (b) **Target Field Name**: The name of the field (in the output records) into which the matched output is written.
   (c) **Overwrite Target Field**: If set to `true`, the content of the target field will be overwritten by the matched output. If set to `false` (the default), the matched output will be appended to the existing content of the target field, with the new content separated from the previous content by the multi-assign delimiter. Note that if you set this to `true` and the source records do not have a target field, then the graph will run successfully but you will see warning messages that the target field does not exist in the input metadata.
   (d) **Case Sensitive Matches**: If set to `true`, the search for terms will be case sensitive. If set to `false` (the default), the search will be case insensitive.
   (e) **Multi-assign delimiter**: The character that separates multiple matched values in the target field.
   (f) **Search Term Maximum Characters Length**: Specifies the maximum length, in characters, of terms in the tag-rules file.
3. Optionally, you can use the **Component name** field to provide a customized name for this component.

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

After the component is configured, the Edit Component dialog should look similar to this example:



Configuring the Edge for the Text Tagger Whitelist component

This task describes how to configure the Edge metadata for the Text Tagger Whitelist component.

In this sample Text Tagger Whitelist graph, the following properties will be created in the Metadata editor:

- `Id` (identifier for the source records)
- `DocumentText` (property in source records that contains text to be tagged)
- `OtherText` (property in source records that contains text that will not be tagged)
- `MyTargetField` (new property generated by the **Text Tagger Whitelist** component)

The `MyTargetField` property contains the tagged values (from the tag-rules file) and will be tagged on the output records.

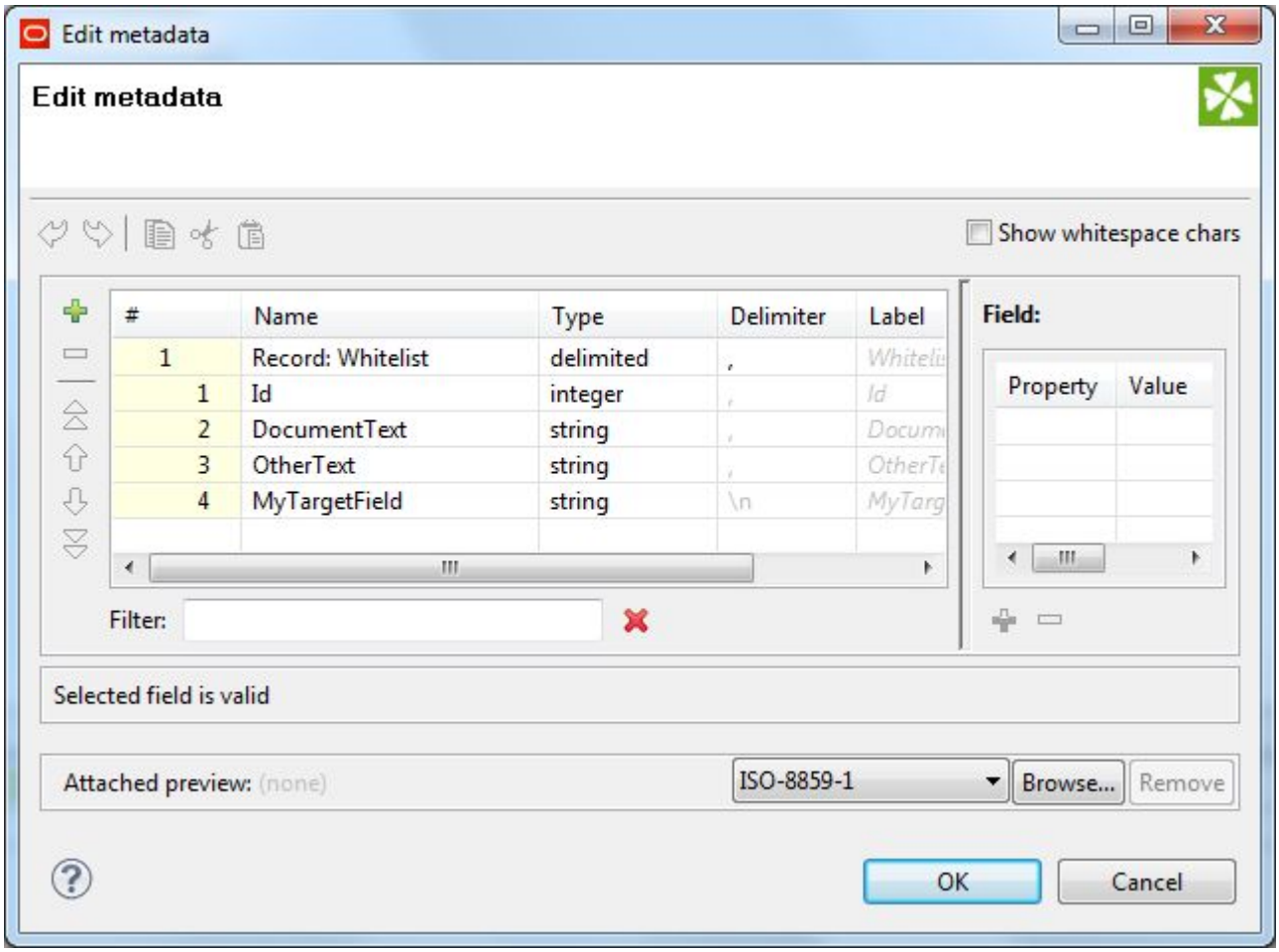
To configure the **Text Tagger Whitelist** component's Edge:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field named **field1**.
2. In the **Record:recordName1** field:
   (a) Optionally, you can change the **recordName1** default name to a name that is appropriate for your data.
   (b) Leave the **Type** field as `delimited`.

(c) Set the **Delimiter** field to the delimiter character that is appropriate for your data.

3. For the other fields:
   (a) Change the **field1** name to `Id` and change its **Type** to `integer`.
   (b) Add new fields by using the + (plus sign) control. Name the fields as appropriate (such as `MyTargetField`) and set their **Type** property to their content (such as `string` for `MyTargetField`).
4. When you have input all your changes in the Metadata editor, click **Finish**.

After configuration, the Metadata editor should look similar to this example:



Configuring the Endeca connector

This task describes how to configure the **Add/Update Records** connector in the graph.

To configure the connector, use the procedure described in the topic *Configuring the Add/Update Records connector on page 50*. In our sample graph, we are using the **Id** attribute as the value for the component's **Spec Attribute** configuration property.

After creating the graph and configuring the components, you can run the graph to tag the records. You can run the graph by clicking the green circle with white triangle icon in the Tool bar: 

Chapter 15

Record Store Reader

This chapter describes how to use the Record Store Reader to read records from a CAS Record Store instance and then load those records into an Endeca data store.

About the Record Store Reader

Adding components to the Record Store Reader graph

Configuring the Record Store Reader component

Generating Edge metadata with the Record Store Wizard

Configuring the Edge for the Record Store Reader component

Configuring the Add/Update Records connector

Running the Record Store Reader graph

About the Record Store Reader

This component reads records from a CAS Record Store instance.

The Endeca Content Acquisition System (CAS) includes components that can crawl data sources for use in an Oracle Endeca Information Discovery application. Data sources include file systems, content management systems, Web servers, and custom data sources. CAS can crawl data sources, convert documents and files to Endeca records, and store them in an Endeca Record Store instance. The Endeca Record Store provides persistent storage for generations of records.

After a CAS crawl stores the records in a Record Store instance, the **Record Store Reader** component can read those records into an Integrator graph. The graph can then use an Endeca data store ingest component (such as the **Add/Update Records** connector) to load the records into an Endeca data store.

Depending on its configuration (via the **CAS Read Type** property), the **Record Store Reader** component can retrieve records as follows:

- The `Full Extract` setting instructs the component to read the latest version of all records in the Record Store. This type of read can be used for a baseline (full load) operation described in the chapter *Full Initial Load of Records on page 32*.
- The `Incremental` setting instructs the component to read records that have been modified or added between the last committed generation in the Record Store and the last generation read by the same client as identified by the component's **CAS Client ID** setting. This type of read can be used for an incremental (delta) operation described in the chapter *Incremental Updates on page 48*.

Optionally, you can add transformation components to the graph that can manipulate or enrich the data in the incoming records. For example, the Text Tagger and/or Text Enrichment components can be incorporated with the Record Store Reader component in a graph.

Record Store Wizard

Integrator includes a Record Store Wizard that makes it easy for you to configure the Edge metadata. The wizard queries the Record Store instance for the Endeca record properties and then populates the Integrator Metadata Editor with those values. Note that the wizard is populating the metadata based on the last committed generation in the Record Store.

When the wizard scans the incoming properties from the Record Store, it replaces any characters that are invalid in Integrator metadata properties. The valid characters are:

- **A** through **Z**
- **a** through **z**
- **0** (zero) through **9**
- _ (underscore)

For example, the `Endeca.Id` is transformed to `Endeca_Id` because periods cannot be used in Integrator metadata properties.

Pre-requisites

The pre-requisite to using the Record Store Reader feature is that you must have installed version 3.0.x of the Endeca Content Acquisition System. With CAS, you must have created a named Record Store instance and run a Web, CMS, or file system crawl to populate the Record Store.

For full information on the Record Store, see the *Endeca CAS Developer's Guide*. Note that you can download the Endeca Content Acquisition System documentation set from the OTN (Oracle Technology Network) Product Page *http://www.oracle.com/technetwork/middleware/endeca/overview*.

Adding components to the Record Store Reader graph

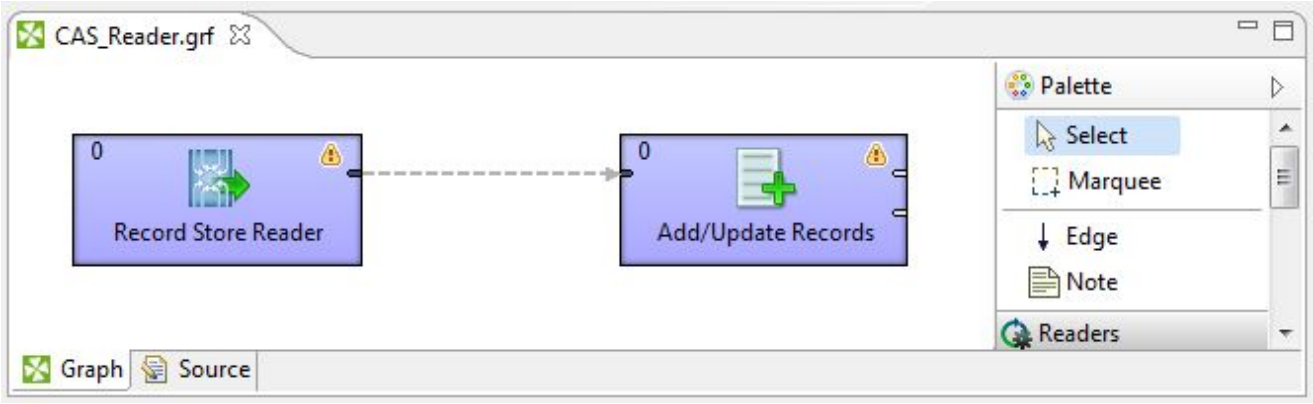
This topic describes how to create a graph that reads records from a CAS Record Store.

This sample graph uses the **Add/Update Records** connector to load the records into an Endeca data store. However, you can use any other Information Discovery ingest connector instead.

To create a project and a corresponding **Record Store Reader** graph:

1. Create an empty project with the **File>New>ETL Project** command.
   In our example, **CAS_Reader** is the name of the project.
2. In the project, create an empty graph with the **File>New>ETL Graph** command.
3. In the Palette pane, drag these components into the Graph Editor:
   (a) Drag the **Record Store Reader** component from the **Discovery** section.
   (b) Drag the **Add/Update Records** connector from the **Discovery** section.
4. In the Palette pane, click **Edge** and use it to connect the components.
5. From the File menu, click **Save** to save the graph.

At this point, the graph should look like this:



Configuring the Record Store Reader component

This topic describes how to configure the **Record Store Reader** component.

This procedure assumes that you have created a graph and added the **Record Store Reader** component.

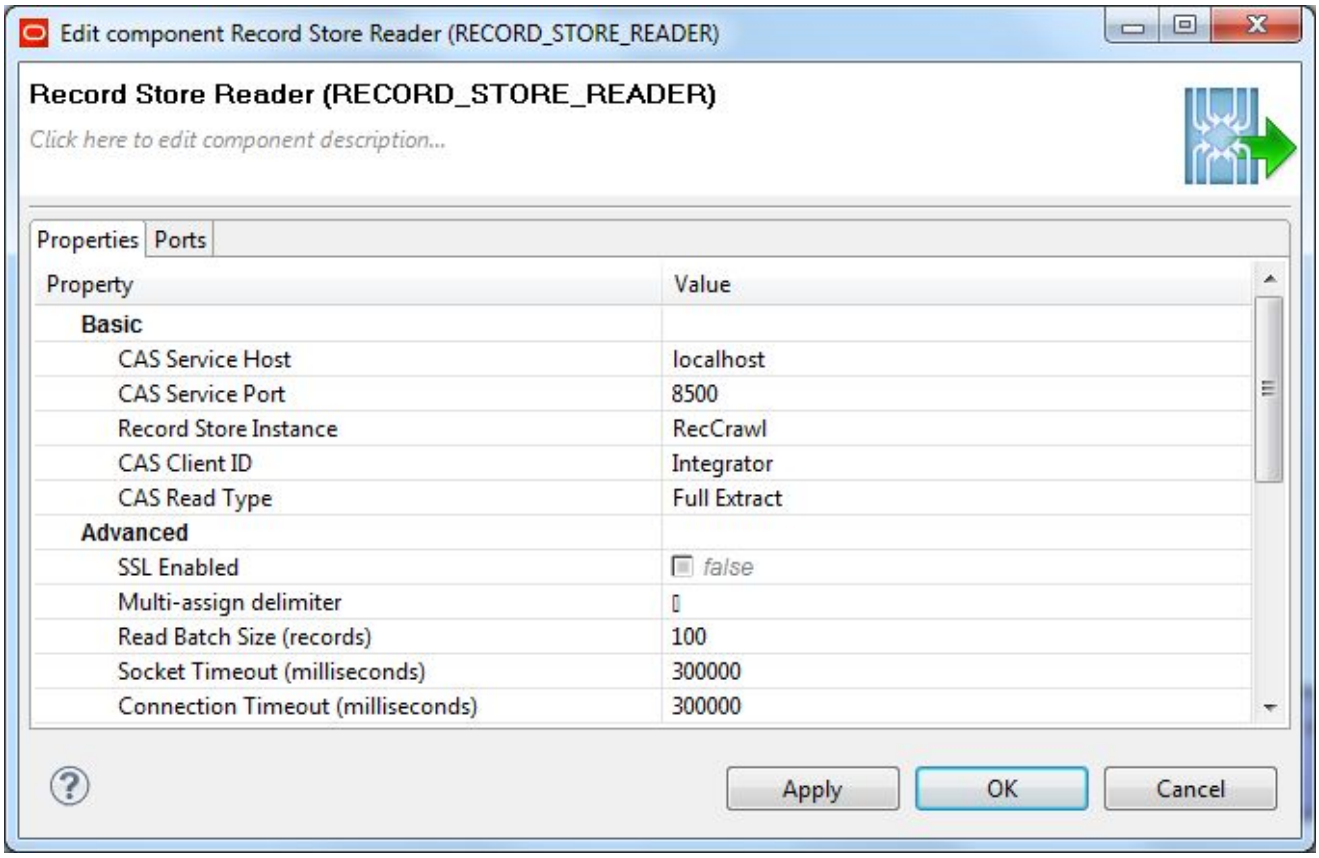
To configure the **Record Store Reader** component:

1. In the Graph window, double-click the **Record Store Reader** component.
   The Edit Component dialog is displayed.
2. In the Edit Component dialog, enter these mandatory settings in the Basic section:
   (a) **CAS Service Host**: The name of the machine on which the Endeca CAS Service is running. The default name is `localhost`.
   (b) **CAS Service Port**: The port on which the CAS Service is listening. The default is `8500`.
   (c) **CAS Record Store Instance**: The name of the unique Record Store instance from which records will be read.
   (d) **CAS Client ID**: An arbitrary name that identifies this client to the Record Store. The default is `Integrator` but you can overwrite it with a client ID of your choosing. The Record Store uses this client ID to keep track of which generations have been read by this client. If multiple clients are accessing the Record Store, each client should have a unique name.
   (e) **CAS Read Type**: Use the drop-down menu to select the type of read operation: `Full Extract` is the default and retrieves baseline records (i.e., all the records from the last-committed generation in the Record Store), while `Incremental` retrieves delta records (i.e., the delta between two or more generations in the Record Store).
3. Still in the Edit Component dialog, you can enter these optional settings in the Advanced section or accept the defaults:
   (a) **SSL Enabled**: Toggle this field to `true` if the Endeca CAS Service is SSL-enabled.
   (b) **Multi-assign delimiter**: The character that separates multi-assign values in an input property. The default is the Unicode DELETE character (\U007F).
   (c) **Read Batch Size**: The number of records in a fetched batch. The default is `100` records per batch.
   (d) **Socket Timeout**: The timeout in milliseconds for waiting for data from the Record Store (i.e., the maximum period inactivity between two consecutive data packets). The default is `300000` milliseconds (5 minutes). A timeout value of zero is interpreted as an infinite timeout.

(e) **Connection Timeout**: The timeout in milliseconds until a connection is established with the Endeca CAS Service. The default is `300000` milliseconds (5 minutes). A timeout value of zero is interpreted as an infinite timeout.

4. Optionally, you can use the **Component name** field to provide a customized name for this component.
5. When you have input all your changes, click **OK**.
6. Save the graph.

After the component is configured, the Edit Component dialog should look similar to this example:



Generating Edge metadata with the Record Store Wizard

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

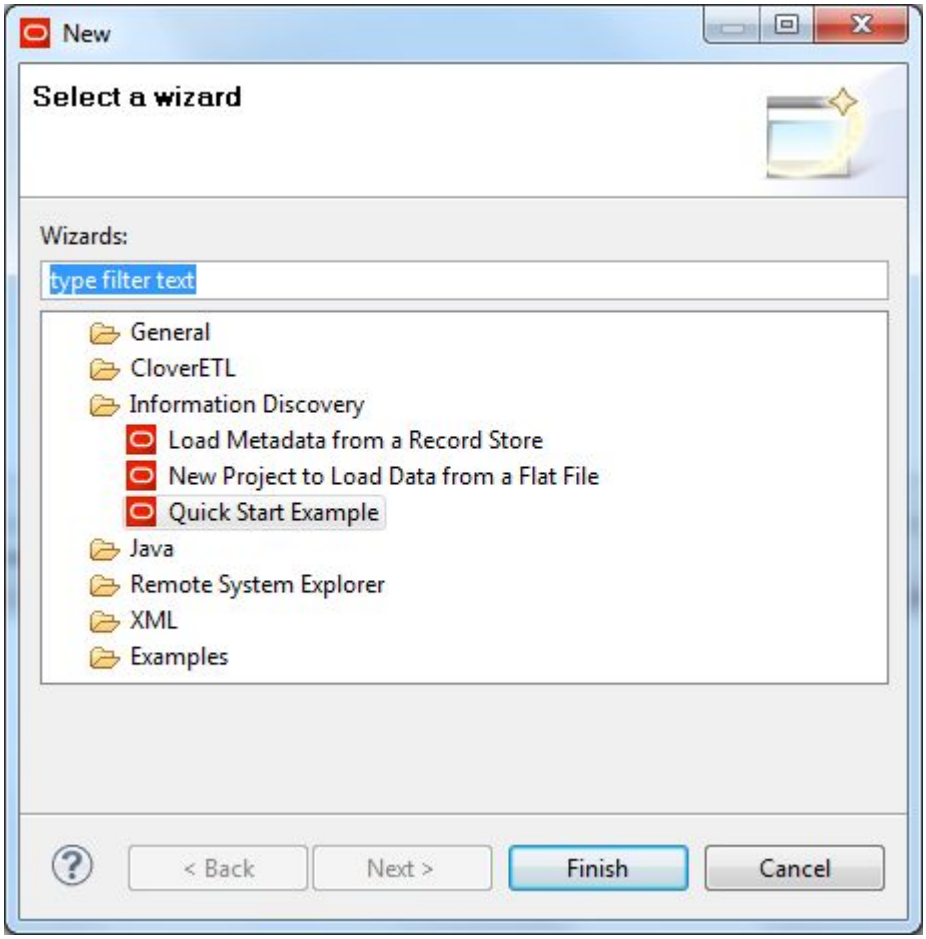
The pre-requisites for this procedure are:

1. You must have created a Record Store instance.
2. The Record Store instance must have at least one committed generation of records. That is, you must have run a full (baseline) crawl with the output sent to the Record Store.
3. The Endeca CAS Service must be running so that the Record Store Wizard can connect to it.

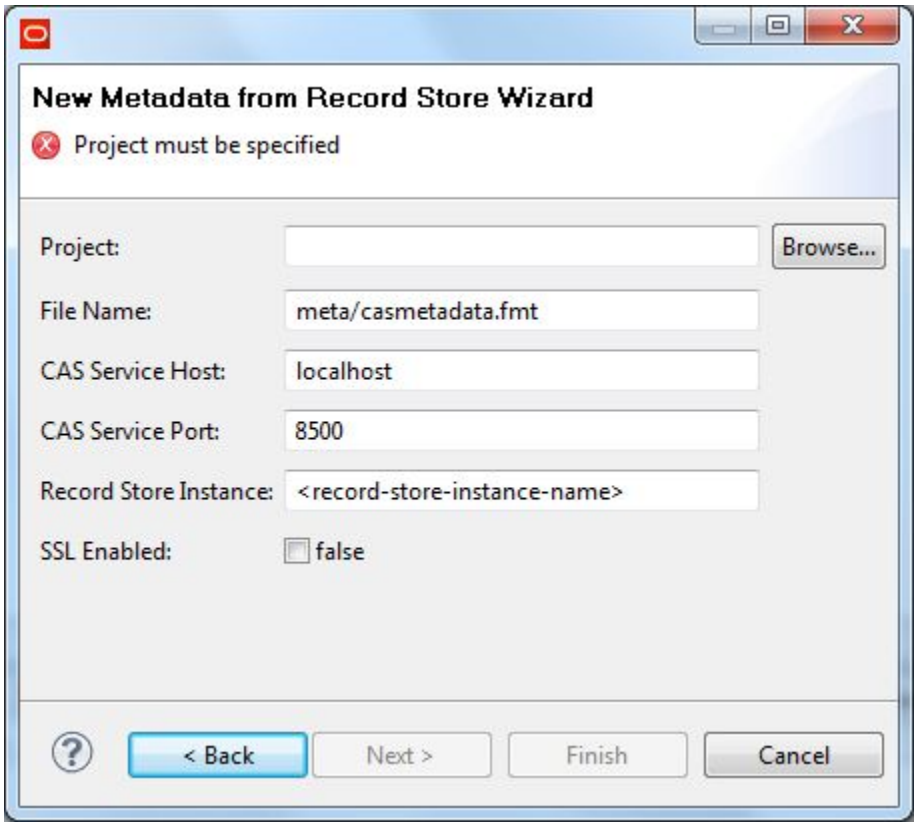
4. You must have created an Integrator project and graph for the **Record Store Reader** component, as you will store the metadata file in this project.

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

1. From your Record Store Reader project, select **File>New>Other**.
   The Select a Wizard menu is displayed:

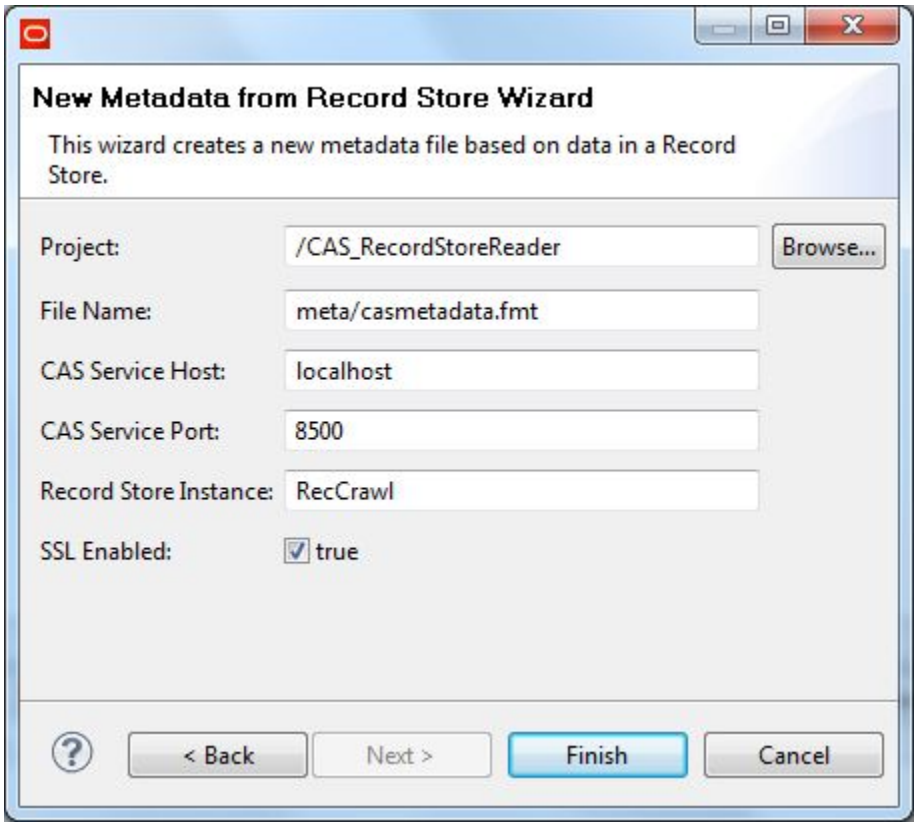


2. In the Select a Wizard menu, select **Load Metadata from a Record Store** and then click **Next**.
   The Record Store Wizard is displayed.

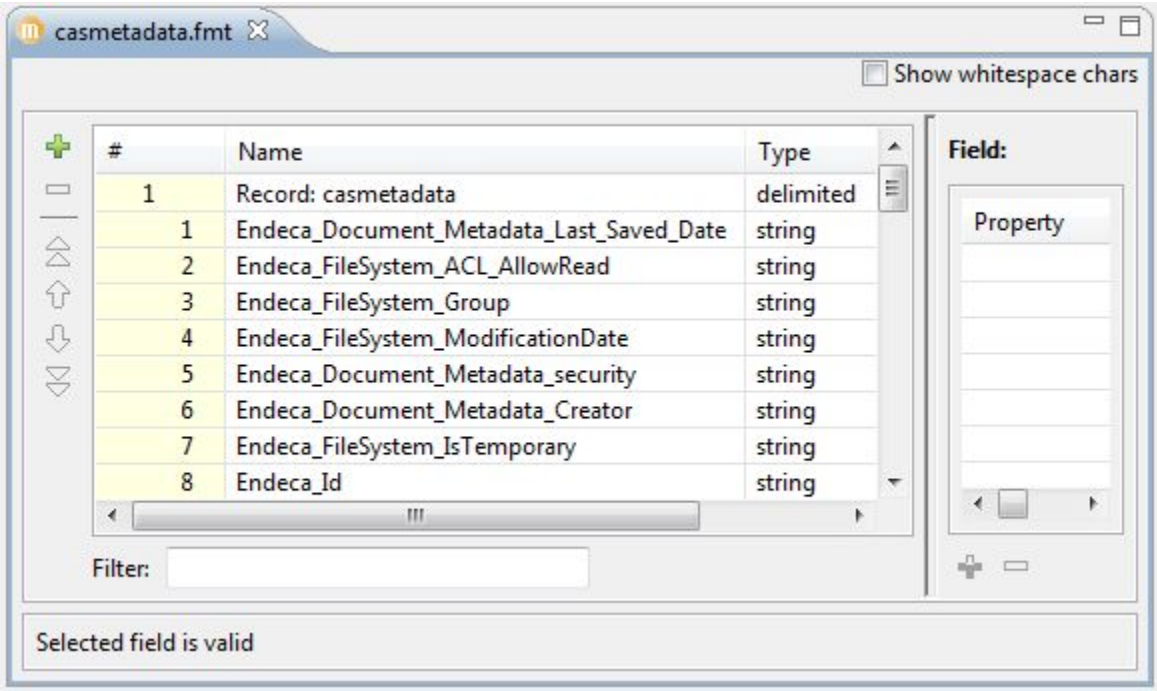


3. In the Record Store Wizard, enter these values:
   (a) **Project**: Click **Browse** and select your project from the Folder Selection dialog.
   (b) **File Name**: Enter the pathname (project folder and file name) for the resulting metadata file. You can use the default name.
   (c) **CAS Service Host**: Enter the name of the machine on which the Endeca CAS Service is running. This name should be the same as the name in the **CAS Service Host** configuration property of the **Record Store Reader** component.
   (d) **CAS Service Port**: Enter the port of the Endeca CAS Service. This port should be the same as the one in the **CAS Service Port** configuration property of the **Record Store Reader** component.
   (e) **Record Store Instance**: Enter the name of the Record Store instance that you created. This name should be the same as the one in the **Record Store Instance** configuration property of the **Record Store Reader** component.
   (f) **SSL Enabled**: Toggle this field to `true` only if the Endeca CAS Service is SSL enabled.

   After filling in the field values, the Record Store Wizard should look similar to this example:



4. When you have input your values in the Record Store Wizard, click **Finish**.
The wizard retrieves the record properties from the Record Store instance and displays them in the Integrator Metadata editor, as shown in this truncated example:



In addition, the external metadata file is stored in the folder you specified in the **File Name** field.

The next step is to assign this metadata to the Edge component in the graph.

Configuring the Edge for the Record Store Reader component

This task describes how to set the external metadata file for the Edge.

To configure the **Record Store Reader** component's Edge:

1. Right-click on the Edge and select **New metadata>Link shared definition**.
   The URL Dialog is displayed.
2. In the URL Dialog:
   (a) Double-click the folder in which you placed the metadata file, such as the `meta` folder.
   (b) Select the metadata file you generated with the Record Store Wizard and click **OK**.
3. Save the graph.

Configuring the Add/Update Records connector

This task describes how to configure the **Add/Update Records** connector in the graph.

To configure the connector, use the procedure described in the topic *Configuring the Add/Update Records connector on page 50*.

In our sample graph, we are using the **Endeca_Id** attribute as the value for the component's **Spec Attribute** configuration property.

After creating the graph, you can run the graph to retrieve the records from the Record Store instance. Before running the graph, make sure that both the Endeca CAS Service and the Endeca data store are running. You can run the graph by clicking the green circle with white triangle icon in the Tool bar: 

Running the Record Store Reader graph

After creating the graph and configuring the components, you can run the graph to add the records to the Endeca data store.

To run the graph to load records from a Record Store instance:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **Add Managed Values** connector and that the Endeca data store has been started. Also make sure that the Endeca CAS Service is running.
2. Run the transaction 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 output includes information about the Record Store instance and what type of read operation will be done:

```
[RECORD_STORE_READER0_0] - RecordStoreReader: Transaction ID = 10
```

```
[RECORD_STORE_READER0_0] - RecordStoreReader: Last Committed Generation ID = 1
[RECORD_STORE_READER0_0] - Full Extract or Baseline read detected
```

The output also lists the number of records that were read in by the **Record Store Reader** and the number of records that were sent to the Endeca data store by the ingest connector.

Chapter 16

Importing and Exporting the Configuration

This chapter discusses how to import and export the configuration and schema of a running Endeca data store.

About importing and exporting

Exporting the configuration

Importing the configuration

Running the configuration graphs with a transaction graph

About importing and exporting

Use the **Export Config** and **Import Config** connectors to export and import the schema and configuration.

The configuration of your index and the schema for your records are created once you initially load the data, the schema, and the configuration into the Endeca data store.

Use cases for exporting and importing schema and configuration

You may need to import and export your schema and configuration in several typical scenarios:

- As part of the baseline update process, when only updates to the data are required but the Dgraph configuration and the schema must remain the same.
- If you have an implementation running in the development environment, it typically should match the implementation running in the production environment in terms of index configuration and the schema for your records. (Although the development application may contain a subset of data). You can use export and import connectors for sharing the configuration and schema between these environments.

What is being exported and imported

The following aspects of your configuration and schema are being exported and imported when you use the **Export Config** or **Import Config** connectors:

- The schema for your records. The schema is represented by PDRs and DDRs that describe the behavior of attributes on your records, such as whether they are searchable, or have hierarchy.
- The configuration, which includes:
  - The indexed configuration — the XML configuration documents, such as documents describing your record search configuration and search interfaces, or thesaurus configuration.
  - All additional configuration information, such as display names or attribute groups, the configuration captured in the GCR, and precedence rules.

Note: The export and import connectors do not export or import the file that stores all word forms used for stemming dictionaries in the Dgraph. This file is created automatically when the Endeca data store is provisioned, and is typically not modified. In cases when you want to replace the file with another language version, use the procedure described in the chapter *Loading Stemming Files on page 96*.

Exporting the configuration

You can export the configuration from an Endeca data store by using the **Export Config** connector.

Adding components to the export graph

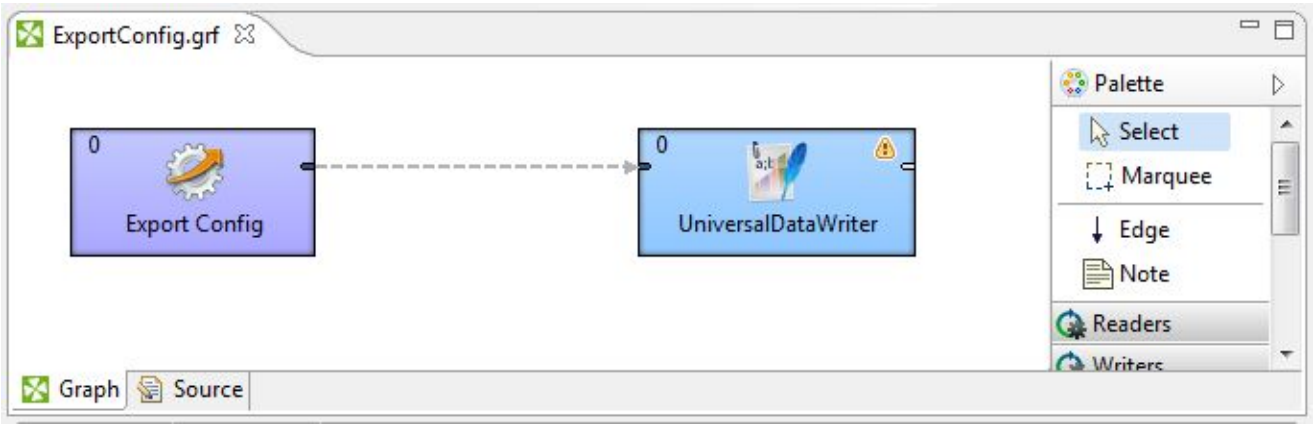
This topic describes the Integrator components that must be added to the export graph.

This procedure assumes that you have created an empty graph (our example is named ExportConfig).

To add components to the graph that exports the configuration from a Dgraph:

1. In the Palette pane, drag the **Export Config** component from the **Discovery** section.
2. In the Palette pane, drag the **UniversalDataWriter** component from the **Writers** section.
3. In the Palette pane, click **Edge** and use it to connect the components.
4. Save the graph.

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

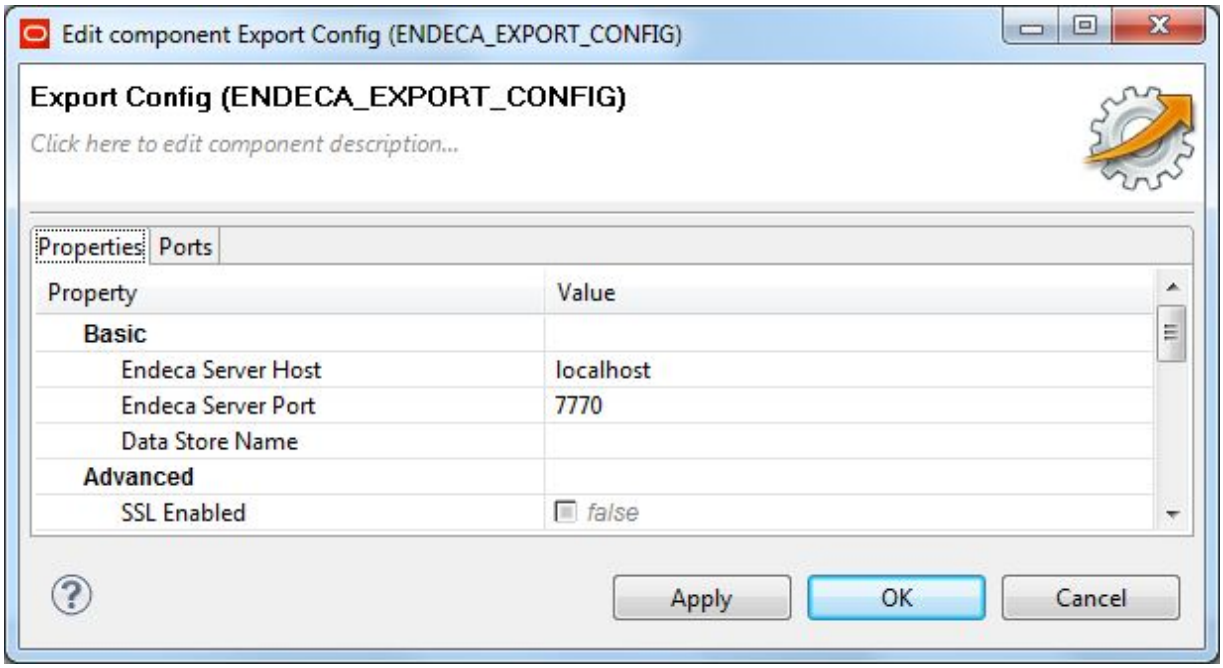


Configuring the Export Config connector

You must configure the **Export Config** connector with the name of the Endeca data store.

To configure the **Export Config** connector:

1. In the Graph window, double-click the **Export Config** component.
   The Edit Component dialog is displayed.



2. In the Writer Edit Component dialog, enter these mandatory settings in the Basic section:
   (a) **Endeca Server Host**: Enter the host name of the machine on which the Endeca Server is running.
   (b) **Endeca Server Port**: Enter the port on which the Endeca Server Engine is listening for requests.
   (c) **Data Store Name**: Enter the name of the Endeca data store from which the configuration will be exported.
3. Still in the Writer Edit Component dialog, you should toggle the **SSL Enabled** field to `true` if the Endeca Server is SSL-enabled.
4. When you have input all your changes, click **OK**.
5. Save the graph.

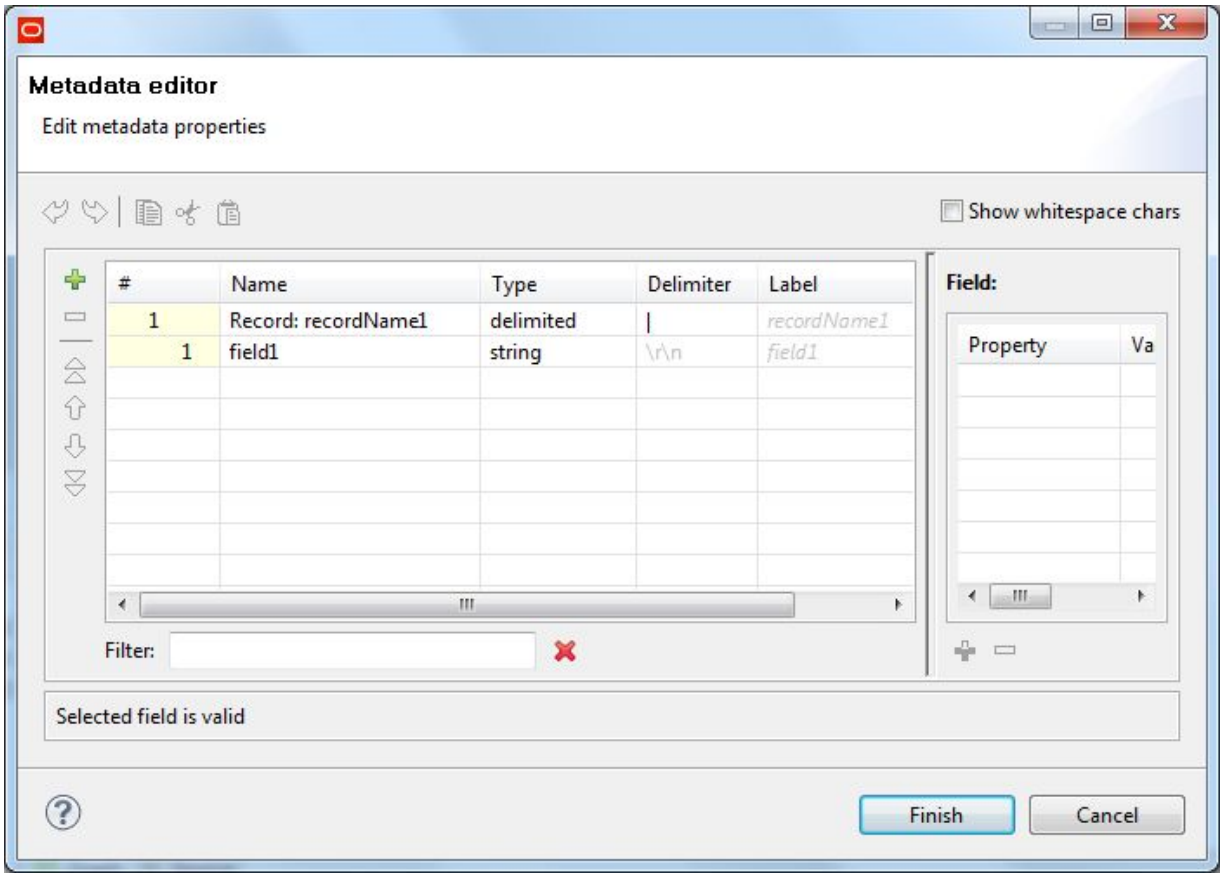
Configuring the Edge in the export graph

This topic describes how to configure the metadata for the Export Config Edge.

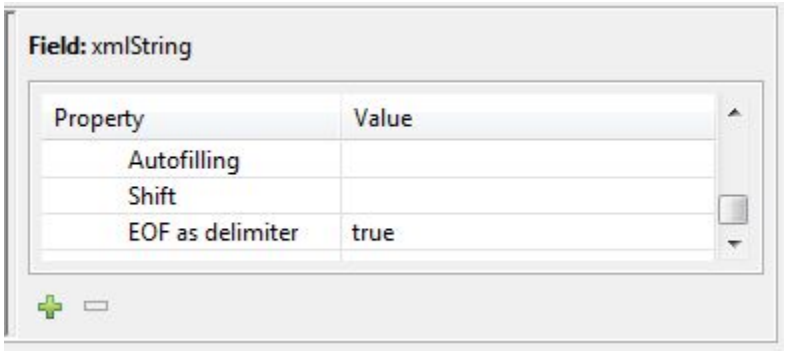
The metadata must be configured to have only one string field and no record delimiter. Therefore, the metadata of the Edge must be manually modified to remove the record and field delimiters from the metadata. This will leave the **EOF as delimiter** property as the sole delimiter.

To configure the Edge Metadata definition for exporting the configuration from an Endeca data store:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a more descriptive name (such as Export).
   (b) Leave the **Type** field as `delimited`.
   (c) Leave the **Delimiter** field as-is for now. (You will delete it in Step 5.)
3. In the Record pane, make these changes to the **field1** property:
   (a) Change the **field1** default name to **xmlString**.
   (b) Leave the **Type** field set to **String**.
   (c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as in this example:



4. When you have input all your changes in the Metadata Editor, click **Finish**.

5. Now you must manually remove the record and field delimiters from the metadata:
   (a) In the Graph Editor, 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, as shown in this example:

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

   (c) Delete the **fieldDelimiter** and **recordDelimiter** attributes, so that the Record element now looks like this:

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

   (d) While still within the Source view, right-click and select **Save** to save the graph.
6. Click the **Graph** icon to return to the Graph Editor.

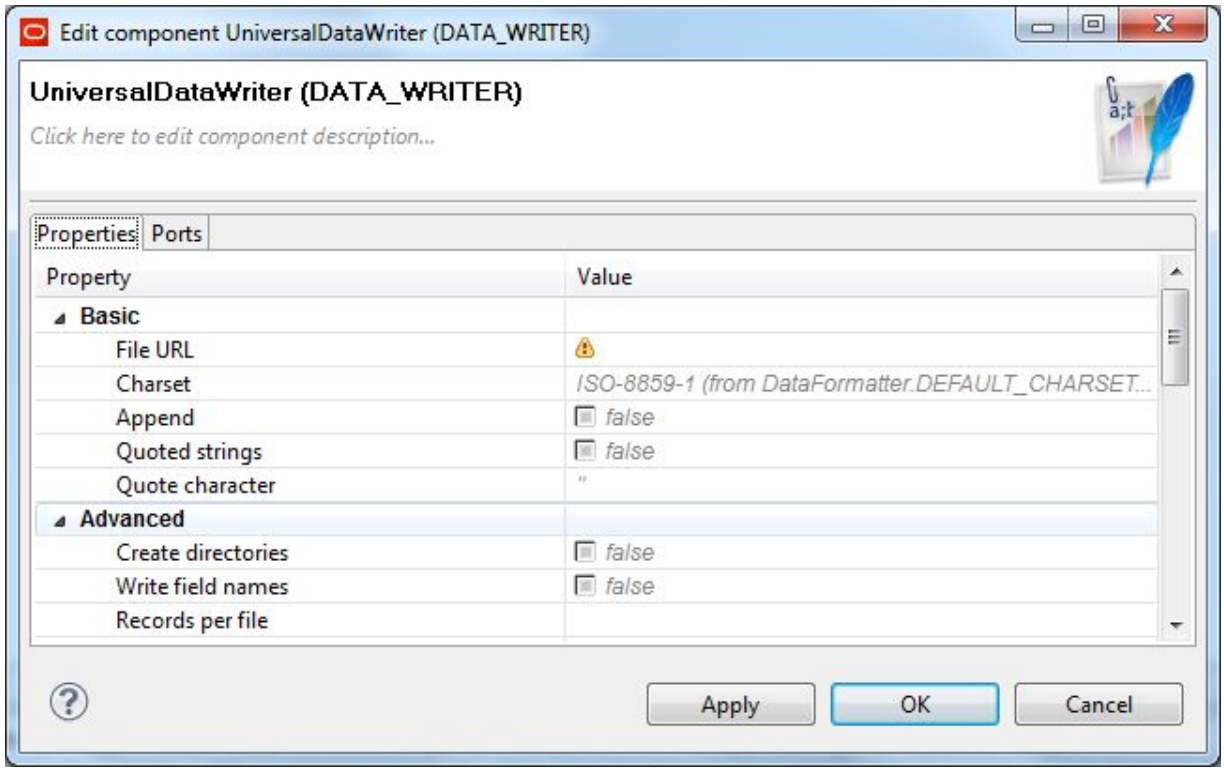
Configuring the UniversalDataWriter component

This task describes how to configure the **UniversalDataWriter** component to write out the configuration file.

You can configure the **UniversalDataWriter** to write out the exported configuration to file within the project or externally. This procedure assumes that you are writing the output to a text file named `config.out` which is located in the project's **data-out** folder.

To configure the **UniversalDataWriter** component to write out the configuration file:

1. In the Graph Editor, double-click the **UniversalDataWriter** component to bring up the Edit Component dialog.
   The Writer Edit Component dialog is displayed.



2. For the **File URL** property:
   (a) Click inside its Value field, which displays a **...** browse button.
   (b) Click the browse button, which brings up the **URL Dialog** screen.
3. In the **URL Dialog**:
   (a) Click the **Workspace view** tab and then double-click the **data-out** folder.
   (b) Select the `config.out` file and click **OK**.
4. Optionally, you can use the **Component name** field to provide a customized name for the component.
5. In the Writer Edit Component dialog, click **OK**.
6. Save the graph.

Instead of running this graph directly, you can create a transaction graph (with a **Transaction RunGraph** connector) with this ExportConfig graph as its child graph, and then run the transaction graph. Transaction graphs are described in the topic *Working with Outer Transaction Graphs on page 22*.

Importing the configuration

You can import the configuration to the Endeca data store using the **Import Config** component.

Adding components to the import graph

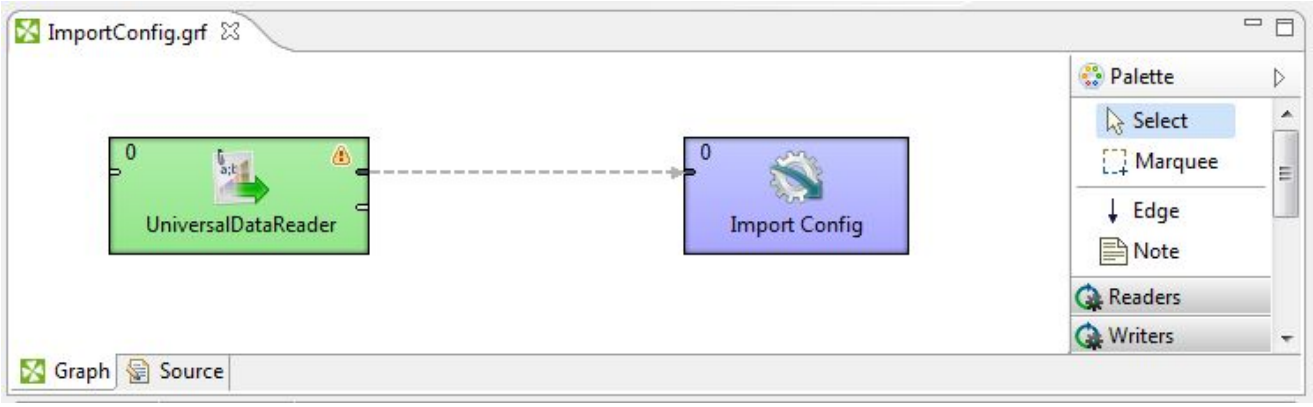
This topic describes the Integrator components that must be added to the import graph.

This procedure assumes that you have created an empty graph (our example is named ImportConfig).

To add components to the graph that imports the configuration into a running Endeca data store:

1. In the Palette pane, drag the **UniversalDataReader** component from the **Readers** section.
2. In the Palette pane, drag the **Import Config** component from the **Discovery** section.
3. In the Palette pane, click **Edge** and use it to connect the components.
4. Save the graph.

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



Configuring the Reader in the import graph

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

This procedure assumes that the Endeca data store's configuration was written to a text file named `config.out` which is located in the project's **data-out** folder. This reader will use that file as its input file.

To configure the Reader component in the import graph:

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, which brings up the **URL Dialog** screen.
3. In the **URL Dialog**:
   (a) Click the **Workspace view** tab and then double-click the **data-out** folder.
   (b) Select the `config.out` file and click **OK**.
4. In the Reader Edit Component dialog, click **OK**.
5. Save the graph.

Configuring the Edge in the import graph

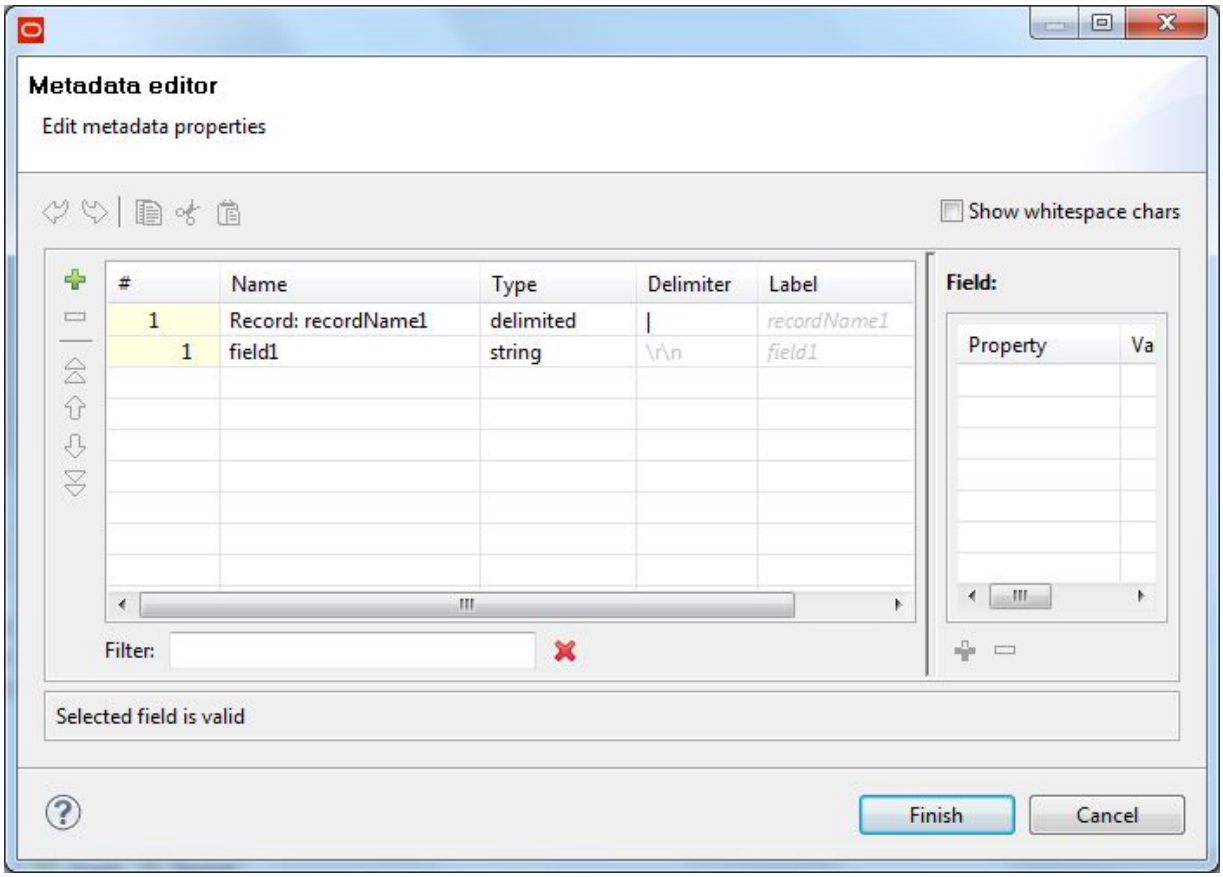
This topic describes how to configure the metadata for the Reader Edge.

The configuration of the Edge in the import graph is similar to that of the export graph. That is, the Edge metadata must be configured to have only one string field and no record delimiter. Therefore, the metadata of

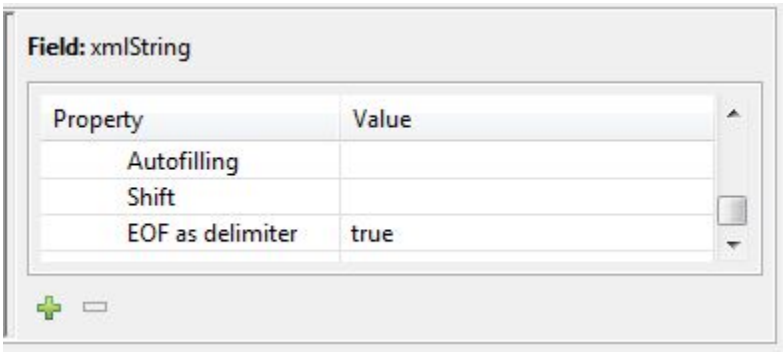
the Edge must be manually modified to remove the record and field delimiters from the metadata. This will leave the **EOF as delimiter** property as the sole delimiter.

To configure the Edge Metadata definition for importing the configuration from a disk file to an Endeca data store:

1. Right-click on the Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the **Record:recordName1** field:
   (a) Change the **recordName1** default value to a more descriptive name (such as Import).
   (b) Leave the **Type** field as `delimited`.
   (c) Leave the **Delimiter** field as-is for now. (You will delete it in Step 5.)
3. In the Record pane, make these changes to the **field1** property:
   (a) Change the **field1** default name to **xmlString**.
   (b) Leave the **Type** field set to **String**.
   (c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as in this example:



4. When you have input all your changes in the Metadata Editor, click **Finish**.
5. Now you must manually remove the record and field delimiters from the metadata:
   (a) In the Graph Editor, 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, as shown in this example:

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

   (c) Delete the **fieldDelimiter** and **recordDelimiter** attributes, so that the Record element now looks like this:

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

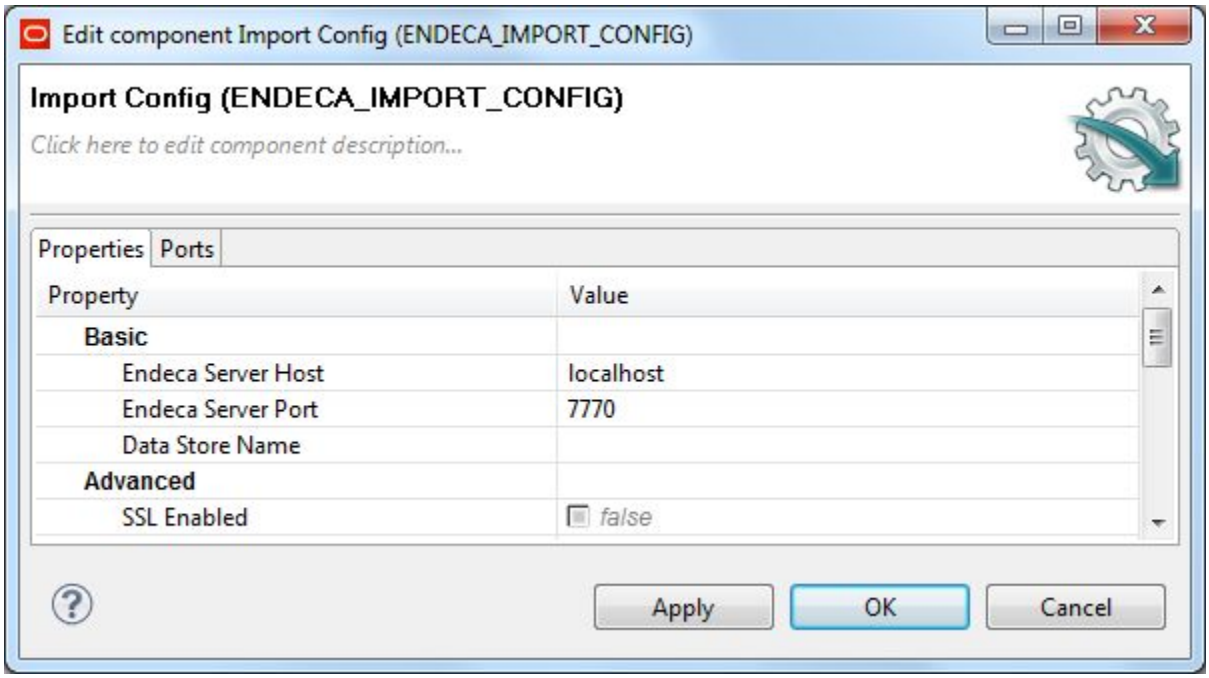
   (d) While still within the Source view, right-click and select **Save** to save the graph.
6. Click the **Graph** icon to return to the Graph Editor.

Configuring the Import Config connector

You must configure the **Import Config** connector with the name of the Endeca data store.

To configure the **Import Config** connector:

1. In the Graph window, double-click the **Import Config** component.
   The Edit Component dialog is displayed.



2. In the Edit Component dialog, enter these mandatory settings in the Basic section:
   (a) **Endeca Server Host**: Enter the host name of the machine on which the Endeca Server is running.
   (b) **Endeca Server Port**: Enter the port on which the Endeca Server is listening for requests.
   (c) **Data Store Name**: Enter the name of the Endeca data store to which the configuration will be imported.
3. Still in the Writer Edit Component dialog, you should toggle the **SSL Enabled** field to `true` if the Endeca Server is SSL-enabled.
4. When you have input all your changes, click **OK**.
5. Save the graph.

Instead of running this graph directly, it is recommended that you create a transaction graph (with a **Transaction RunGraph** connector) with this ImportConfig graph as its child graph, and then run the transaction graph.

Running the configuration graphs with a transaction graph

You should run the export and import configuration graphs with a transaction graph.

A transaction graph uses a **Transaction RunGraph** connector to safely run one or more graphs within the transaction environment of an Endeca data store. This connector can start an outer transaction, run the set of graphs so that they succeed or fail as a unit, and finally commit the transaction (or roll it back upon failure). It is therefore recommended that instead of running each of the export and import graphs in standalone mode, you instead build one transaction graph that runs both configuration graphs.

In addition, the transaction graph should run a **Reset Data Store** graph after the Endeca data store configuration is exported, so that the data store is reset to the empty state. Then all three graphs should be run together in a single transaction.

The steps for setting up a transaction graph that runs the export/import graphs and the reset graph would be:

1. Create an export graph that uses the **Export Config** connector, to export the Endeca data store's configuration.
2. Create a graph (using the **Reset Data Store** connector) that resets the Endeca data store.
3. Create an import graph that uses the **Import Config** connector, to import the configuration into the Endeca data store.
4. Create a transaction graph that uses a **Transaction RunGraph** connector to run the above three graphs.
   The procedure to create a transaction graph is described in *Working with Outer Transaction Graphs on page 22*.
5. Run the transaction graph as you would run any other graph.

Chapter 17

Restoring View Definitions

This chapter describes how to load saved view definitions into an Endeca data store.

About restoring view definitions

Adding components to the views graph

Configuring the Reader for the views input file

Configuring both Edges in the views graph

Configuring the Reformat component for views

Configuring the views WebServiceClient component

About restoring view definitions

Integrator allows you to build a graph to reload your saved view definitions into an Endeca data store.

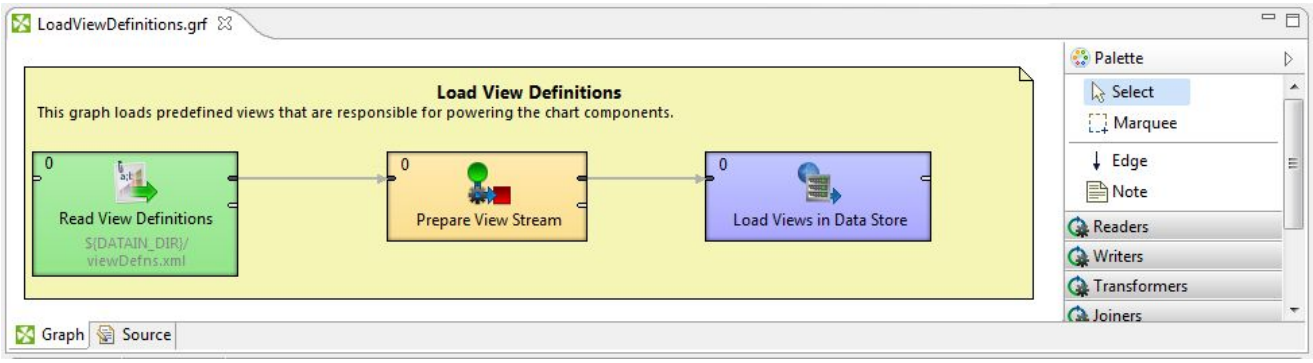
You use the **View Manager** component of Studio as your primary interface to create and manage views. That component provides a rich graphical user interface that simplifies the management of views.

However, there may be occasions when you want to use the **Reset Data Store** component to clear all data and configuration information (including view definitions) from your Endeca data store and then reload the data and configuration. In this case, after reloading your data records and configuration documents, you also want to restore your custom view definitions so that you do not have to manually re-create them with the **View Manager**.

This means that you must first retrieve the view definitions from the data store with the SConfig's `listEntities` operation and save them to a file. Note that this operation retrieves the Base view, as well as your custom views. Those custom views, minus the Base view, are then reloaded with the graph described in this chapter.

Example graph

The example graph used in this chapter from the Quick Start application. This graph is named LoadViewDefinitions and looks like this:



This graph uses three Integrator components that run in this order:

1. The **UniversalDataReader** component reads in the file containing the saved view definitions.
2. The **Reformat** component prepares the view definitions for ingest, mainly by removing the Base view.
3. The **WebServiceClient** component uses the SConfig's `putEntities` operation to load the view definitions into the Endeca data store.

The assumption of this scenario is that the Base view has already been created in the data store by the reloading of the attribute schema and the data records. The only thing missing is your custom views.

Adding components to the views graph

This topic describes the components that are needed in the views graph.

This procedure assumes that you have created an empty graph.

To add components to a graph for loading view definitions:

1. In the Palette pane, drag these components into the Graph Editor:
   (a) The **UniversalDataReader** component from the **Readers** section.
   (b) The **Reformat** component from the **Transformers** section.
   (c) The **WebServiceClient** component from the **Others** section.
2. In the Palette pane, click **Edge** and use it to connect the components.
3. From the File menu, click **Save** to save the graph.

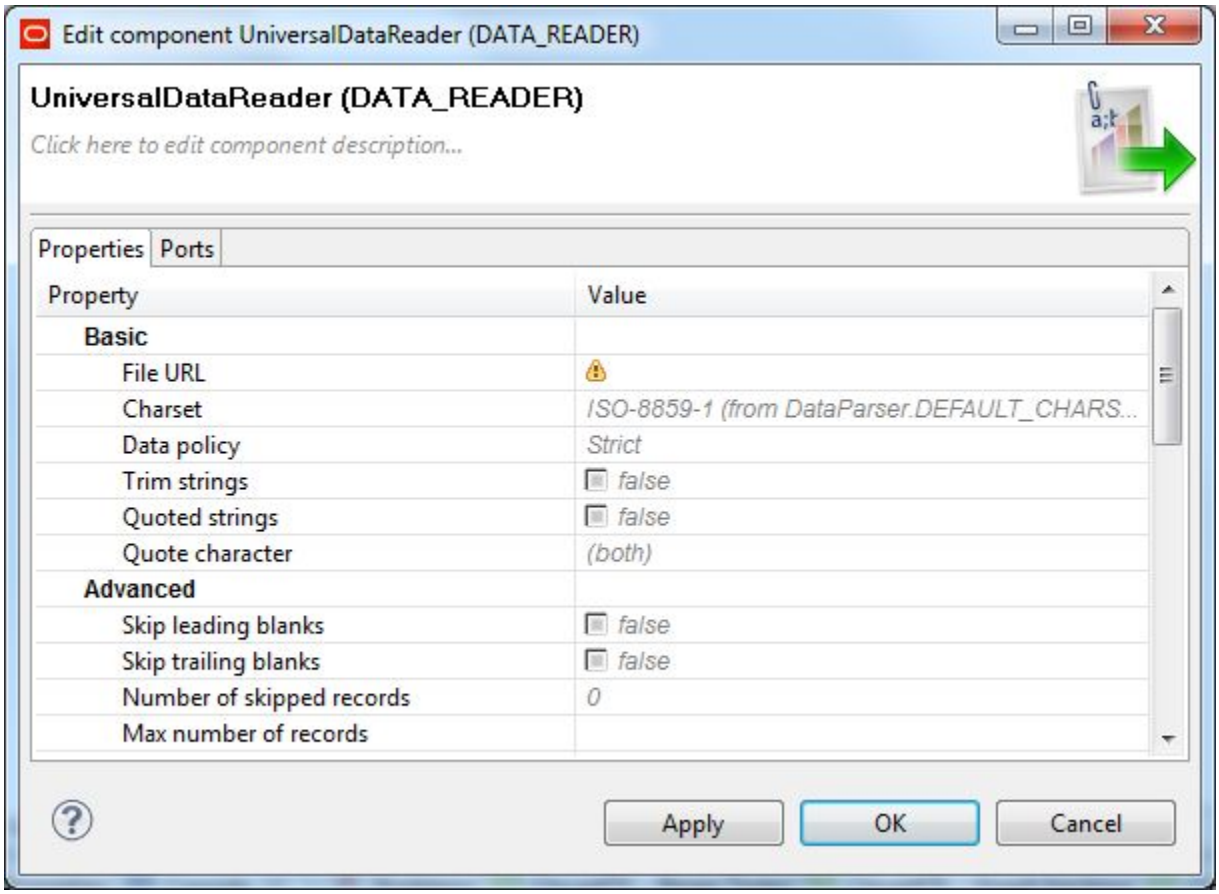
Configuring the Reader for the views input file

This task describes how to configure the reader component to read in the file that specifies the view definitions.

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

To configure the **UniversalDataReader** component for the views definition input file:

1. In the Graph Editor, double-click the **UniversalDataReader** component.
   The Edit Component dialog is displayed.



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** input file folder.
   (d) Select the views definition input file and click **OK**.
3. Click **OK** to apply your configuration changes to the component.
4. Save the graph.

Configuring both Edges in the views graph

This topic describes how to configure metadata for the two Edges.

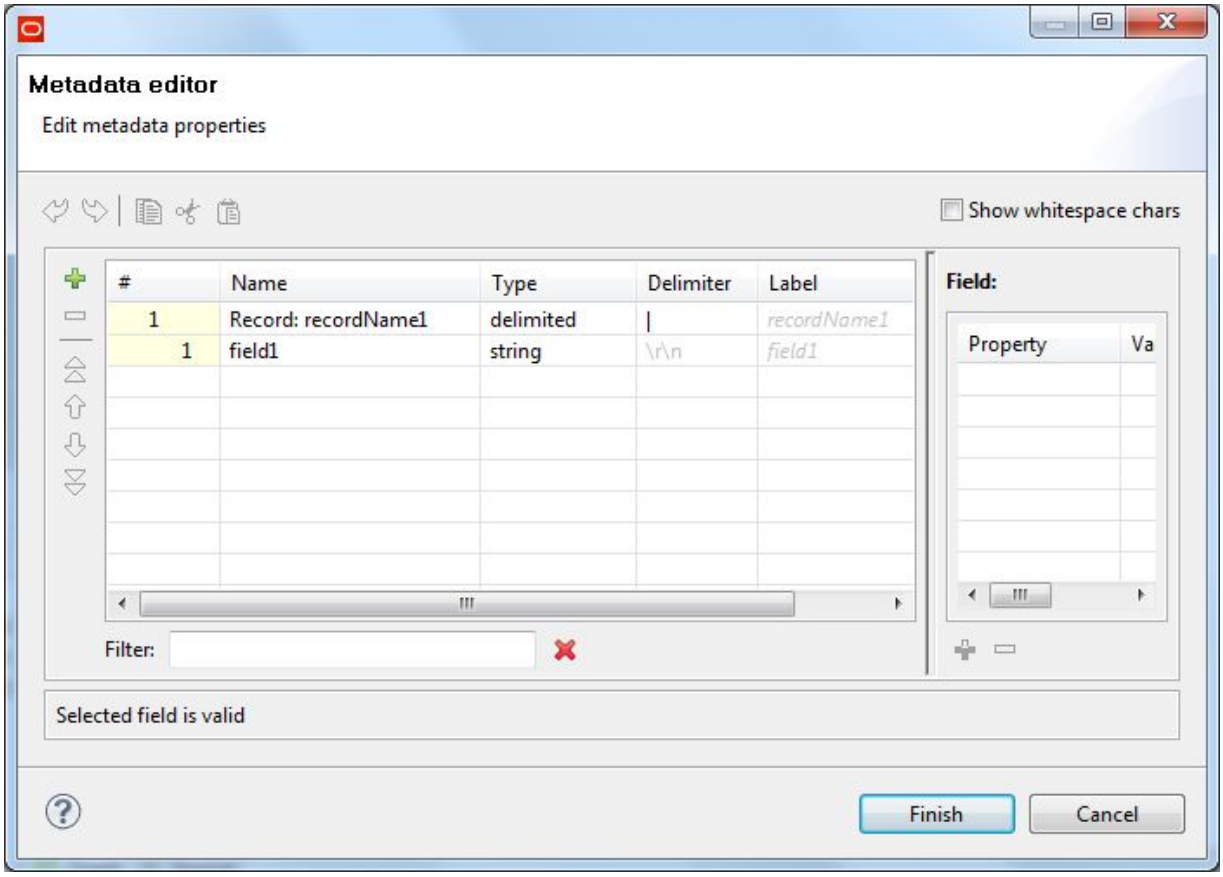
This procedure assumes that you have two Edges in the graph (one connecting the **UniversalDataReader** and **Reformat** components and the second connecting the **Reformat** and **WebServiceClient** components).

The procedure will configure the first Edge and then use its metadata for the second Edge.

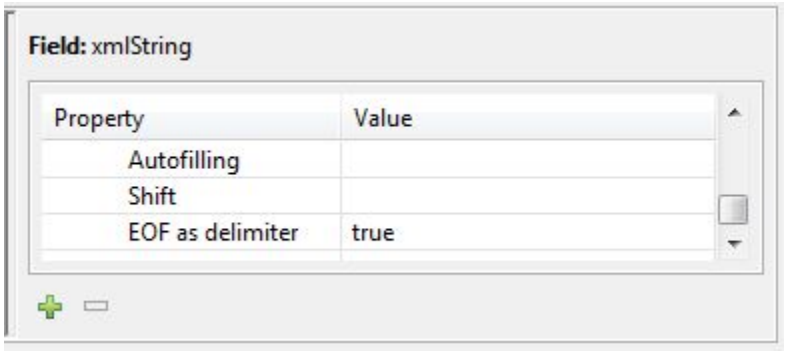
The contents of the views definition file will be read in as one long XML string. Therefore, the metadata must be configured to have only one string field and no record or field delimiters. This means that you must manually modify the Edge's metadata to remove the default record and field delimiters from the metadata. This will leave the **EOF as delimiter** property as the sole delimiter.

To configure the Edges for the graph:

1. Right-click on the first Edge and select **New metadata>User defined**.
   The Metadata editor is displayed with one default field.



2. In the Record pane, click the **Record** Name field and change its name to a more descriptive one (such as **viewXmlStream** in our example).
3. In the **field1** field:
   (a) Change the **field1** default name to **xmlString**.
   (b) Leave the **Type** field set to **String**.
   (c) In the Field Details pane, set the **EOF as delimiter** property to `true`, as in this example:



4. Now you must manually remove the record and field delimiters from the metadata:
   (a) In the Graph Editor, 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, as shown in this example:

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

   (c) Delete the **fieldDelimiter** and **recordDelimiter** attributes, so that the Record element now looks like this:

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

5. Click the **Graph** icon to return to the Graph Editor.
6. Right-click on the second Edge and choose **Select Metadata>viewXmlStream (id:metadata0)**.
   As the metadata is applied, the Edge changes from a dashed line to a solid line.
7. Save the graph.

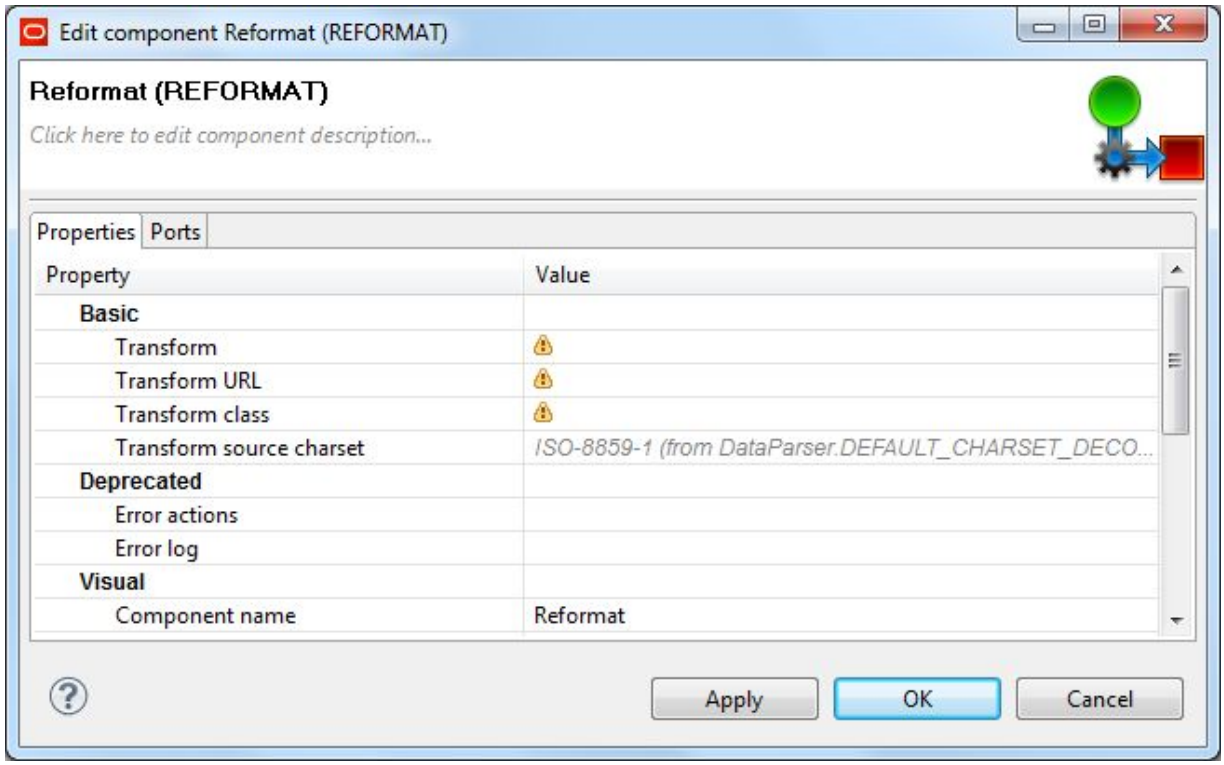
Configuring the Reformat component for views

The views definition data must be transformed by removing the Base view so that only your custom views are sent to the Endeca data store.

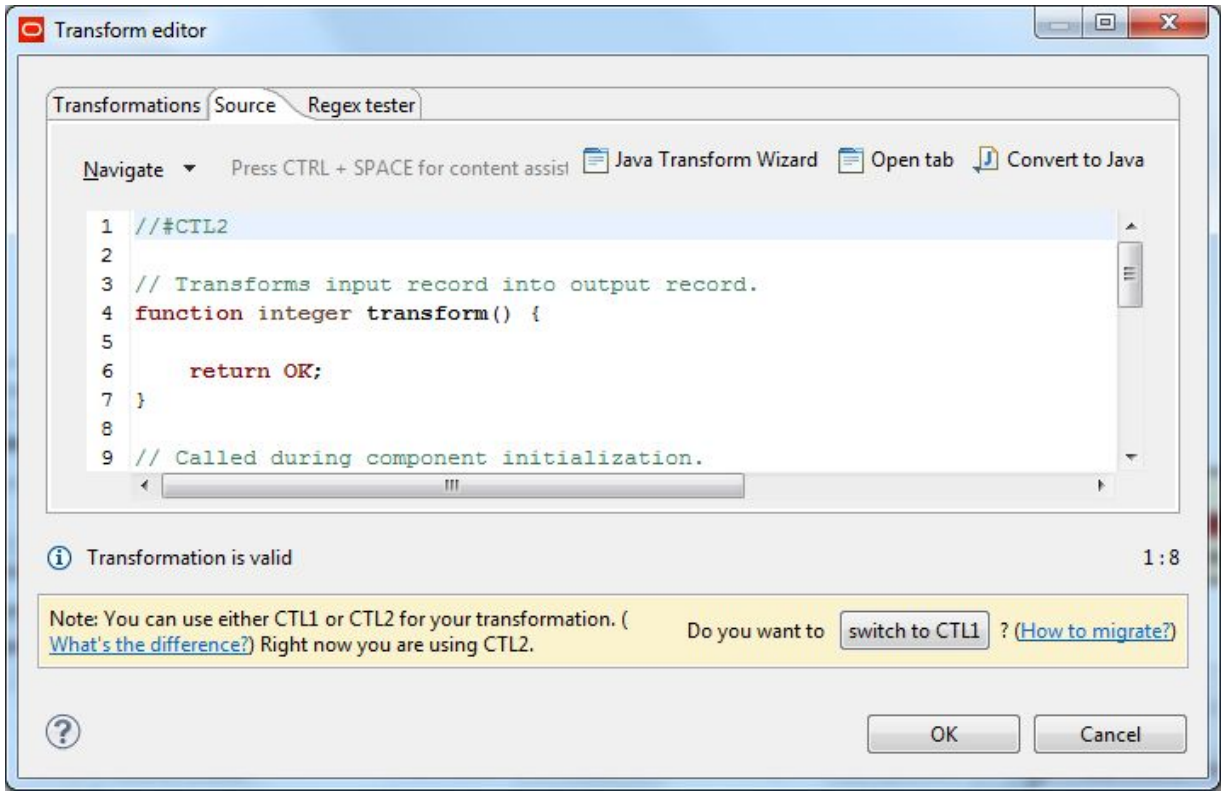
This procedure assumes that you have configured the metadata for both Edges.

To configure the **Reformat** component in the views graph:

1. In the Graph window, double-click the **Reformat** component.
   The Edit Component dialog is displayed.



2. Single-click in the **Transform** field and then click the **...** button.
The CTL template for the transform function is displayed.



3. Modify the CTL script so that it looks like this:

```
// Transforms input record into output record.
function integer transform() {
   string xmlString = substring($0.xmlString,
     indexOf($0.xmlString,"<semanticEntity"),indexOf($0.xmlString,"
     <semanticEntity key=\"Base\"") - indexOf($0.xmlString,
     "<semanticEntity")-1);
   xmlString = replace(xmlString, '[\n]', ' ');
   $out.0.xmlString = replace(xmlString,'[\s]+',' ');

   return ALL;
}
```

4. When you have finished your edits, click **OK**.
5. Optionally, you can use the **Component name** field to provide a customized name for this component.
6. Click **OK** to apply your configuration changes.
7. Save the graph.

When it runs, the component removes the Base view and prepares it to be sent in the `xmlString` property to the **WebServiceClient** component.

Configuring the views WebServiceClient component

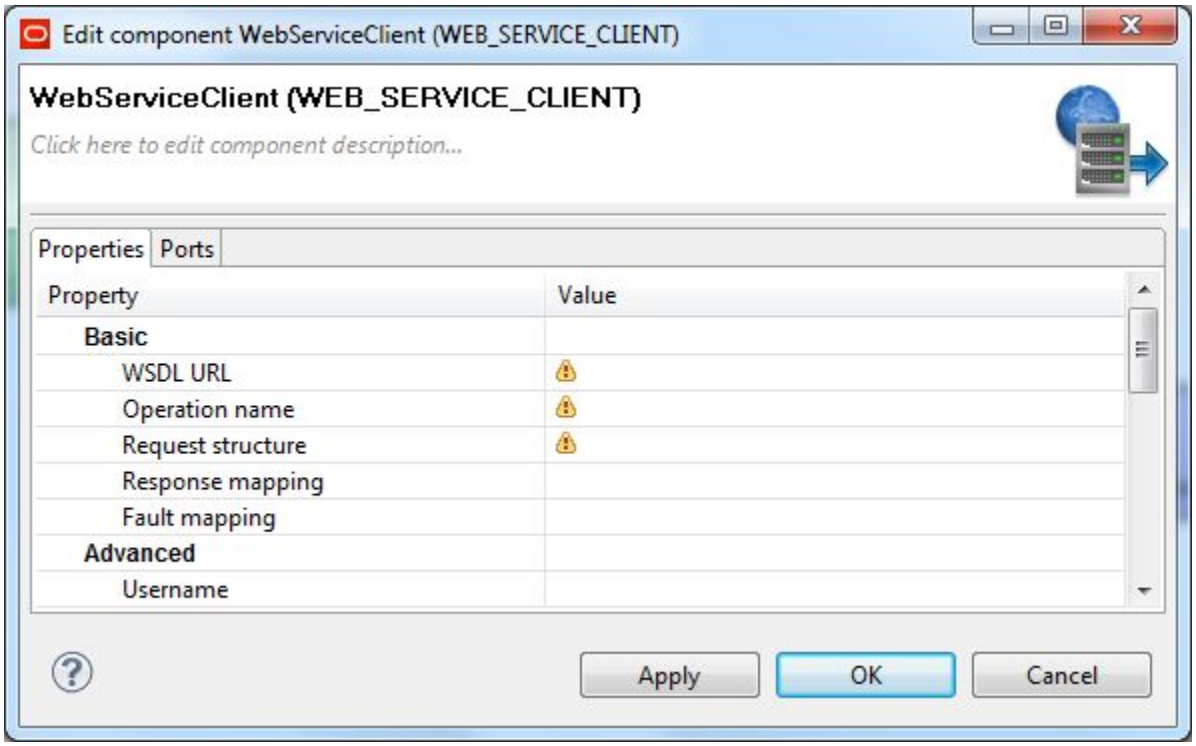
This topic describes how to configure the **WebServiceClient** component with the `putEntities` operation.

To configure the **WebServiceClient** component for loading view definitions:

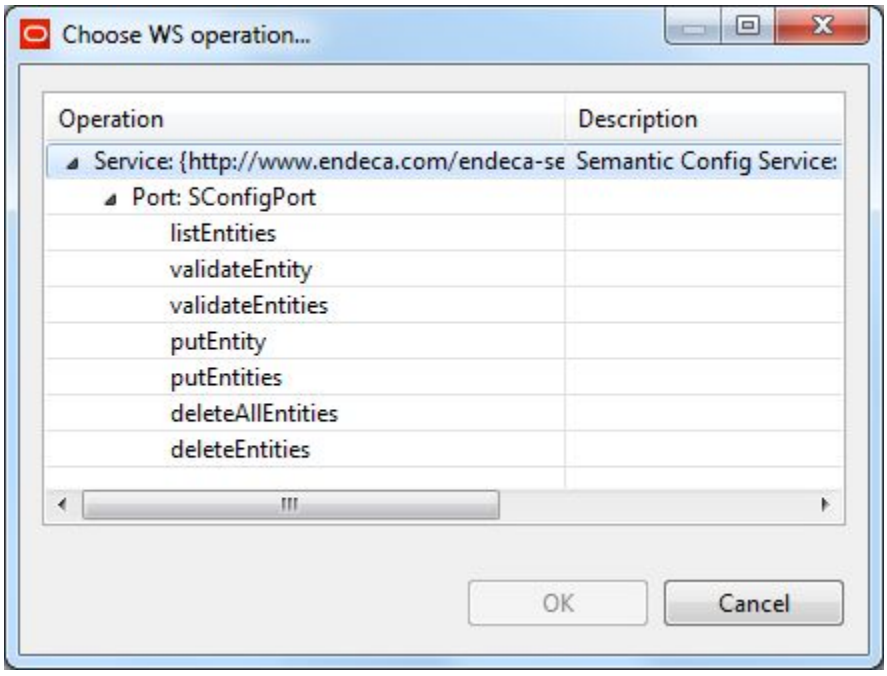
1. Make sure that both the Endeca Server and the specific Endeca data store are running and its Administration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

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

   The URL command returns the WSDL of the SConfig (Entity Configuration) Web service.
2. In the Graph editor, double-click the **WebServiceClient** component.
   The 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:



5. In the **Choose WS operation** dialog, select **putEntities** 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 an empty **Edit request structure** dialog.
7. Add this text to the **Generate request** field and then click **OK**.

```
<putEntities xmlns="http://www.endeca.com/endeca-server/sconfig/1/0">
$xmlString
</putEntities>
```

8. Optionally, you can use the **Component name** field to provide a customized display name for this component.
9. When you have entered all your changes in the Edit Component dialog, click **OK**.
10. Save the graph.

When you run the graph, only your custom view definitions are loaded into the Endeca data store.

Chapter 18

Deleting Data

This chapter describes how to delete records from an Endeca data store and how to removed key/value assignments from individual records.

Format of the delete input file

Adding components to the delete data graph

Configuring the Reader for the delete input file

Configuring the metadata for data deletes

Configuring the Delete Data connector

Running the delete data graph

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 an Endeca data store:

- Delete a full record from the data store (this implies deleting a specific record that represents your source data).
- 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.

Important: Do not use the **Delete Data** connector for deleting those records that define attributes, managed attributes, groups, or views. In particular, do not use this connector to delete PDR, DDR, or other schema records.

The two restrictions of this connector are:

- It cannot delete managed attributes from a taxonomy, but it can remove managed value assignments from a 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 or managed 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
ProductID|3000|Colors|green
ProductID|4000|Handling|
ProductID|5000||
```

When the connector is run with this input file:

- The assignment "green" from the Colors standard attribute is removed from Record 3000.
- All assignments from the Handling standard attribute are removed from Record 4000.
- Record 5000 is deleted from the Endeca data store.

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

Adding components to the delete data graph

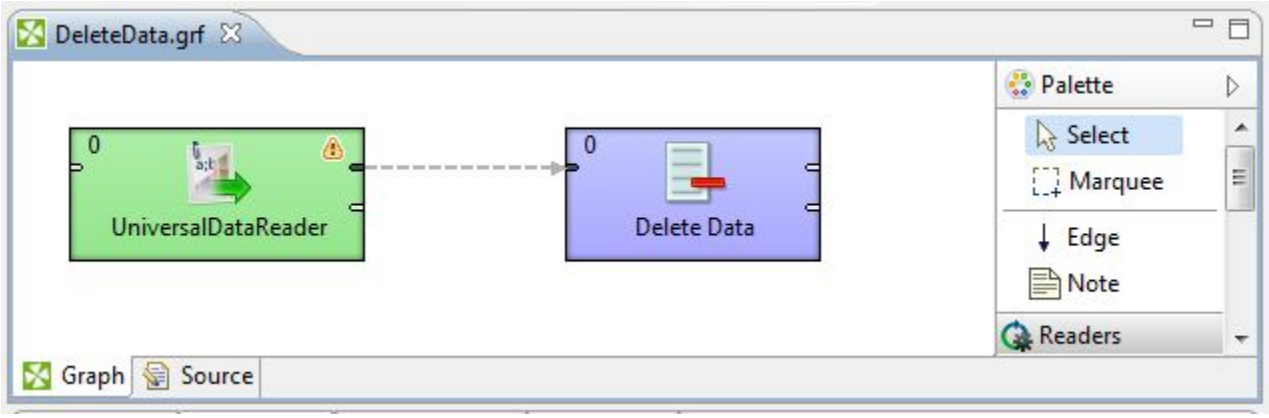
Building a graph to delete data requires that you add the **Delete Data** connector to the graph.

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 **Discovery** 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

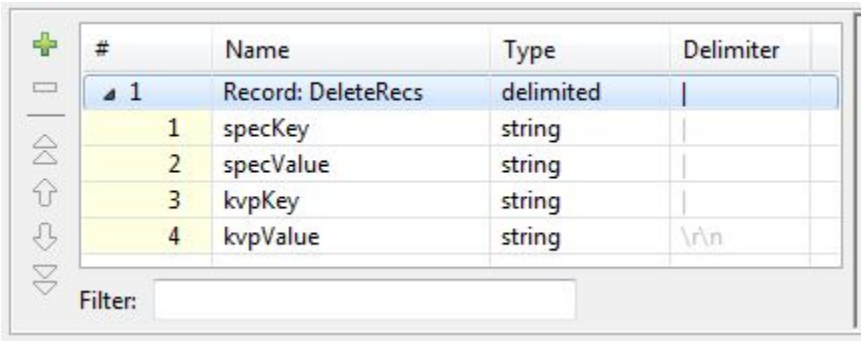
The Edge component must be configured with a Metadata definition.

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



   (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 for the Endeca data store that contains the records.

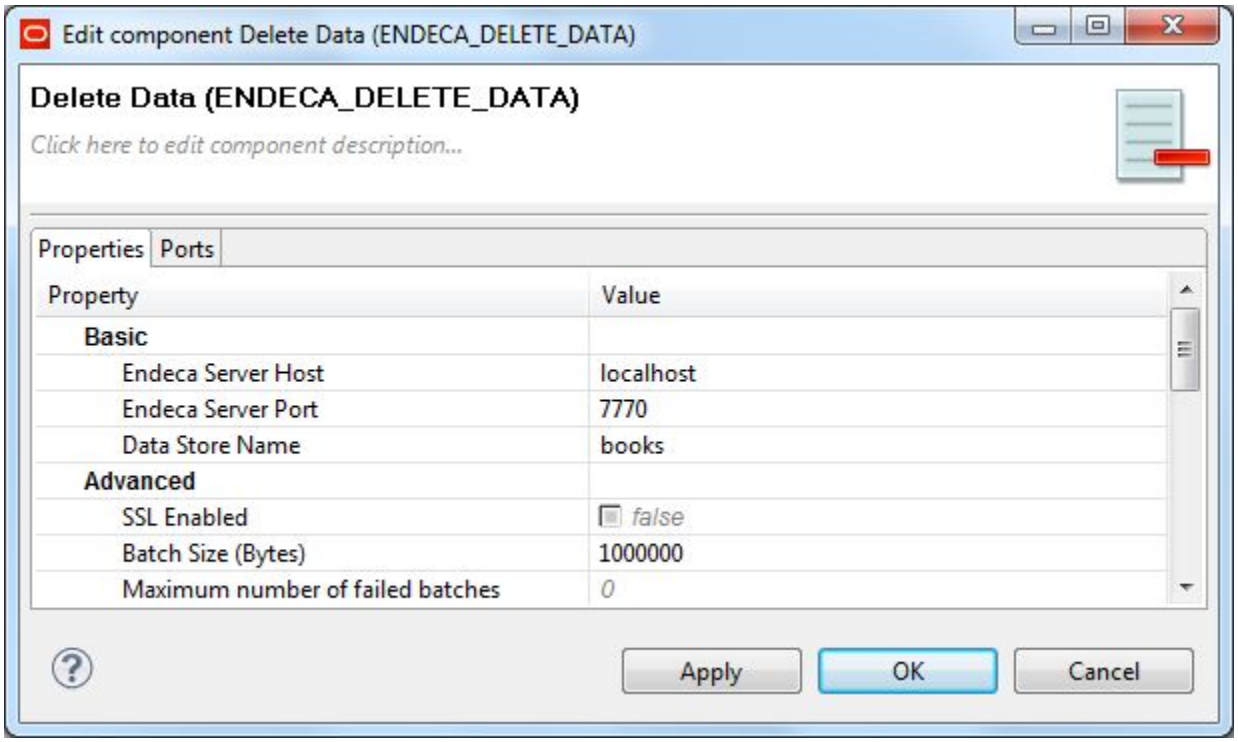
This procedure assumes that you have created a graph and added the **Delete Data** connector.

To configure the **Delete Data** connector:

1. In the Graph window, double-click the **Delete Data** component.
   The Integrator Edit Component dialog is displayed.

2. In the Edit Component dialog, enter these settings:
   (a) **Endeca Server Host**: The host name of the machine on which the Endeca Server is running.
   (b) **Endeca Server Port**: The port on which the Endeca Server was started.
   (c) **Data Store Name**: The name of the Endeca data store that contains the records to be removed or modified.
   (d) **SSL Enabled**: Toggle this field to `true` if the Endeca Server is SSL-enabled.
   (e) **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.
   (f) **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:



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 Endeca data store.

To run the graph to delete data from the Endeca data store:

1. Make sure that the Endeca Server is running on the host and port that are configured in the **Delete Data** connector. Also make sure that the Endeca data store is running.
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: 27/05/12 10:17:39
INFO  [WatchDog] - Node                   ID         Port      #Records         #KB aRec/s   aKB/s
INFO  [WatchDog] - ------------------------------------------------------------------------------
INFO  [WatchDog] - UniversalDataReader    DATA_READER0                                FINISHED_OK
INFO  [WatchDog] -  %cpu:..                          Out:0           3           0      3       0
INFO  [WatchDog] - Delete Data            ENDECA_DELETE_DATA0                         FINISHED_OK
INFO  [WatchDog] -  %cpu:..                           In:0           3           0      3       0
INFO  [WatchDog] - ---------------------------------** End of Log **---------------------------------
INFO  [WatchDog] - Execution of phase [0] successfully finished - elapsed time(sec): 1
INFO  [WatchDog] - ----------------------** Summary of Phases execution **---------------------
INFO  [WatchDog] - Phase#            Finished Status         RunTime(sec)    MemoryAllocation(KB)
INFO  [WatchDog] - 0                 FINISHED_OK                    1              6119
INFO  [WatchDog] - ------------------------------** End of Summary **-------------------------
INFO  [WatchDog] - WatchDog thread finished - total execution time: 1 (sec)
INFO  [main] - Freeing graph resources.
INFO  [main] - Execution of graph successful !
```

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 Endeca data store by the **Delete Data** connector.

Chapter 19

Creating an Endeca Data Store Snapshot

This chapter describes how to build a graph that creates a snapshot of a running Endeca data store.

Snapshot operation

Configuring the snapshot WebServiceClient component

Running the graph to generate the snapshot

Snapshot operation

This topic describes the Administration Web Service operation that can create a snapshot.

A snapshot represents a data-layer view of the state of an Endeca data store index at a specific point in time. The "Capturing Snapshots" chapter of the *Oracle Endeca Server Administrator's Guide* fully describes snapshots and how you can restore an Endeca data store from a snapshot.

You use the Administration Web Service's `createSnapshotOperation` operation to create a snapshot. The following example shows the syntax for a snapshot create:

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

The meanings of the `createSnapshotOperation` attributes are as follows:

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

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

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

The operation can be placed in a request structure of a **WebServiceClient** component in an Integrator graph.

Configuring the snapshot WebServiceClient component

This topic describes how to configure the **WebServiceClient** component with the `createSnapshotOperation` operation.

This procedure assumes that you have created a graph and added the **WebServiceClient** component. It also assumes that the directory for the snapshot output has been created.

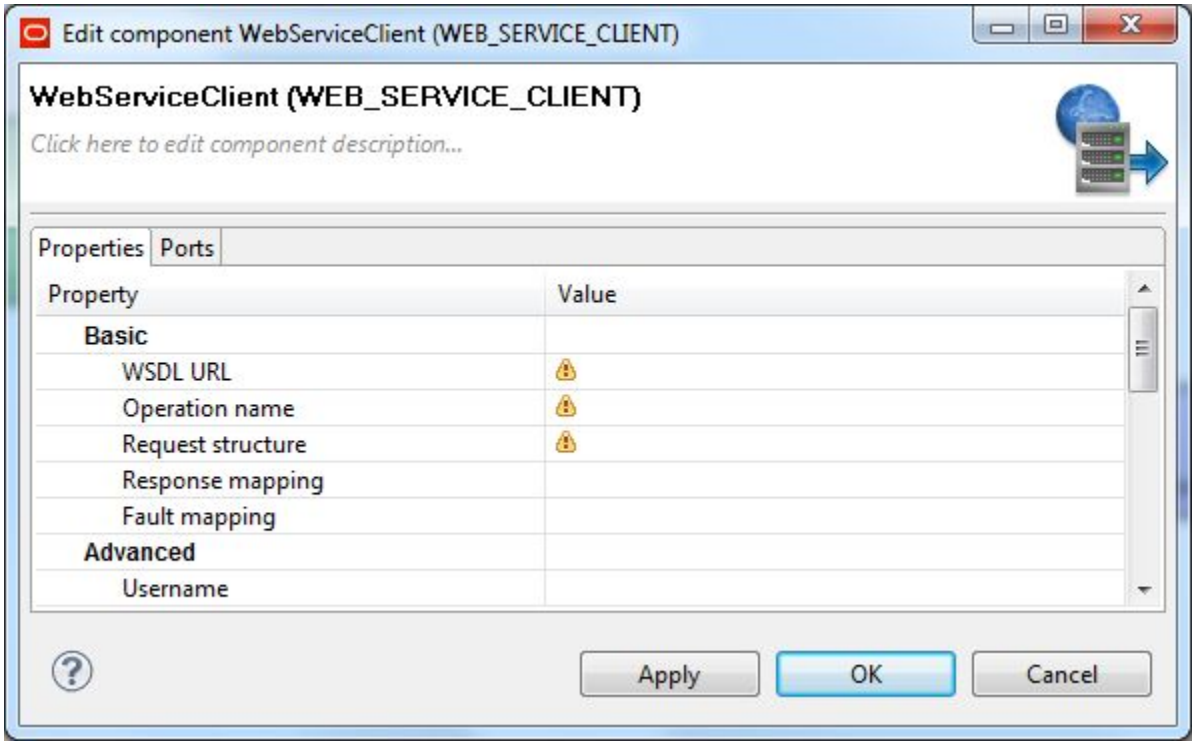
To configure the **WebServiceClient** component for a snapshot operation:

1. Make sure that the Endeca data store instance is running and its Administration Web Service is available by issuing a URL command (similar to the following example) from your browser. Be sure to use the correct port number of your Endeca Server (7770 in the example) and the name of the Endeca data store ("bikes" in the example).

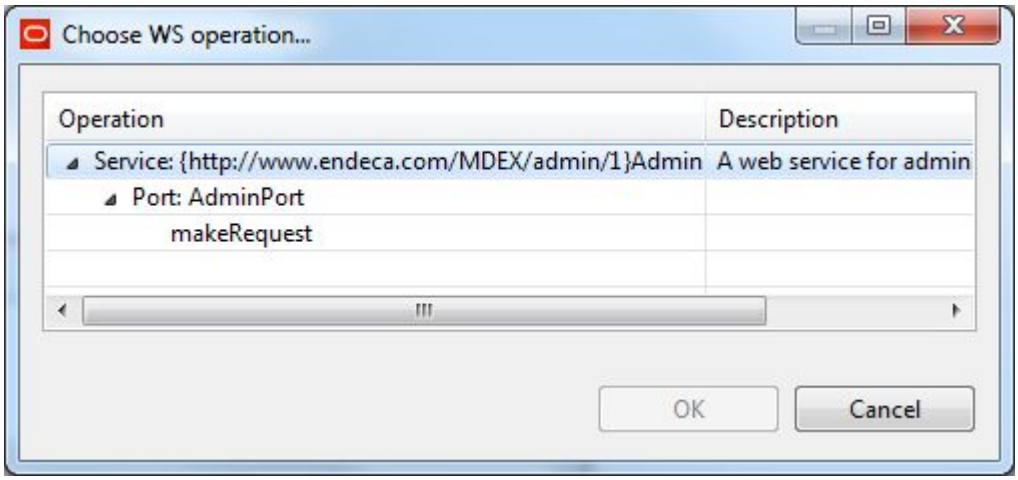
   ```
   http://localhost:7770/ws/admin/bikes?wsdl
   ```

   The URL command returns the WSDL of the Web service.

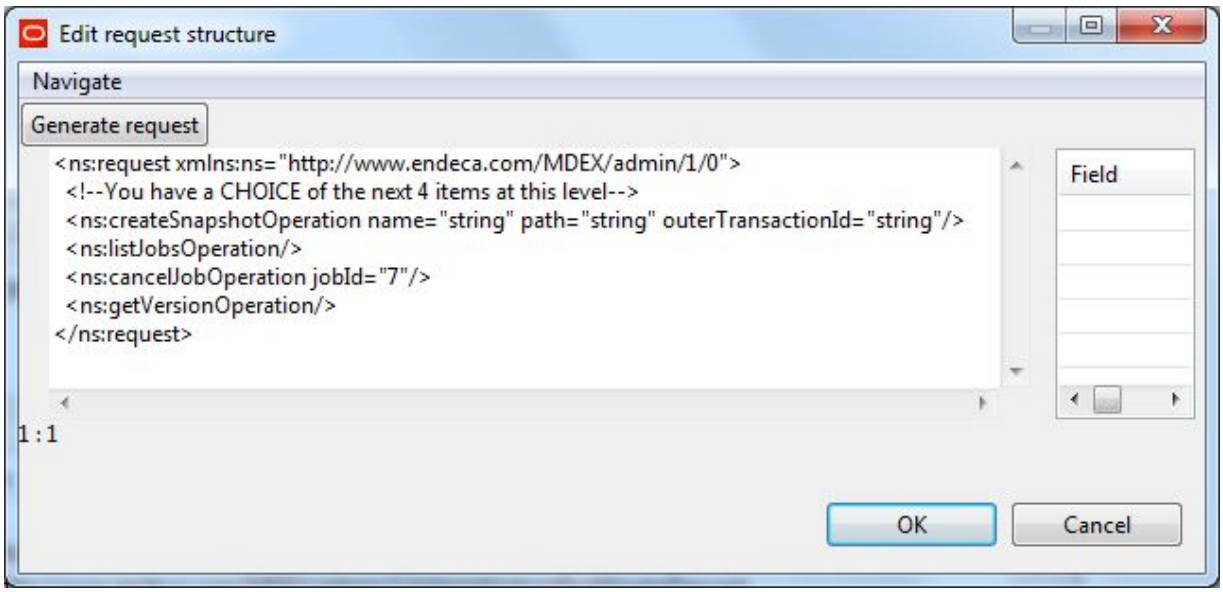
2. In the Graph editor, double-click the **WebServiceClient** component.
   The Edit Component dialog is displayed.



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:



5. In the **Choose WS operation** dialog, select **makeRequest** 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. Finally, click the Generate request button to generate a template request structure from the Web service:



7. Edit the template request structure as follows:
   (a) Remove the `listJobsOperation`, `cancelJobOperation`, and `getVersionOperation` elements.
   (b) For the `createSnapshotOperation` element, use the `name` attribute to specify a name for the snapshot and the `path` attribute to specify the parent directory in which the snapshot output will be placed. You can either remove the `OuterTransactionId` element (if you are not planning to use this component in an outer transaction), or specify a transaction ID.
   (c) Click **OK**.

The resulting text for the request structure should look similar to this example that does not use an outer transaction ID:

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

```
   <ns:createSnapshotOperation name="MySnapshot"
      path="file:///C:/Endeca/Apps/bikes/snapshots/"/>
</ns:request>
```

8. Optionally, you can use the **Component name** field to provide a customized name for this component.
9. When you have entered all your changes in the Edit Component dialog, click **OK**.
10. Save the graph.

After creating the graph and configuring the component, you can run the graph to generate a snapshot of the Endeca data store. In the Windows example above, the resulting data files directory is named `C:\Endeca\Apps\bikes\snapshots\MySnapshot`.

Running the graph to generate the snapshot

This topic describes how to run the snapshot graph.

To run the snapshot graph:

1. Make sure that you have an Endeca Server running on the host and port that are configured in the **WebServiceClient** connector. In addition, make sure that the specified Endeca data store has been started.
2. Run the graph.
   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 resulting snapshot directory will have three sub-directories.

Important: Because snapshots represent internal files needed to restore the data structures of the Endeca data store, they are not human-readable and should be treated as read-only. Modifying a snapshot can corrupt the Endeca data store.

Chapter 20

Wizards

This chapter describes the Information Discovery wizards in the Integrator.

About the Information Discovery wizards

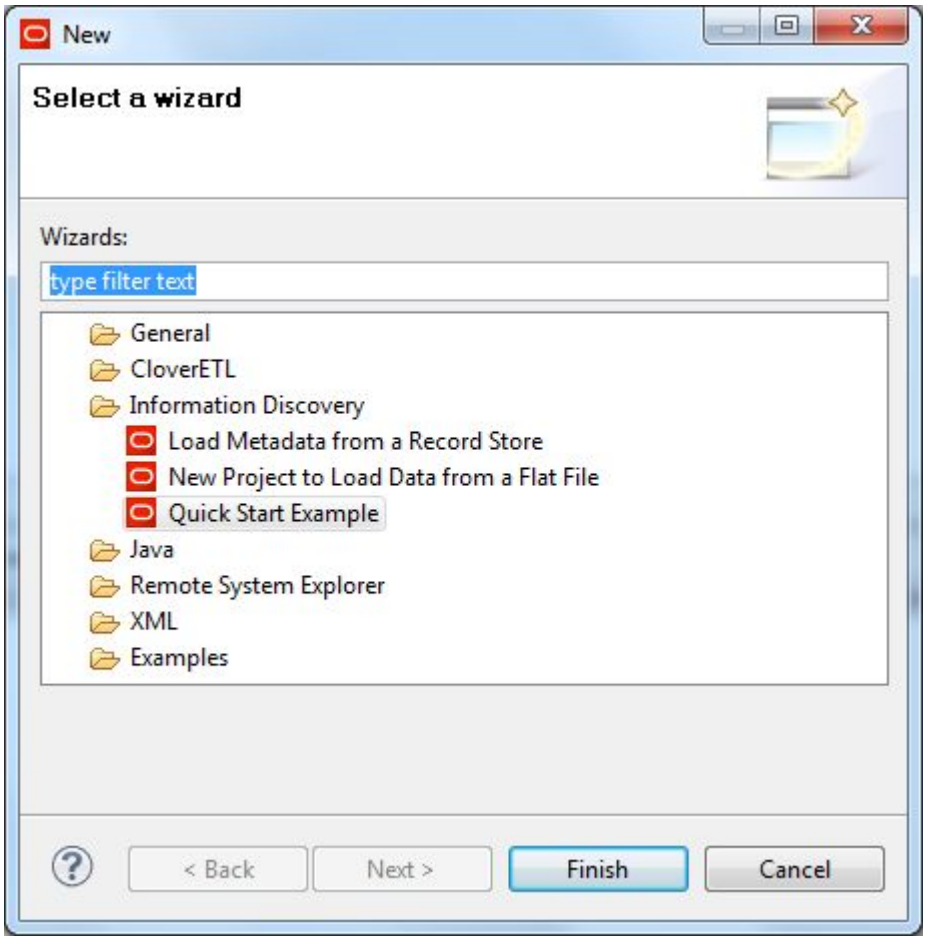
Quick Start project

New Project from a Flat File Wizard

About the Information Discovery wizards

There are three Information Discovery wizards.

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



The **Load Metadata from a Record Store** wizard generates metadata for a Record Store Reader by querying a CAS Record Store instance. This wizard is documented in the topic *Generating Edge metadata with the Record Store Wizard on page 166*.

The other two wizards are described in this chapter.

Quick Start project

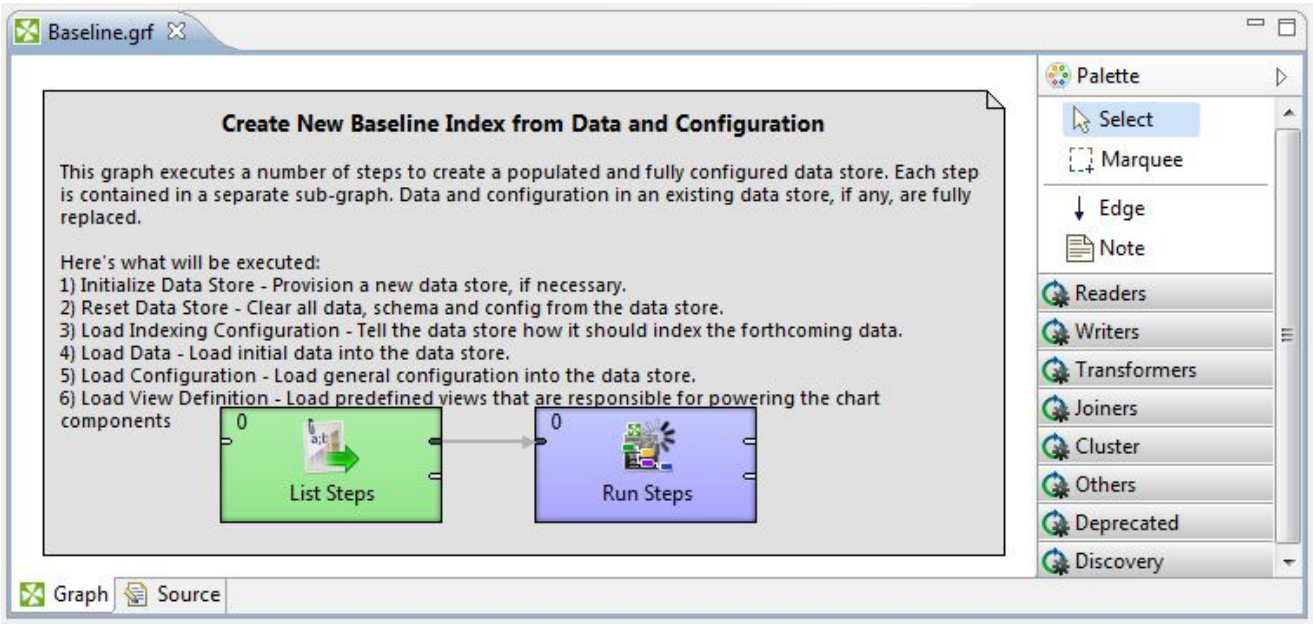
The Quick Start Wizard loads the Quick Start project.

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

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

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

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



Quick Start project components

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

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

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

Running the Quick Start project

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

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

New Project from a Flat File Wizard

This wizard makes it easy for you to create a new project from a flat file.

The **New Project to Load Data from a Flat File** wizard creates a fully-configured Integrator project from a source flat file of your choosing. When you run the project, one of the graphs profiles your data and uses the results to automatically choose a primary key (record spec) and construct graph components that perform such functions as loading the attribute schema, the search interfaces, attribute groups, and the source data.

The flat file will typically be a CSV (comma-separated values) file or a delimited or fixed-length text file. Note that this wizard supports the ingest of only one source file.

Creating a new project from a flat file

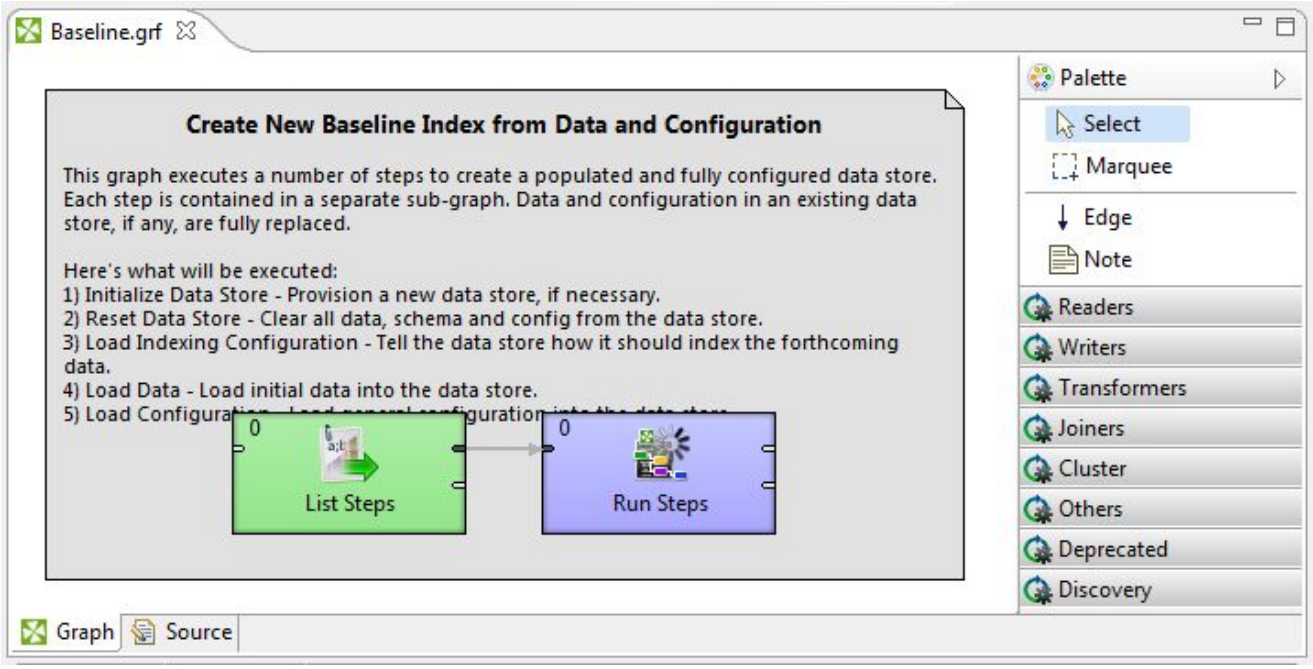
This topic describes how to run the New Project from a Flat File Wizard.

The only pre-requisite to this task is to have a flat file to use as the source input. You should clean the data so that it is consistent. For example, if you have a source property in which 95% of the property values are integers and the other 5% are string values, then change the strings to integers so that the property is parsed correctly by the data profiler. The flat file can be stored in another project in Integrator or on a file system directory on your machine.

To run the New Project from a Flat File Wizard:

1. From within Integrator, select **File>New>Other**.
   The Select a Wizard menu is displayed.
2. Select **New Project to Load Data from a Flat File** and then click **Next**.
3. In the **Project name** field of the next dialog, enter the name of the new project and then click **Next**. You can leave the **Use default location** box checked so that the project is created in your current workspace.
4. In the Data File dialog, click **Browse** and then use the URL Dialog to locate the source flat file. Click **OK** when you have selected the file.
   Note that you can use the **Local files** tab to locate files on your local file system.
5. After the file is loaded into the Data File dialog, click **Next**.
6. In the Metadata editor, click **Finish**.

The wizard creates the project with six graphs, including the Baseline graph from which you can run the project:



Before running the project, you should look at the project's `workspace.prm` file and make any appropriate changes for your environment (for example, if your Endeca Server is running on a non-default port).

Note that the first time that the project graph is executed, the schema and configuration will be determined by the ProfileData graph. Then if you make any changes to the schema or configuration, those changes will be lost on subsequent runs of the Baseline graph because the data will be profiled again.

Chapter 21

Connector Reference

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

Bulk Add/Replace Records connector

Add/Update Records connector

Add KVPs connector

Add Managed Values connector

Delete Data connector

Export Config connector

Import Config connector

Record Store Reader

Reset Data Store connector

Text Enrichment component

Text Tagger Regex component

Text Tagger Whitelist component

Transaction RunGraph connector

Visual and Common configuration properties

Connector output ports

Bulk Add/Replace Records connector

This connector adds new records or replaces existing records in an Endeca data store.

The **Bulk Add/Replace Records** connector adds or replaces records via the Dgraph's Bulk Load 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 Endeca data store 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, or the Dgraph configuration documents.

- 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 Endeca data store, the new standard attribute is automatically created with system default values for the PDR (see *Standard attribute default values on page 9*).
- No client-side batching is used and there is only a single, streaming connection to the Dgraph.
- You can run this connector in a sub-graph within a top-level graph that starts an outer transaction. For the bulk records operations to run successfully within an outer transaction, the connector relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.

The connector rejects non-XML 1.0 characters upon ingest. That is, a valid character for ingest must be a character according to production 2 of the XML 1.0 specification. If an invalid character is detected, the connector displays this error message:

```
Error: Character <c> is not legal in XML 1.0
```

The record with the invalid character is rejected, but the rest of the ingest operation continues.

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

Post ingest behavior

The default behavior of the Bulk Load Interface is to force a merge to a single generation at the end of every bulk-load ingest operation. This behavior is intended to maximize query performance at the end of a single, large, homogenous data update that would occur during a regularly scheduled update window.

However, the **Bulk Add/Replace Records** connector has a configuration property (named **Post Ingest Query Optimization**) that controls when this post-ingest merge occurs:

- If this property is set to `true` (the default), the merge is forced immediately after ingest.
- If the property is set to `false`, a merge is not forced at the end of an update, but instead relies on the regular background merge process to keep the generations in order over time. This behavior is more suitable for parallel heterogeneous data updates where low overall update latency is paramount.

A second configuration property (named **Post Ingest Dictionary Update**) controls when the aspell spelling dictionary is updated:

- A setting of `true` (the default) means a dictionary update is forced immediately after the ingest.
- A setting of `false` means the dictionary update is disabled. You can later update the dictionary via the `updateaspell` administrative operation, which is described in the *Oracle Endeca Server Administrator's Guide*.

By setting these properties in the connector, you can control the post-ingest merge behavior to meet the requirements of your application.

Metadata schema

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

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

Use cases

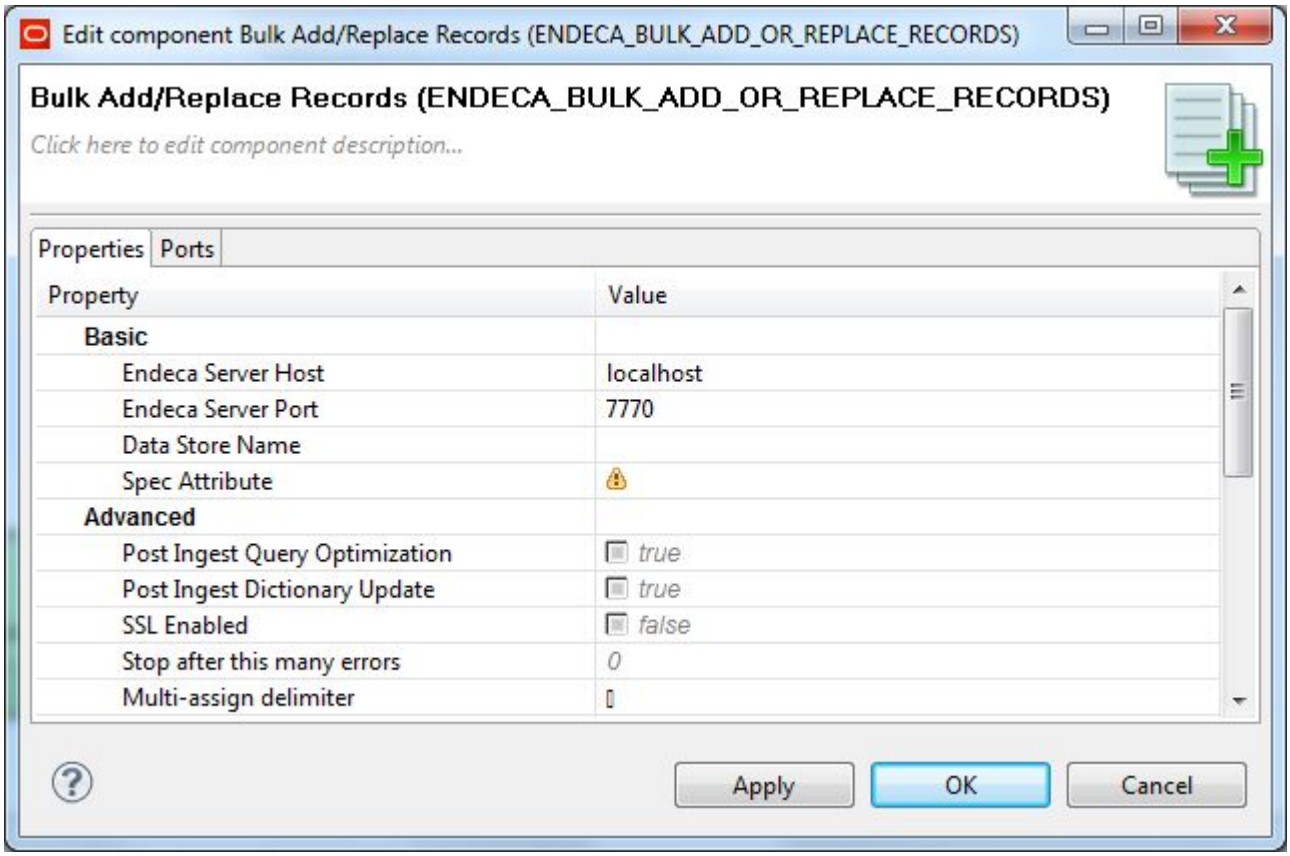
The **Bulk Add/Replace Records** connector is intended to be used with bulk data when delayed update visibility and compromised concurrent query performance are acceptable.

Some of the use cases for this connector are:

- Full index initial load of records, with no loaded schema. In this scenario, the Endeca data store has no user 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 with system default values.
- Full index initial load of records, with your record schema already loaded.
- Adding more new records to the Endeca data store any time after the initial loading of records. As in the initial load case, new standard attributes that do not exist in the Dgraph are automatically created with default system values.
- Replacing existing records in the Endeca data store 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 Integrator Edit component:



The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see *Visual and Common configuration properties on page 238*.

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started. The Endeca Server default port is 7770, but it can be changed via the server configuration file.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca data store, the property is automatically created with the system default values.

Configuration Property	Purpose	Valid Values
Post Ingest Query Optimization	Determines whether records are merged immediately after the ingest operation ends.	• If `true` (the default), the data merge is done immediately after the ingest operation finishes. • If `false`, the data merge is not done immediately. Instead, the data will later be merged according to the Dgraph's merge policy.
Post Ingest Dictionary Update	Determines whether the spelling dictionary is updated immediately after the ingest operation ends.	• If `true` (the default), the aspell dictionary is updated immediately. • If `false`, the dictionary update is disabled. You can update the dictionary via the `updateaspell` administrative operation.
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 Endeca Server. In this case, both the Endeca Server and the Endeca data store 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. The default is the Unicode DELETE character (\U007F). You can leave the field with its default value if your source does not have multi-assign properties.

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

Add/Update Records connector

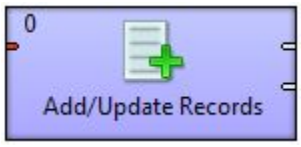
This connector adds new records or updates existing records in an Endeca data store.

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), or the Dgraph configuration documents (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 Endeca data store, the new standard attribute is automatically created with system default values for the PDR (see *Default values for new attributes on page 9*).
- Updates are batched on the client-side with multiple concurrent connections to the Dgraph.

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 Integrator field represents a property on a Dgraph record.

The metadata type of the Integrator field (as shown in the Integrator Metadata Editor) translates to the `mdex` property type. For example, the Integrator `integer` data type translates to the `mdex:int` property type. Note that this behavior can be overridden to support Integrator 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.

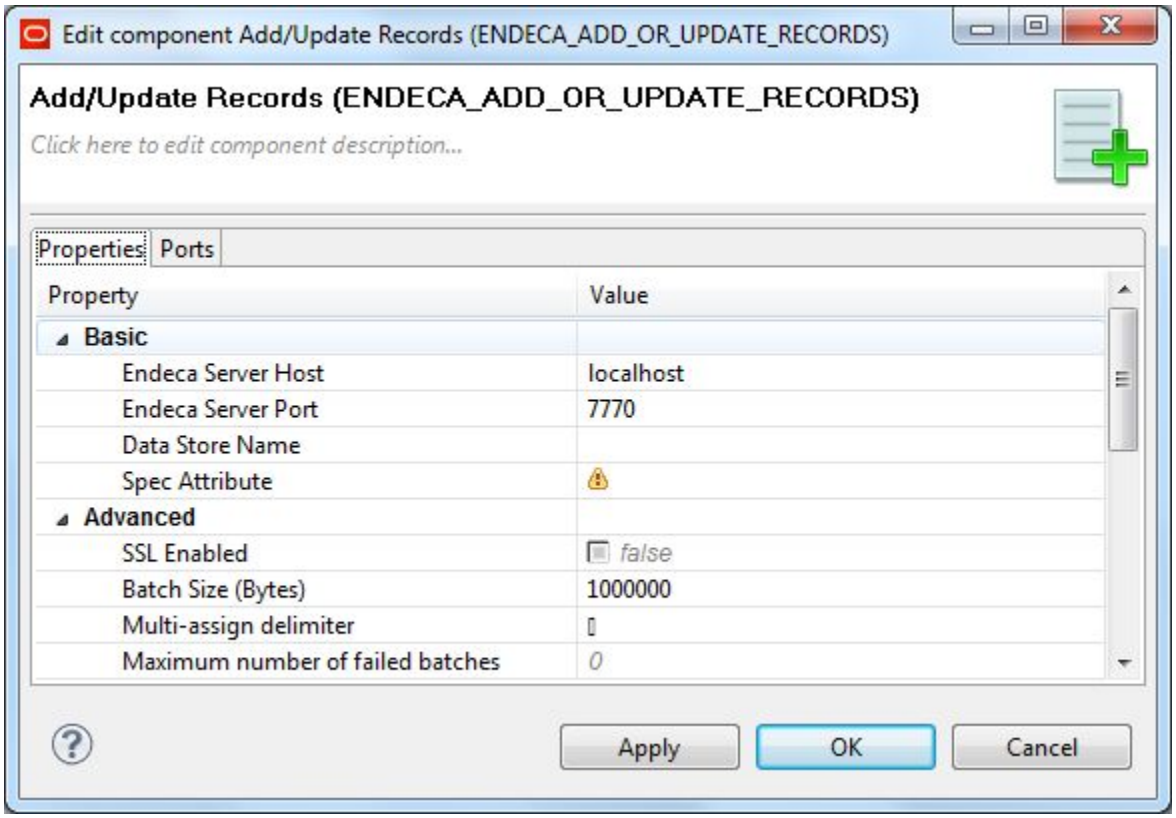
Some of the use cases for this connector are:

- Full index initial load of records, with no loaded schema. In this scenario, the Endeca data store has no user 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.
- 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 Endeca data store any time after the initial loading of records. As in the initial load case, new standard attributes that do not exist in the Endeca data store 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 Integrator Edit component:



The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see *Visual and Common configuration properties on page 238*.

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started. The Endeca Server default port is 7770, but it can be changed via the server configuration file.

Configuration Property	Purpose	Valid Values
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Dgraph, 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 Endeca Server. In this case, the Endeca Server 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 and sent to the Dgraph 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. The default is the Unicode DELETE character (\U007F). 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. Note that only the individual batch is extended, the overall batch size setting is not adjusted for future batches.

Add KVPs connector

This connector updates Endeca records in the Endeca data store 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 Dgraph, the new standard attribute is created by DIWS with system default values for the PDR (see *Standard attribute default values on page 9*). 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:

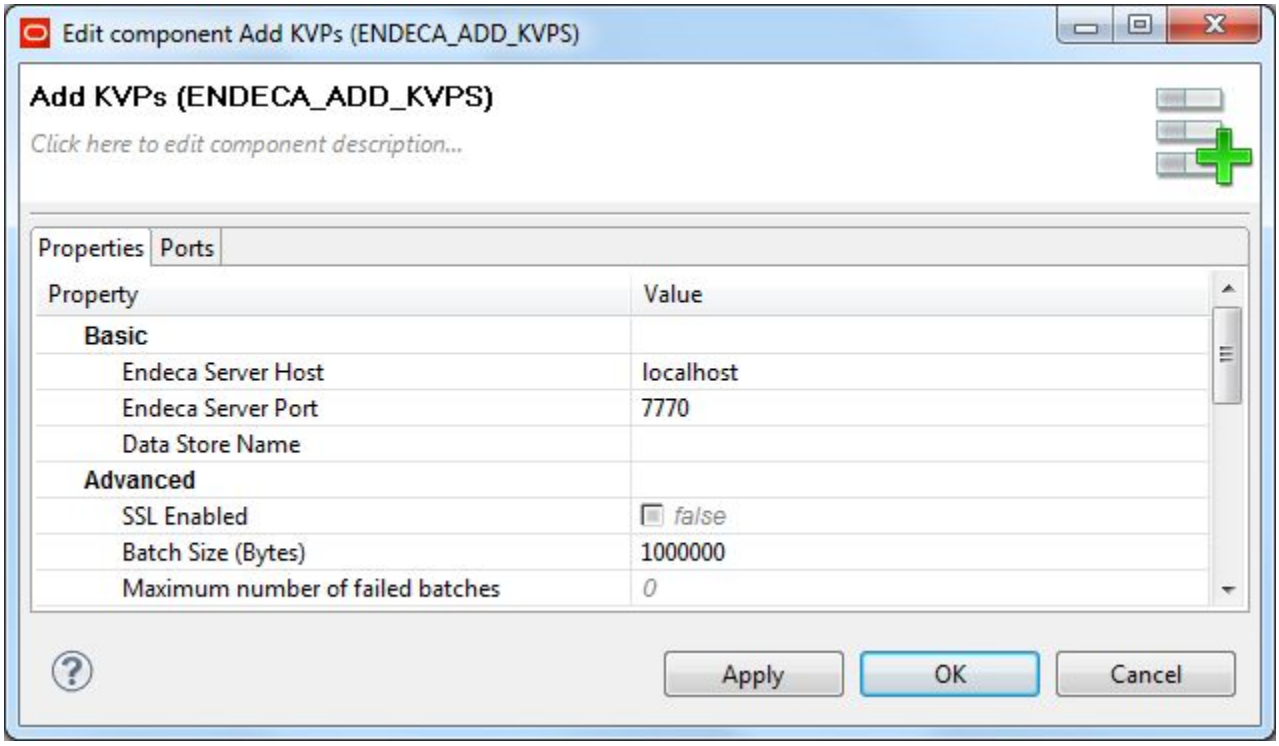
```
specKey|specValue|kvpKey|kvpValue|mdexType
```

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 Dgraph, 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 Integrator Edit component:



The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.

Configuration Property	Purpose	Valid Values
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 and sent to the Dgraph 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 Endeca data store.

The **Add Managed Values** connector is intended to load a taxonomy (Endeca managed attribute values) into the Endeca data store. 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 Endeca data store, the managed attribute is created by DIWS with system default values for the DDR and (if does not already exist) for the PDR. For a list of the default values, see *Default values for new attributes on page 9*.
- 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:

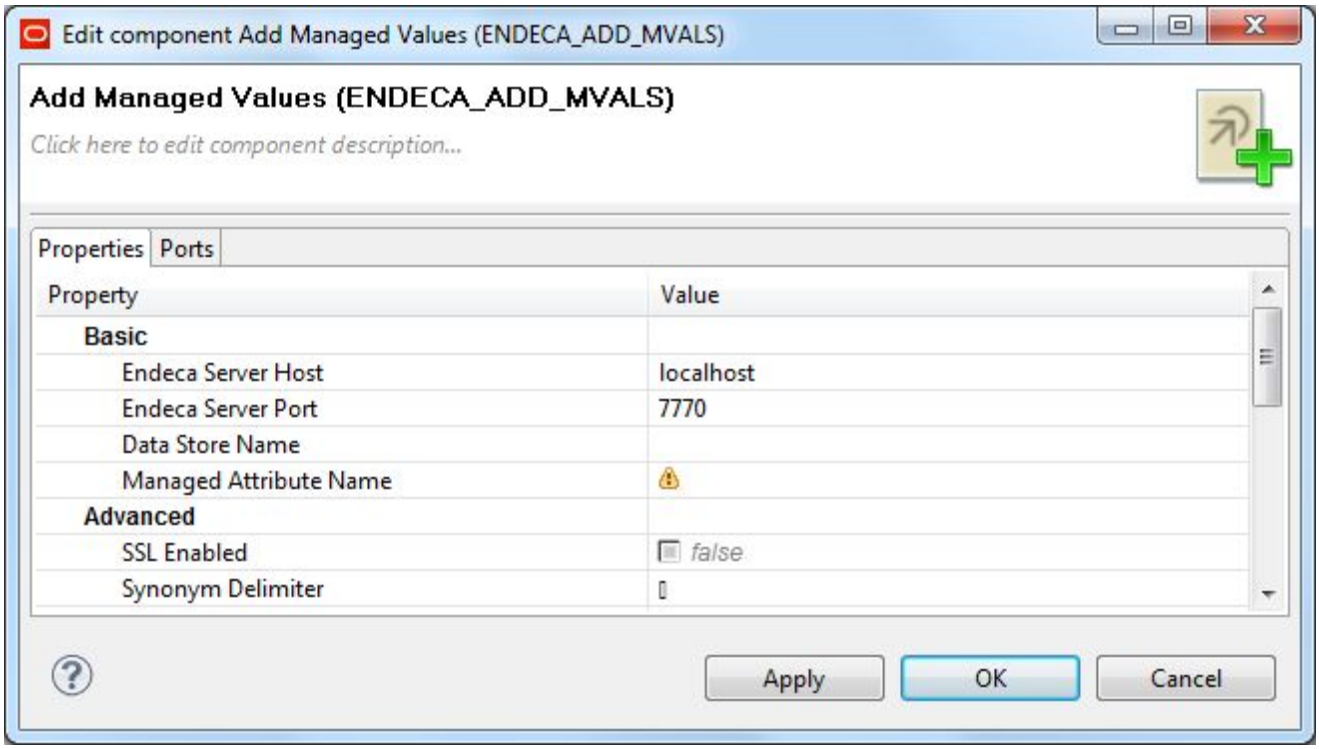
```
spec|displayname|parent|synonym
```

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 Integrator Edit component:



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

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.

Configuration Property	Purpose	Valid Values
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca data store, DIWS automatically creates the managed attribute with 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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.
Synonym delimiter	Sets the delimiter for specifying multiple synonyms.	A single character that is the synonym delimiter. The default is the Unicode DELETE character (\U007F).

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 this connector cannot be used to remove managed values from a taxonomy, but it can be used to remove managed attribute assignments 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:

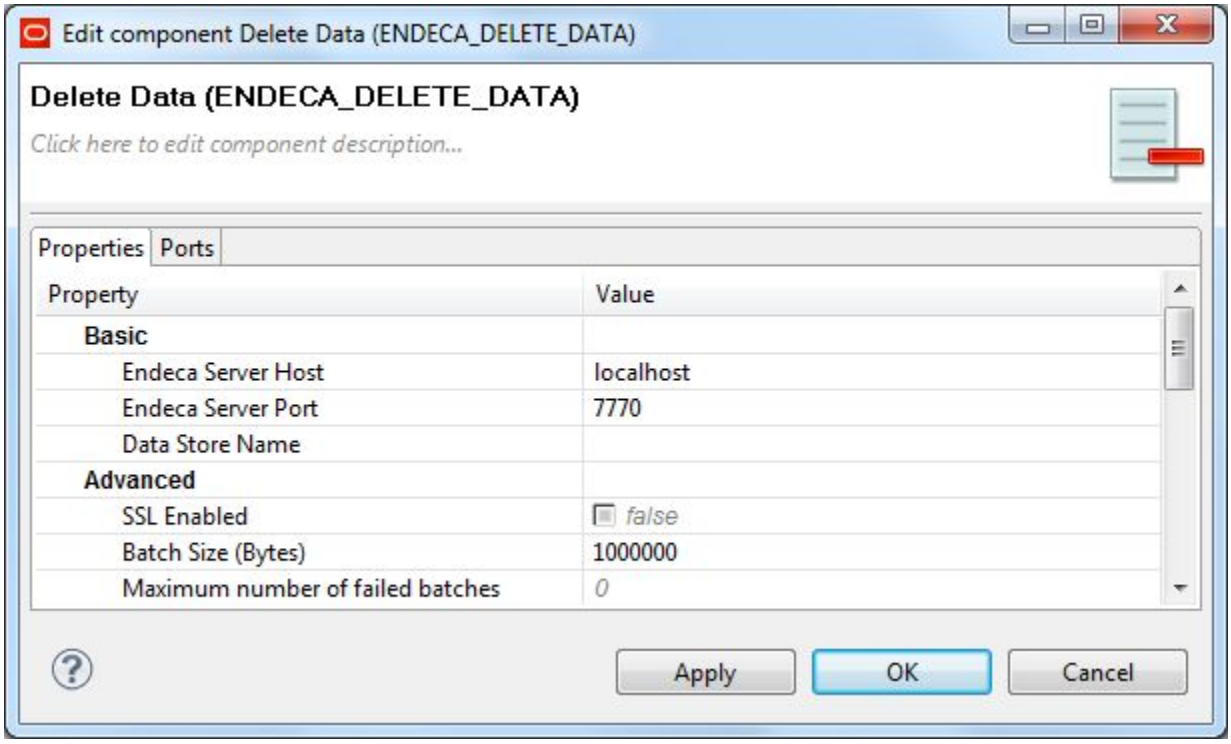
- *specKey* is the name of the primary key (record spec) of the record on which the delete operation will be performed.
- *specValue* is the value of the record's primary key.
- *kvpKey* is the name (key) of the Endeca standard attribute to which the assignment belongs. If *kvpValue* is blank, then all 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
ProductID|3000|Color|purple
ProductID|4000|Availability|
ProductID|5000||
```

Configuration properties

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



The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca Server. In this case, the Endeca Server 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 Dgraph 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 and sent to the Dgraph 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.

Export Config connector

This connector lets you export the schema and configuration stored in an Endeca data store.

The **Export Config** connector exports the schema and configuration using Configuration Web Service requests. The characteristics of this connector are:

- The connector lets you export your configuration and schema by pointing to an output port. This port can be connected to another component, such as any Writer component that would write the exported configuration and schema into a file. This file can later be used for importing.
- You can run this connector in a sub-graph within a top-level graph that starts an outer transaction. For the export operation to run successfully within an outer transaction, the connector relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.
- The connector exports all configuration and schema, but it does not export the `en_word_forms_collection` document that stores all word forms used for the stemming dictionary in the Dgraph. This document is loaded automatically when you create an Endeca data store, and is typically not modified.
- The connector also does not export your view definitions. To obtains these, use the Entity Configuration Service's `listEntities` operation.

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

Use cases

The **Export Config** and **Import Config** connectors are intended to be used in the following cases:

- Both of these connectors support cases where, after loading the default configuration, you change portions of it in Studio, such as attribute groups or attribute group names. From this point on, you may want to keep using this changed configuration, even if you run subsequent data updates. The connectors allow you to do this.
- The **Export Config** should also be used as part of the graph in which you run a baseline update for loading data (although, it is not intended to be used with the initial baseline update).

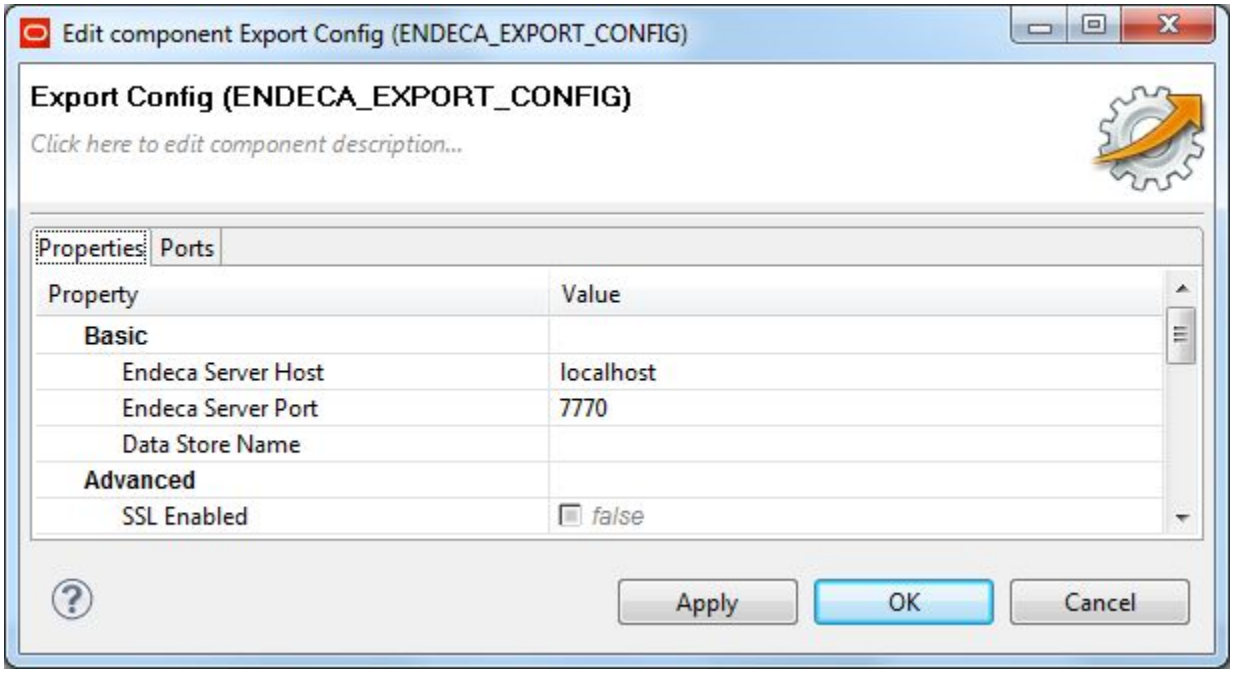
  In a typical scenario of a repeatable baseline update, you create a graph in which you start a transaction using the **Transaction RunGraph** component, export all configuration and schema using **Export Config**, run the **Reset Data Store** to remove all records and re-provision the Endeca data store, import the previously saved configuration and schema with **Import Config**, and then reload the records. At this point, the transaction can close and the node on which the baseline update was run can resume answering queries.

Metadata schema

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

Configuration properties

The configuration for the **Export Config** connector is set via the Integrator Edit component:



The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.

Import Config connector

This connector lets you import the schema and configuration into an Endeca data store.

The **Import Config** connector imports schema and configuration using Configuration Web Service operations. The characteristics of this connector are:

- This connector lets you import schema and configuration that was previously exported to a file. The **UniversalDataReader** component can read the file that stores the previously exported configuration and schema. The reader's output port can point to the input port on **Import Config** connector which imports this file.
- You can run this connector in a sub-graph within a top-level graph that starts an outer transaction. For the import operation to run successfully within an outer transaction, the connector relies on an outer transaction ID that you must specify in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.
- When importing, be aware that only basic XML validation takes place. Since the **Import Config** connector uses Configuration Web Service operations, the configuration that is sent to the Dgraph must be the one that the Configuration Web Service is designed to accept. Thus, the file that you are importing must comply with the requirements of the Configuration Web Service WSDL document, and contain only valid records describing the configuration and schema.

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

Use cases

The **Import Config** and **Export Config** connectors are intended to be used in the following cases:

- Both of these connectors support cases where, after loading the default configuration, you change portions of it in Studio, such as attribute groups or attribute group names. From this point on, you may want to keep using this changed configuration, even if you run subsequent data updates. The connectors allow you to do this.
- The **Import Config** should be used as part of the graph in which you run a baseline update for loading data (although, it is not intended to be used with the initial baseline update).

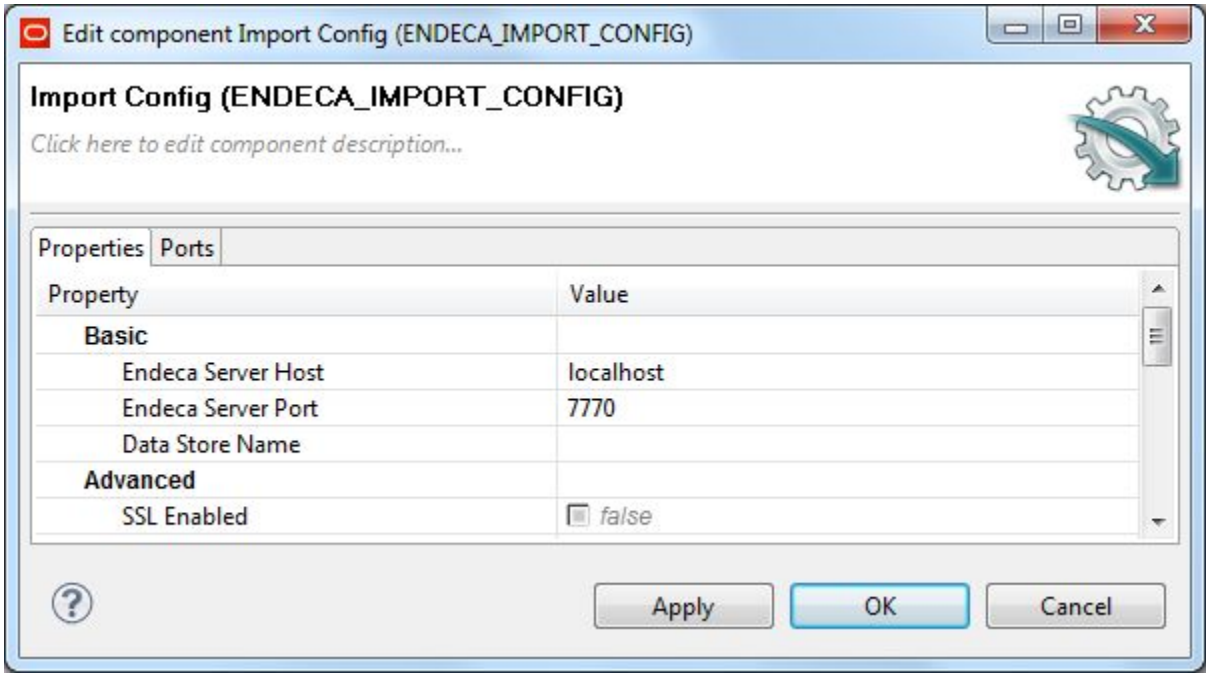
  In a typical scenario of a repeatable baseline update, you create a graph in which you start a transaction using the **Transaction RunGraph** component, export all configuration and schema using **Export Config**, run the **Reset Data Store** to remove all records and re-provision the Endeca data store, import the previously saved configuration and schema with **Import Config**, and then reload the records. At this point, the transaction can close and the node on which the baseline update was run can resume answering queries.

Metadata schema

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

Configuration properties

The configuration for the **Import Config** connector is set via the Integrator Edit component:



The configuration properties that you can set are:

Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Dgraph process for the data store must be running.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.

Record Store Reader

This reader component reads records from a CAS Record Store instance.

The **Record Store Reader** component reads Endeca records that have been stored in an Endeca Record Store. The records are data sources that have been crawled on file systems, content management systems, Web servers, and custom data sources. The extracted records can be written to an output file or loaded into an Endeca data store via an Information Discovery connector.

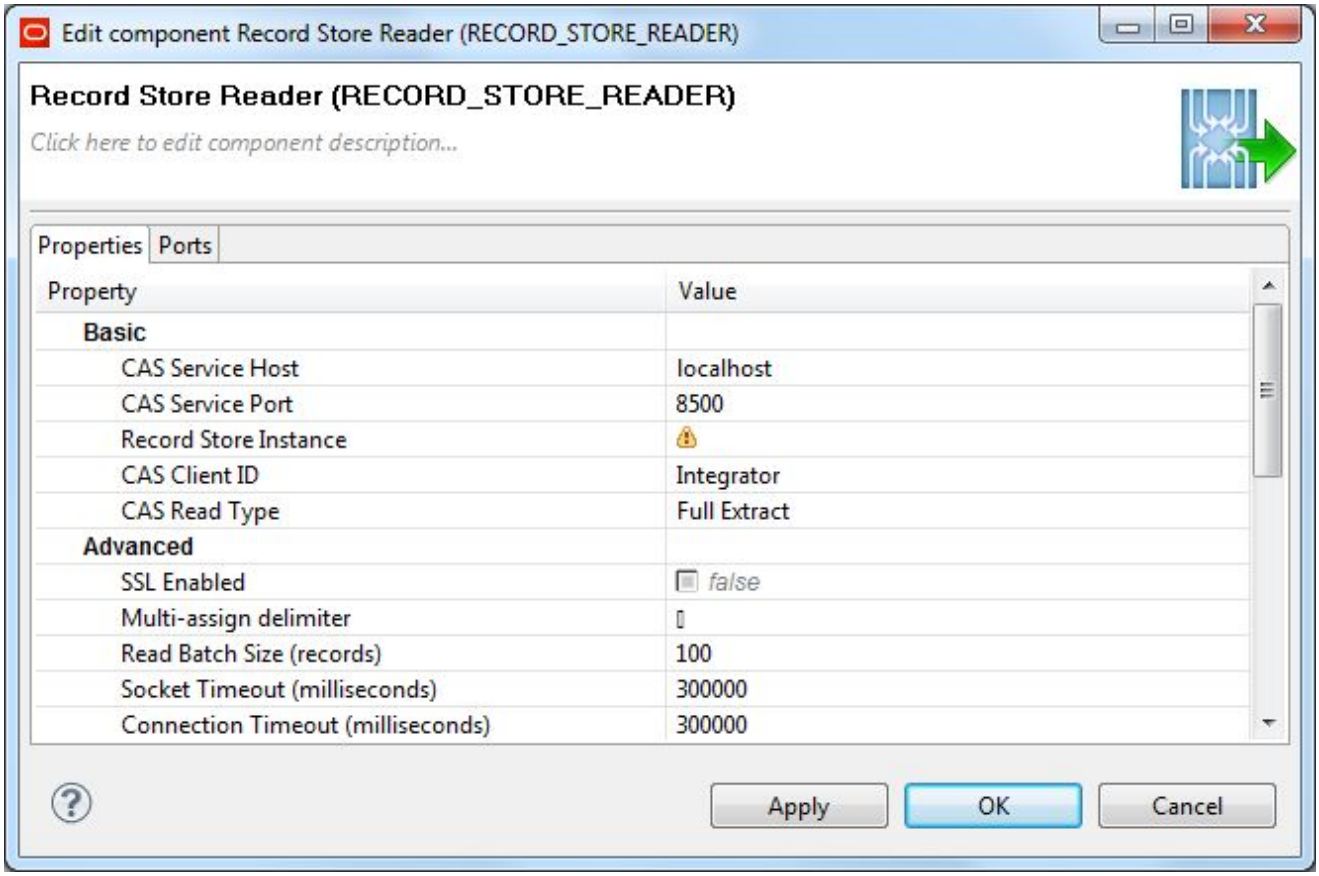
When added to a graph, the component icon looks like this:

Metadata schema

The metadata schema for the **Record Store Reader** component is not fixed. The metadata can be extracted from the Record Store instance by using the Record Store Metadata Wizard.

Configuration Properties

The configuration for the **Record Store Reader** component is set via the Integrator Edit component:



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

Configuration Property	Value
CAS Service Host	The name of the machine on which the Endeca CAS Service is running. The default name is `localhost`.
CAS Service Port	The port on which the CAS Service is listening. The default is `8500`.
CAS Record Store Instance	The name of the unique Record Store instance from which records will be read. There is no default name.
CAS Client ID	An arbitrary name that identifies this client to the Record Store. The default is `Integrator` but you can overwrite it with a client ID of your choosing. The Record Store uses this client ID to keep track of which generations have been read by this client. If multiple clients are accessing the Record Store, each client should have a unique name.
CAS Read Type	The drop-down menu in this field lets you select the type of read operation from the Record Store: • `Full Extract` is the default and retrieves baseline records (i.e., all the records from the last-committed generation in the Record Store). • `Incremental` retrieves delta records (i.e., the delta between two or more generations in the Record Store).
SSL Enabled	• Set to `true` if the Endeca CAS Service is SSL-enabled. • Set to `false` (the default), if the Endeca CAS Service is not SSL-enabled.
Multi-assign delimiter	The character that separates multi-assign values in an input property. The default is the Unicode DELETE character (\U007F).
Read Batch Size	The number of records in a fetched batch. The default is `100` records per batch.
Socket Timeout	The timeout in milliseconds for waiting for data from the Record Store (i.e., the maximum period inactivity between two consecutive data packets). The default is `300000` milliseconds (5 minutes). A timeout value of zero is interpreted as an infinite timeout.
Connection Timeout	The timeout in milliseconds until a connection is established with the Endeca CAS Service. The default is `300000` milliseconds (5 minutes). A timeout value of zero is interpreted as an infinite timeout.

Reset Data Store connector

This connector lets you reset the Endeca data store back to the empty state.

The connector does this by removing all the records (including the schema and view definitions) from the Endeca data store, re-provisioning the Endeca data store, and updating the spelling dictionary.

The characteristics of this connector are:

- The **Reset Data Store** connector utilizes operations from the Data Ingest Web Service. These operations delete all records (including schema records) and configuration, and provision the Endeca data store. Next, this connector utilizes an administrative command for updating the spelling dictionary (`admin?op=updateaspell`).
- You can run this connector in its own graph, or within a graph that starts an outer transaction. In particular, Endeca recommends to run the **Reset Data Store** connector within a **Transaction RunGraph**. Note that only one outer transaction can be open at a time.
- You can run this connector in a sub-graph within a top-level graph that starts an outer transaction. For the reset operations to run successfully, the connector relies on an outer transaction ID. You should specify this ID in the `OUTER_TRANSACTION_ID` parameter in the `workspace.prm` file in your project.

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

Use cases

The **Reset Data Store** can be used as part of the project graphs with which you run a baseline update (as in the Quick Start sample project).

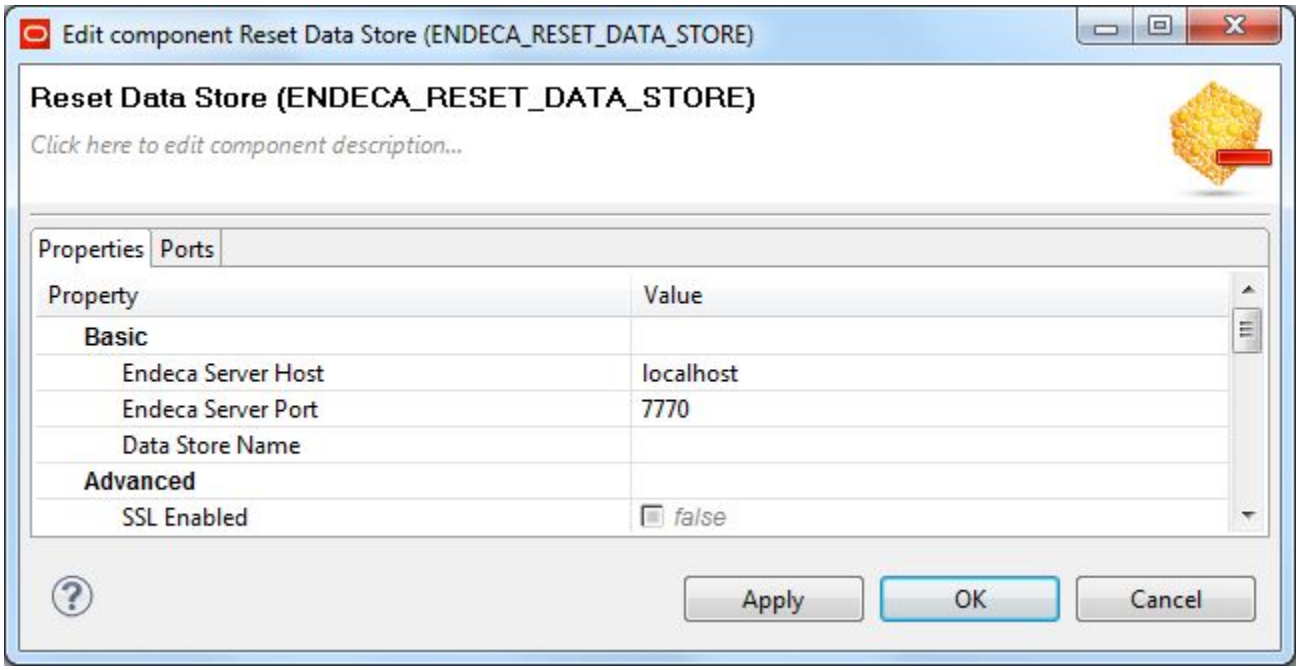
In a typical scenario of a repeatable baseline update, you create a graph in which you start a transaction using the **Transaction RunGraph** component, export all configuration and schema using **Export Config**, run the **Reset Data Store** to remove all records and re-provision the Endeca data store, import the previously saved configuration and schema with **Import Config**, and then reload the records. At this point, the transaction can close and the node can resume answering queries.

Metadata schema

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

Configuration properties

The configuration for the **Reset Data Store** connector is set via the Integrator Edit component:



The configuration properties that you can set are:

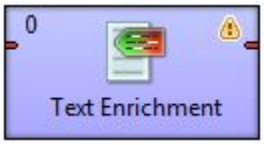
Configuration Property	Purpose	Valid Values
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.

Text Enrichment component

This transformation component provides basic entity extraction and summarization capabilities.

The **Text Enrichment** component utilizes the Salience Engine from Lexalytics to extract entities (people, places, organizations, themes, and quotes) from source files. The extracted entities can be written to an output file or loaded into an Endeca data store via an Information Discovery connector.

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

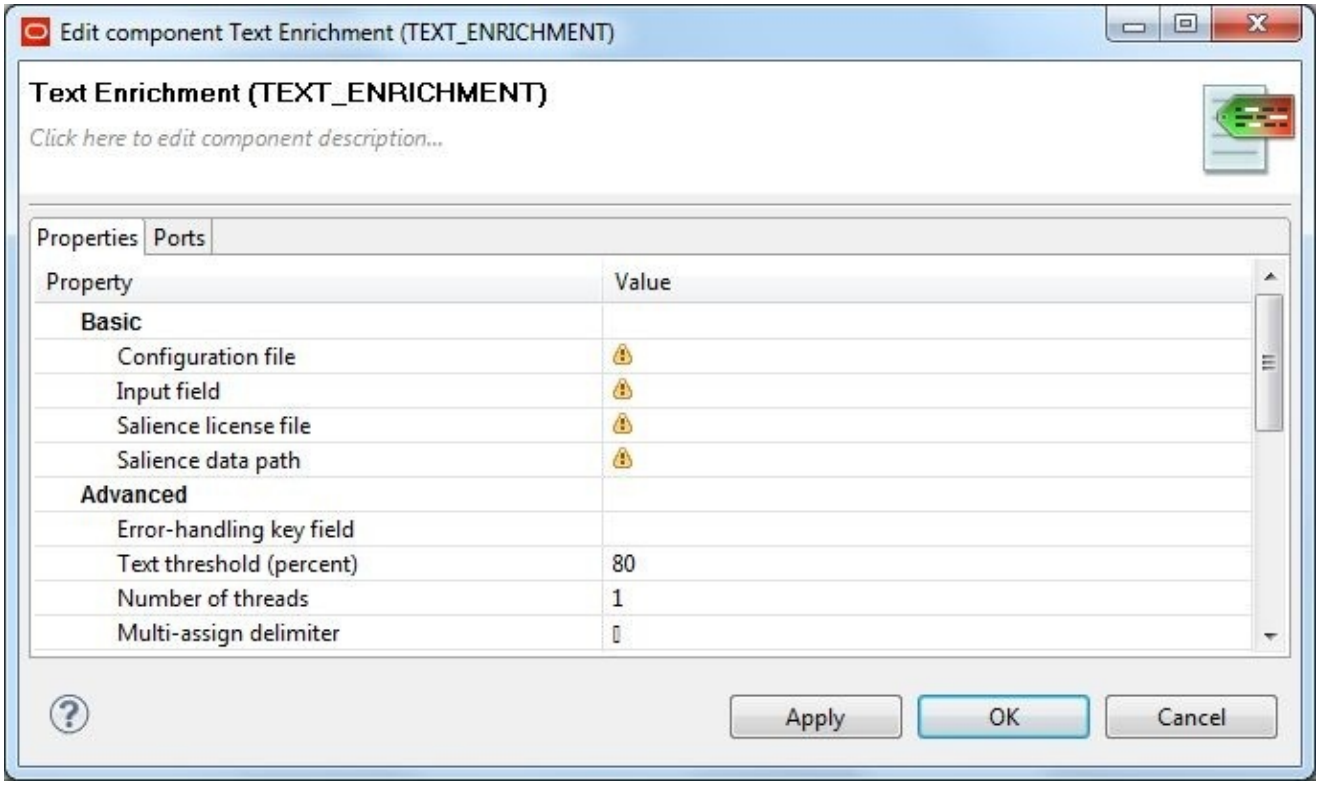


Metadata schema

The metadata schema for the **Text Enrichment** component is not fixed.

Configuration properties

The configuration for the **Text Enrichment** component is set via the Integrator Edit component:



The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see *Visual and Common configuration properties on page 238*.

Configuration Property	Value
Configuration file	The absolute path of the Text Enrichment properties file.
Input field	The name of the source field (in the source record) from whose data entities and sentiment will be extracted.
Salience license file	The absolute path of the Lexalytics Salience license file.
Salience data path	The absolute path of the Lexalytics `data` directory.
Error-handling key field	A field for error-handling output. Specifying a field name is mandatory. If you do not have a specific error field, you can specify the record specifier (primary key) field name (note that this record specifier field name must exist in the input metadata).
Text threshold (percent)	The minimum percentage of alpha-numeric characters that the input field has to have to be processed. If no threshold is provided, the default value for processing is `80`.
Number of threads	The number of threads the component should run on. If no thread count is provided, the default value is 1.
Multi-assign delimiter	The character that separates multiple values inside data fields. The default is the Unicode DELETE character (\U007F).

Text Tagger Regex component

This transformation component uses a regex (regular expression) to match text in a specified text field on the incoming records and then tags the output records with a property containing the text.

Besides specifying a regex search pattern, you must also specify a render pattern for the output text.

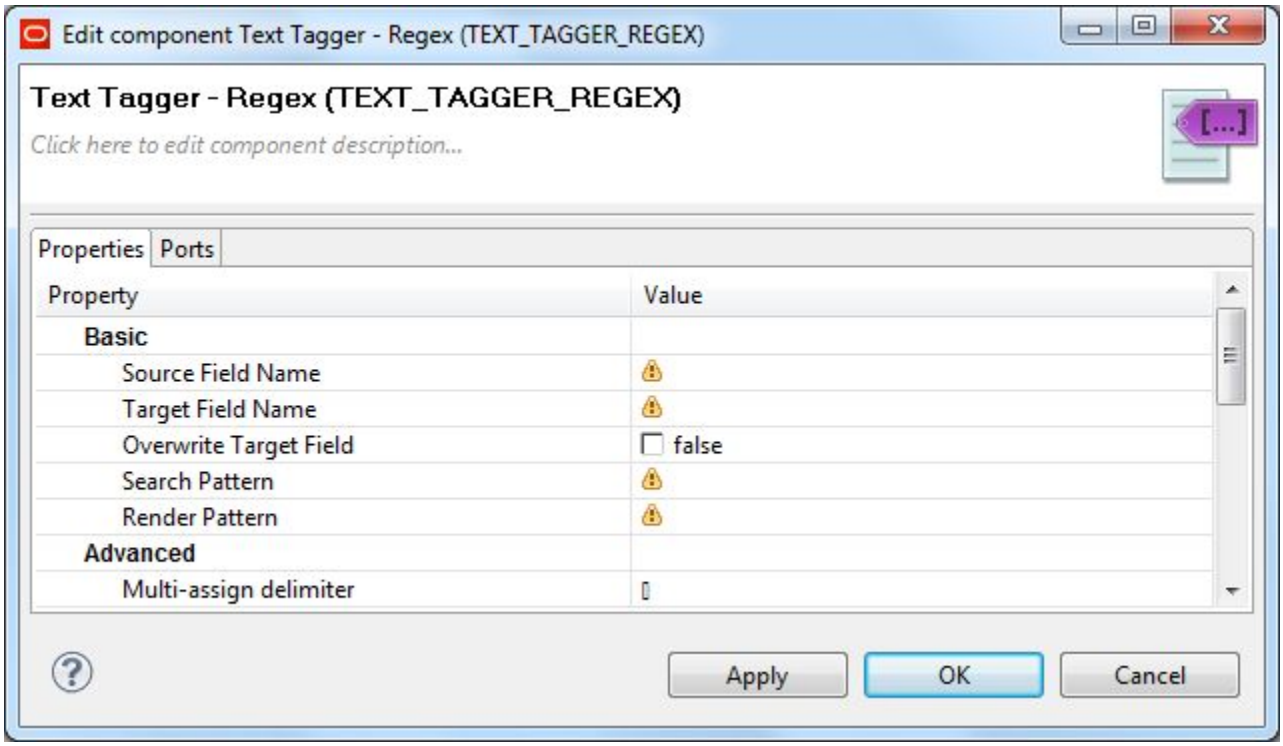
When added to a graph, the component icon looks like this:

Metadata schema

The metadata schema for the **Text Tagger Regex** component is not fixed.

Configuration properties

The configuration for the **Text Tagger Regex** component is set via the Integrator Edit component:



Configuration Property	Value
Source Field Name	The name of the text field (in the input records) in which the regex will search for matches.
Target Field Name	The name of the field (in the output records) into which the matched output is written.
Overwrite Target Field	• If `true`, the content of the target field will be overwritten by the matched output. • If `false` (the default), the matched output will be appended to the existing content of the target field, with the new content separated from the previous content by the multi-assign delimiter.
Search Pattern	The regex pattern to use when searching for matched text in the source field.
Render Pattern	The regex pattern to use when rendering (writing) the matched output in the target (output) field.
Multi-assign delimiter	The character that separates multiple matched values in the target field. The default is the Unicode DELETE character (\U007F).

Text Tagger Whitelist component

This transformation component uses a tags-rule file to define which terms to match in a specified text field on incoming records.

This component takes a list of search-term/tag-value pairs and searches for the terms in the configured unstructured text property (the source field) on the input records. If a term is found on a record, the tag value (from the tags-rule file) is tagged on the target field on the record.

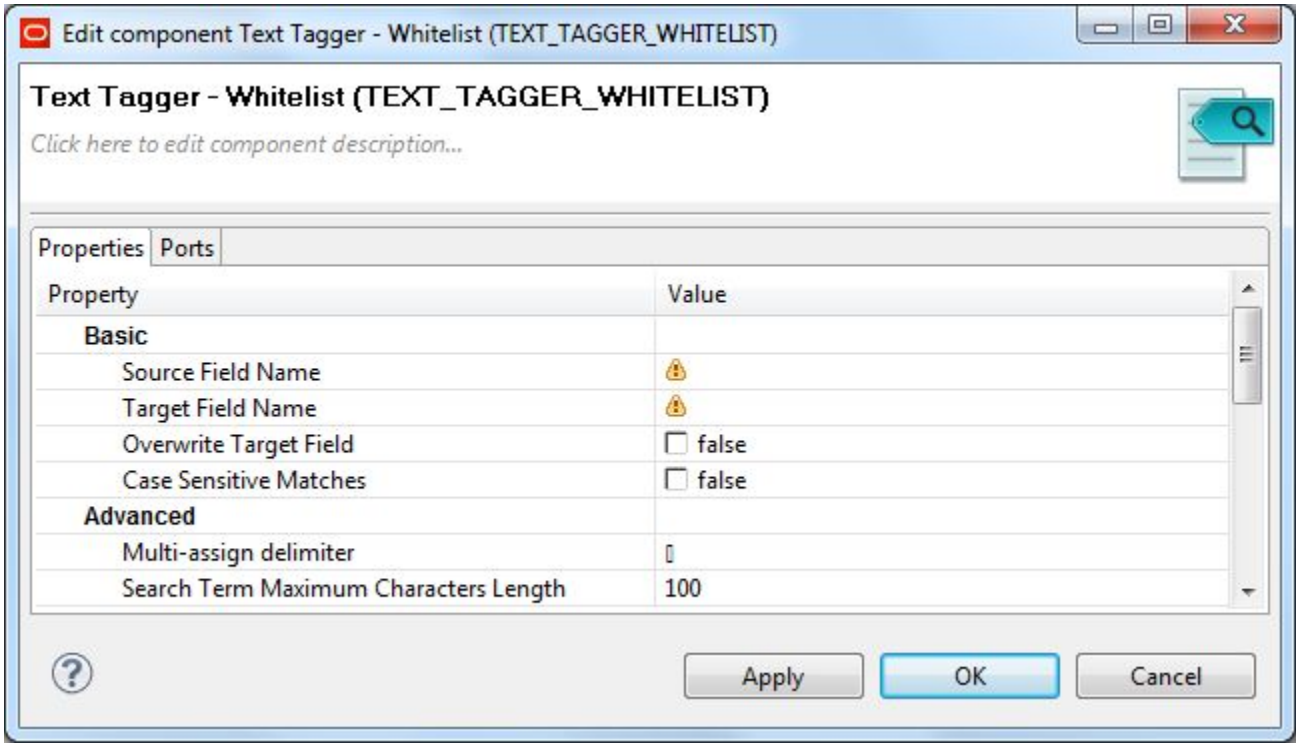
When added to a graph, the component icon looks like this:

Metadata schema

The metadata schema for the **Text Tagger Whitelist** component is not fixed.

Configuration properties

The configuration for the **Text Tagger Whitelist** component is set via the Integrator Edit component:



Configuration Property	Value
Source Field Name	The name of the text field (in the input records) in that should be matched against the terms in the tag-rules file.
Target Field Name	The name of the field (in the output records) into which the matched output is written.
Overwrite Target Field	• If `true`, the content of the target field will be overwritten by the matched output. • If `false` (the default), the matched output will be appended to the existing content of the target field, with the new content separated from the previous content by the multi-assign delimiter. Note that if you set this to `true` and the source records do not have a target field, then the graph will run successfully but you will see warning messages that the target field does not exist in the input metadata.
Case Sensitive Matches	• If `true`, the search for terms will be case sensitive. • If `false` (the default), the search will be case insensitive.
Multi-assign delimiter	The character that separates multiple matched values in the target field. The default is the Unicode DELETE character (\U007F).
Search Term Maximum Characters Length	The maximum length, in characters, of terms in the tag-rules file.

Transaction RunGraph connector

Use this connector to run Integrator graphs, similar to the standard **RunGraph** component available with the Integrator. Unlike the standard **RunGraph**, **Transaction RunGraph** starts the outer transaction and runs multiple sub-graphs within that transaction.

The **Transaction RunGraph** connector has the following characteristics:

- It is similar to the **RunGraph** component — it runs one or more Integrator graphs. If one sub-graph will be run inside **Transaction RunGraph**, you specify its name in the component's **Graph URL** attribute.

  If more than one sub-graph will be run, you can include names of all sub-graphs in a single file and send this file through the **UniversalDataReader** to the input port of the **Transaction RunGraph** connector.
- The **Transaction RunGraph** starts and commits an outer transaction using the Transaction Web Service. Only one outer transaction can be open at a time.
- In case of transaction failure, the connector rolls back to the state before the transaction had started, and commits the transaction. (This is the default behavior in case of a transaction failure, but you can configure other options, **Commit** and **Do nothing**, described below.)

- Because this connector starts an outer transaction, it uses the outer transaction ID as follows: The **Transaction RunGraph** overrides the transaction ID with the string `transaction`, for the duration of the transaction. This assumes that the project uses the empty string value for the ID in `workspace.prm`, specified as follows: `OUTER_TRANSACTION_ID=`. When **Transaction RunGraph** runs, the empty ID string is overwritten by the string `transaction`.
- All connectors or sub-graphs that:
  - Utilize a request to the Dgraph through a Web service or Bulk Load Interface, and
  - Run inside **Transaction RunGraph**

  must reference the outer transaction ID. You should specify this ID as an empty string as follows: `OUTER_TRANSACTION_ID=` in the `workspace.prm` file for your project. All Information Discovery-specific connectors that utilize Dgraph Web services or the Bulk Load Interface automatically reference this ID. Additionally, if these components are run within **Transaction RunGraph**, they use the ID `transaction`.
- If you are using a **WebServiceClient** component that is configured to run any of the Dgraph Web services, and plan to use this component inside **Transaction RunGraph**, the **Request Structure** field for the component must include an `OuterTransactionId` as the first element, with a value of an outer transaction.

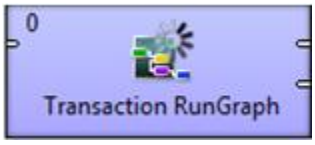
  Note: If you do not use outer transactions, then your Web service-based components should still use the `OuterTransactionID` referencing the value in `workspace.prm`. If the value is empty, the transaction ID attribute is ignored by the Dgraph. This allows components to run outside of transactions, without having to modify `workspace.prm`.

  For example, the following request specified in the **Request Structure** references the outer transaction ID as a parameter:

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

  In this example, the element `OuterTransactionId` specifies the ID of the outer transaction listed in the `workspace.prm` file for your project.

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



Use cases

The **Transaction RunGraph** should be used as part of the graph in which you run a baseline update:

- In a typical scenario of an initial baseline update, you create a graph in which you start an outer transaction using the **Transaction RunGraph** component, provision the Endeca data store using the **Reset Data Store** component, and then use one or more sub-graphs to load data and configuration.

- In a typical scenario of a repeatable baseline update, you create a graph in which you start an outer transaction using the **Transaction RunGraph** component, export all configuration and schema using **Export Config**, run **Reset Data Store** to remove all records and re-provision the Endeca data store, import the previously saved configuration and schema with **Import Config**, and then reload the records and the record attribute values. At this point, the outer transaction can close and the node on which the baseline update was run resumes processing query requests.

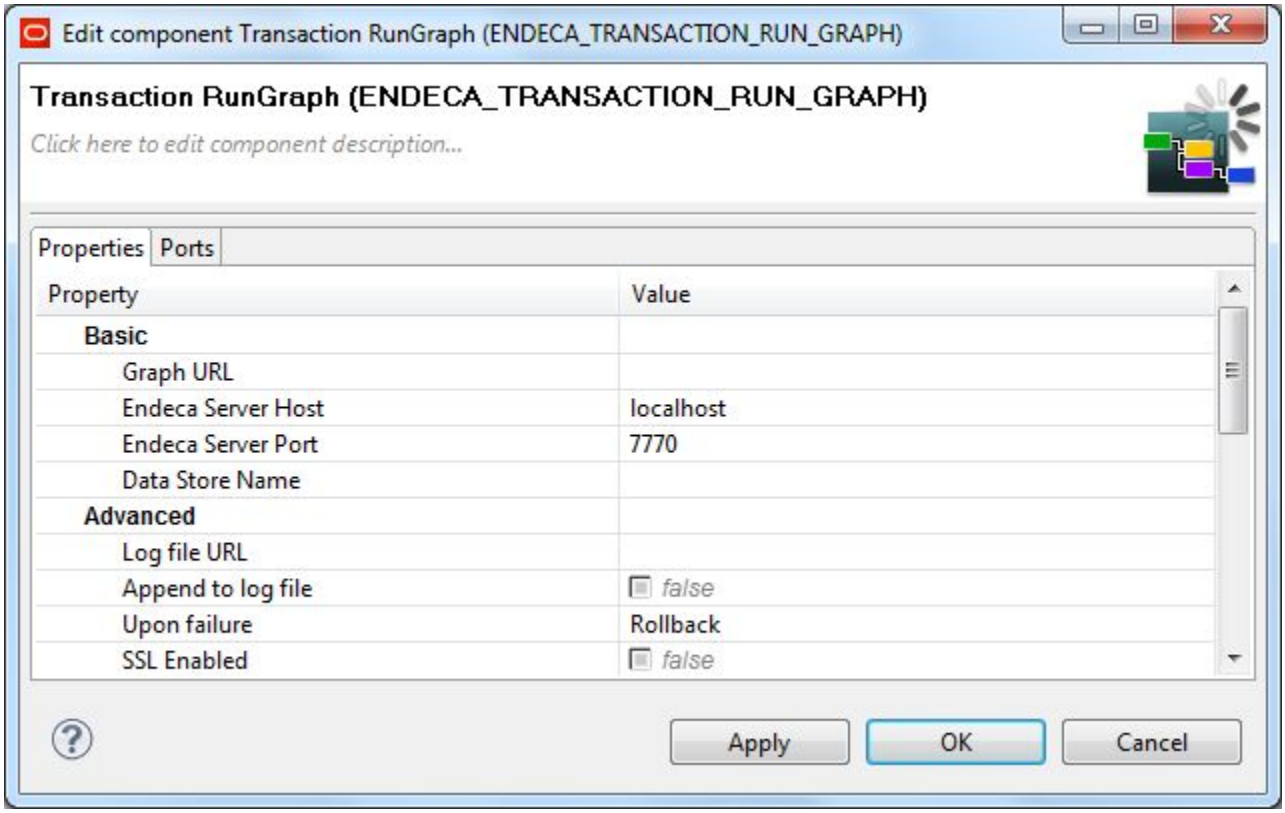
Metadata schema

The metadata schema for the **Transaction RunGraph** connector is not fixed.

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

Configuration properties

The configuration for the **Transaction RunGraph** connector is set via the Integrator Edit component:



The **Basic** and **Advanced** configuration properties that you can set are listed in the following table. For the other properties, see *Visual and Common configuration properties on page 238*.

Configuration Property	Purpose	Valid Values
Graph URL	Identifies the name of one graph, including path, that should be executed by the component.	Any of the graphs can be used. If standard graphs are used that call any of the Dgraph Web services, they should reference the outer transaction ID in their **Request Structure**, with the `<OuterTransactionId>` element.
Endeca Server Host	Identifies the machine on which the Endeca Server is running.	The name or IP address of the machine. `localhost` can be used as the name.
Endeca Server Port	Identifies the port on which the Endeca Server is listening.	The port number on which the Endeca Server was started.
Data Store Name	Identifies the Endeca data store into which the records will be added. The Endeca data store must be started.	The name of the Endeca data store that was specified with the Endeca Server `create-ds` command.
Upon failure	Enables selecting the behavior in case of failure.	• **Rollback**. This is the default. In case of transaction failure, enables to roll back to the state before the transaction had started, and commit the transaction. • **Commit**. In case of transaction failure, enables to commit those changes that have been made successfully before the failure had occurred, and commit the transaction. • **Do nothing**. In case of failure, does nothing. In this case, you may need to investigate the logs, and decide whether you want to apply any of the actions that are configured within the transaction manually. Note that in this case you may also need to manually stop the outer transaction by using a graph that runs the "commit transaction" operation.

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 Endeca Server. In this case, the Endeca Server must also be SSL-enabled.

Creating a Transaction RunGraph graph

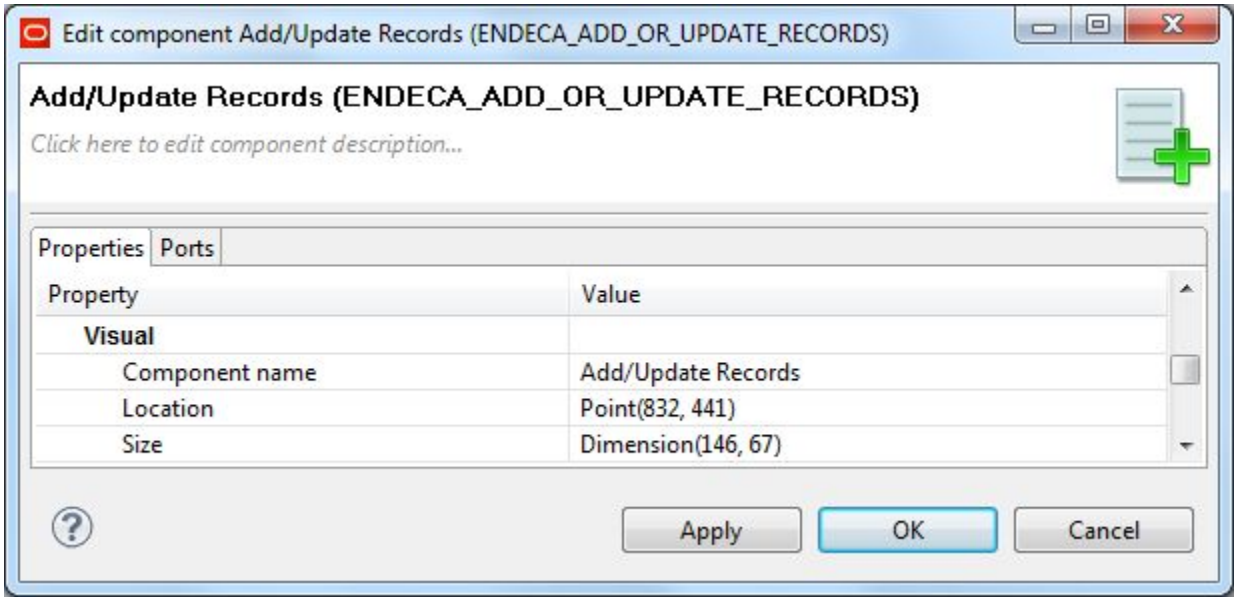
Visual and Common configuration properties

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

Information Discovery 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 *Oracle Endeca Information Discovery Integrator Guide*.

Visual properties

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

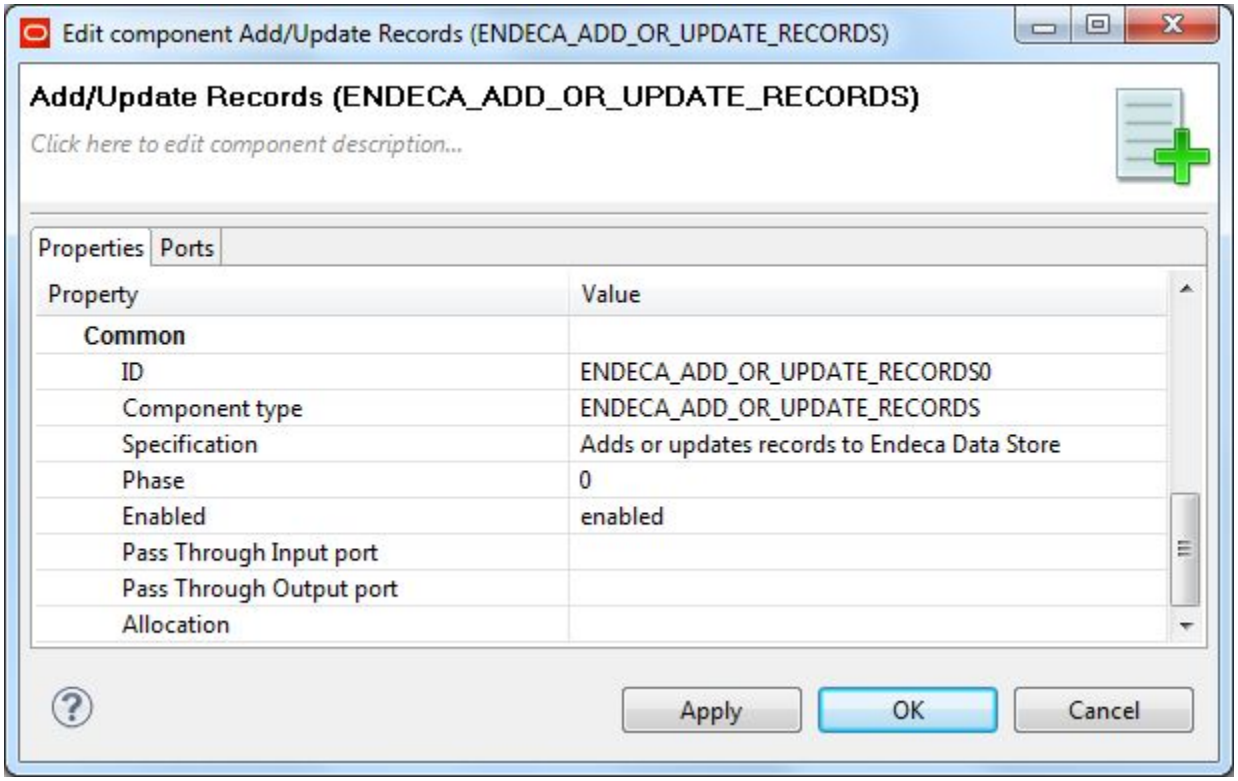


The **Visual** configuration properties are:

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 to a more descriptive one.
Location	Describes the location (using an X-axis and Y-axis) of the component icon within the graph.	Do not edit this field. Instead, use your cursor to move the component in the graph to the desired position.
Size	Describes the dimensions (size) of the component icon within the graph.	Do not edit this field.
Click here to edit component description	Located in the header of the component, this field lets you add some descriptive text that is displayed in the component icon in the graph.	Text describing what this component does (for example, text that best describes what this component does in the graph).

Common properties

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



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.
Allocation	If the graph is executed by a Cluster of Integrator Servers, this attribute must be specified in the graph.	For information on this property, see the *Oracle Endeca Information Discovery Integrator Guide*.

Connector output ports

The Information Discovery connectors that deal with ingest have two output ports each.

The two output ports are:

- **Port 0** returns status information. That is, it describes how many batches of records were successfully ingested.
- **Port 1** returns error information. That is, it describes the batches of records that failed to ingest. Note that each record corresponds to a failed batch, not individual records.

Port 0 metadata

Connector	Field 1	Field 2	Field 3	Field 4	Field 5
Add/Update Records	Start Row (Long)	End Row (Long)	Number of Records Affected (Long)	Time Taken in Seconds (Numeric)	n/a
Add KVPs	Start Row (Long)	End Row (Long)	Number of Records Affected (Long)	Time Taken in Seconds (Numeric)	n/a
Add Managed Values	Start Row (Long)	End Row (Long)	Number of Managed Attributes Added (Long)	Number of Managed Values Added (Long)	Time Taken in Seconds (Numeric)
Delete Data	Start Row (Long)	End Row (Long)	Number of Records Deleted (Long)	Number of Records Affected (Long)	Time Taken in Seconds (Numeric)
Bulk Add/Replace Records	Records Added (Long)	Records Queued (Long)	Records Rejected (Long)	State (String)	n/a

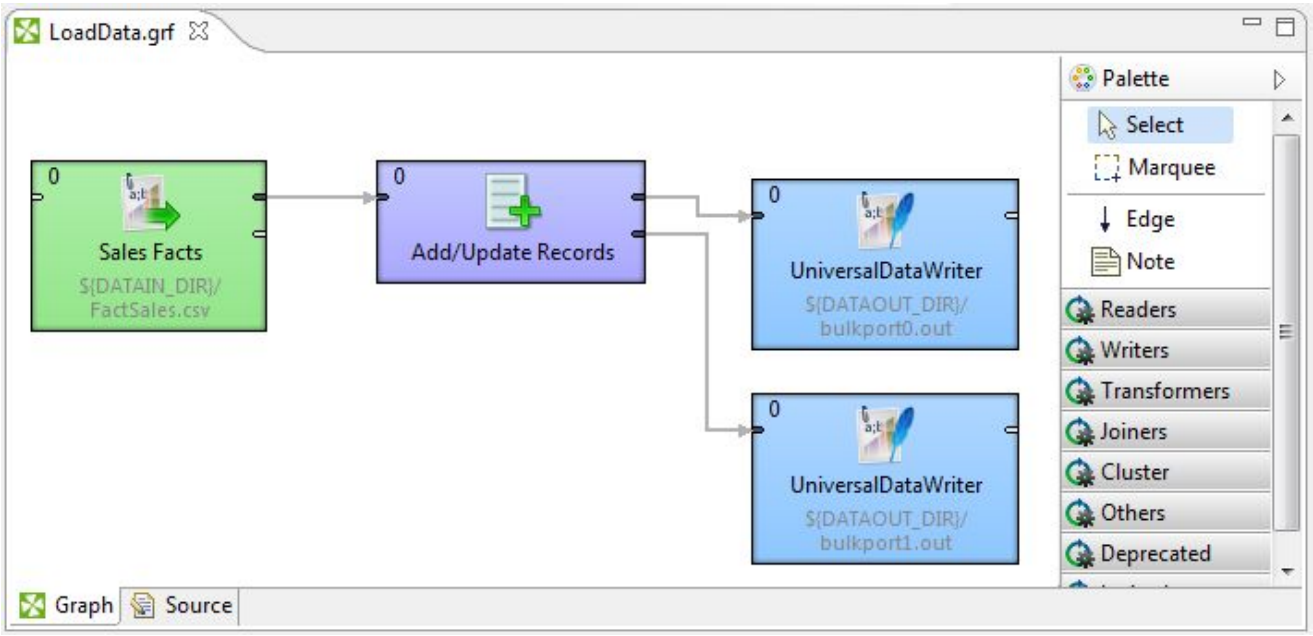
Port 1 metadata

Connector	Field 1	Field 2	Field 3
All DIWS connectors	Start Row (Long)	End Row (Long)	Fault Message (String)
Bulk Add/Replace Records	Fault Message (String)	n/a	n/a

Writing the output to a file

You can write the output port information to a file by connecting a Writer component to the output port of the Information Discovery connector. This sample graph has one **UniversalDataWriter** component writing out

data from port 0 of the **Add/Update Records** connector and a second one attached to Port 1 of the connector:



Chapter 22

Troubleshooting Problems

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

OutOfMemory errors

Transaction-related errors

Connection errors

Multi-assign delimiter error

OutOfMemory errors

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

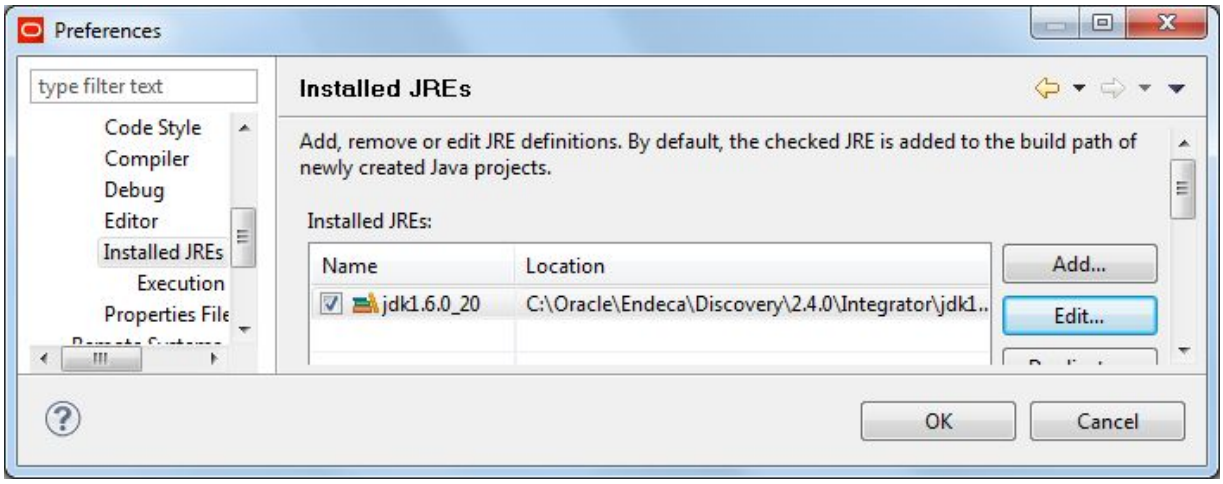
In an unsuccessful run, the Console Tab will show an `OutOfMemory` error similar to this example:

```
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
```

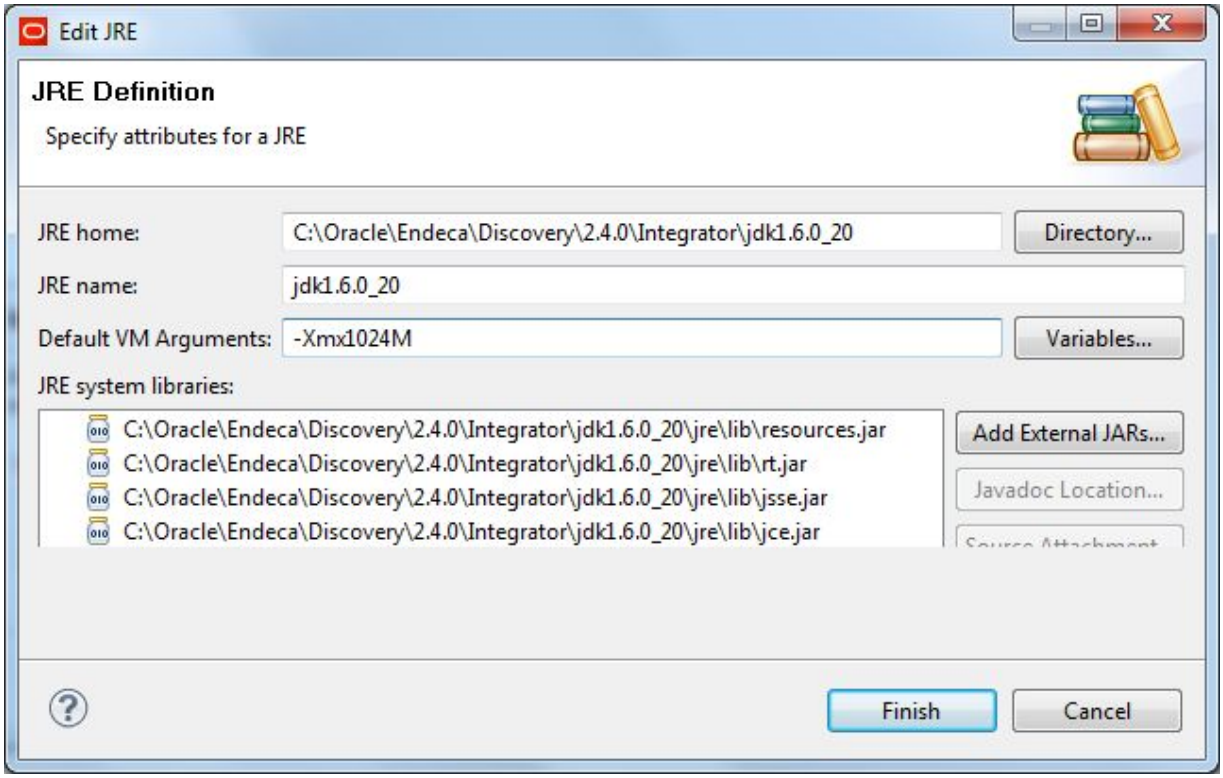
You can avoid these errors by increasing the memory allocated to the Java process running the service. The **Edit JRE** menu lets you increase the memory size on a global basis.

To avoid `OutOfMemory` errors:

1. Select **Preferences** from the **Window** menu.
2. From the **Preferences** menu, select **Java>Installed JREs**.
   The **Installed JREs** dialog is displayed.



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 this example:



5. Click **Finish** to apply your change and close the **Edit JRE** menu.
6. Click **OK** to close the **Preferences** menu.

Transaction-related errors

You may receive various outer transaction-related errors if you attempt to overlap running graphs wrapped in an outer transaction with graphs that do not start an outer transaction.

In general, only one outer transaction can be open and running at a time.

The following examples illustrate a few possible scenarios in which transaction-related errors may occur:

- Suppose you have two projects, one without an outer transaction (A), and one with it (B). Project A runs successfully until project B starts (and opens an outer transaction). If project A is half-way through its run and project B starts, the remaining steps in the project A will begin to fail because their components do not reference the outer transaction ID.
- Suppose you have two projects containing **Transaction RunGraph** connectors. They will run successfully if you run them serially, but any attempt to run them in parallel will result in the second project failing.
- Suppose you have a **Transaction RunGraph** set to **Do Nothing** as its failure action. If this graph fails the first time, it will also fail the second time you try to run it, because it is trying to open an outer transaction that has already been started. Therefore, if such a graph fails, to troubleshoot it, run the inner graphs separately, without running the **Transaction RunGraph**. Alternatively, you can manually commit the transaction after each failure, using the **RollBack Transaction** graph, specifying the ID of the transaction. Note that the ID defaults to the `transaction` string when run with a **Transaction RunGraph**.

Consider implementing one of the following recommendations (depending on your use case):

- Identify whether an outer transaction is currently running by issuing a `listOuterTransaction` request with the Transaction Web Service.
- Before running a graph that is configured to open its own outer transaction, verify that an already running outer transaction commits successfully.
- In some instances, when an already running transaction fails to commit, you may need to manually commit it by rolling it back. Once one outer transaction is closed, you can start a new transaction, if needed.
- Instead of running a new graph that starts an outer transaction separately, add a component that was previously part of this graph to any existing graph that starts an outer transaction.

  For example, you can run a graph for importing or exporting configuration and schema inside a **Transaction RunGraph**, or any other sample graph for running a baseline update (for subsequent data loading). Similarly, It is recommended that you run the **Reset Data Store** connector inside a graph that starts an outer transaction.

To summarize, to avoid transaction-related errors, ensure that projects containing transactions do not overlap. Errors are avoided if at any given time, only one outer transaction is open.

Committing or rolling back an outer transaction

Requirements for running graphs within a transaction

Connection errors

This topic illustrates connection errors that may occur between your Information Discovery connectors and the Endeca Server.

if an Information Discovery connector is incorrectly configured as the Endeca Server's host or port, an error similar to this example will result when the connector attempts to make a connection to the Endeca Server:

```
ERROR [ENDECA_ADD_KVPS1_0] - Connection refused: connect Error connecting
to the Endeca Server. If applicable, ensure your SSL settings are correct
ERROR [ENDECA_ADD_KVPS1_0] - Failed with the following exception:
    java.rmi.RemoteException: Connection refused: connect Error connecting
 to the Endeca Server. If applicable, ensure your SSL settings are correct;
 nested exception is:
    org.apache.axis2.AxisFault: Connection refused: connect
ERROR [WatchDog] - Graph execution finished with error
...
```

A similar error will also occur if the Endeca Server and/or the Endeca data store is not running or if a connector that is not enabled for SSL attempts to connect to an SSL-enabled Endeca Server.

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 ProductID:34699 with unknown dimension value
   "Red;Green" within dimension "ProductType"
ERROR [WatchDog] - Graph execution finished with error
ERROR [WatchDog] - Node ENDECA_BULK_ADD_OR_REPLACE_RECORDS0 finished
   with status: ERROR
```

In this example, the multi-assign source is "Red;Green" (with the semi-colon being the delimiter). To correct the problem, specify the correct multi-assign delimiter in the **Multi-assign delimiter** field of the connector's configuration screen.

Index

A

Add KVPs connector
- configuration properties 216
- configuring for key-value pair loads 118
- enabling SSL 216
- reference details 215

Add Managed Values connector
- adding to graph 125
- configuration properties 218
- configuring for taxonomy loads 127
- enabling SSL 219
- reference details 217

Add/Update Records connector
- adding to graph 49
- configuration properties 213
- configuring for incremental updates 50
- configuring for PDR output 64
- enabling SSL 214
- reference details 212

attribute schema
- configuration input file 54
- managed attributes input file 67

attribute schema load
- about 53
- loading DDRs 67
- loading PDRs 53

B

baseline update
- *See* full index load

Bulk Add/Replace Records connector
- adding to graph 38
- configuration properties 209
- configuring for full index load 45
- enabling SSL 211
- reference details 207

C

CAS Record Store Reader 163

Commit Transaction graph 29

Common configuration properties for connectors 239

configuration and schema, importing and exporting 172

configuration documents
- about 75
- adding components to graph 84
- configuring Reader component 84
- Global Configuration Record 76
- loading 82
- loading GCR 95

configuration file, Text Enrichment 132

Custom properties, creating 14

D

data types, supported 8

DDRs, loading
- *See* loading DDRs

Delete Data connector
- adding to graph 193
- configuration properties 220
- configuring for delete operation 195
- enabling SSL 221
- reference details 219
- types of delete operations 192

deleting data
- adding components to graph 193
- configuring Delete Data connector 195
- configuring metadata 194
- configuring Reader component 194
- running the graph 196
- source data format 192

Document Summary, Text Enrichment 131

E

Edge component
- configuring for deleting data 194
- configuring for full index load 41
- configuring for loading index configuration 85
- configuring for loading PDRs 56
- configuring for loading taxonomy 128
- configuring for transaction graph 26

Enabled configuration property for components 240

Export Config connector
- configuration properties 223
- configuring in a graph 173
- reference details 222

exporting configuration and schema 172

externally managed taxonomies, loading 122

F

flat file, creating new project from 205

full index load
- configuring Bulk Add/Replace Records connector 45
- configuring Edge component 41
- configuring Reader component 39
- creating graph 37
- overview 32
- running graph 47
- source data format 34

G

Global Configuration Record
- about 76
- loading 95

graph
- adding Add Managed Values connector 125
- adding Add/Update Records connector 49
- adding Bulk Add/Replace Records connector 38
- adding Delete Data connector 193
- adding Record Store Reader 164
- adding UniversalDataReader component 38
- creating empty 37
- running for full index load 47
- running to add key-value pairs 121
- running to delete data 196
- running to load incremental updates 51
- running to load Record Store records 170
- running to load taxonomy 129
- Text Enrichment 139

GUI overview, Integrator 1

I

Import Config connector
- configuration properties 225
- configuring in a graph 180
- reference details 224

importing configuration and schema 172

incremental updates
- overview 48
- running graph 51
- using the Add/Update Records connector 50

index configuration loads
- about 82
- configuring the Edge component 85
- using WebServiceClient component 92

Information Discovery connectors
- Add KVPs 215
- Add Managed Values 217
- Add/Update Records 212
- Bulk Add/Replace Records 207
- Delete Data 219
- enabling SSL 20
- Export Config 222
- Import Config 224
- overview 2
- Record Store Reader 226
- Reset Data Store 228
- Text Enrichment 230
- Text Tagger Regex 231
- Text Tagger Whitelist 233
- TransactionRunGraph 234
- Visual and Common configuration properties 238

Integrator
- about the Server 5
- additional documentation 11
- creating empty graph 37
- creating projects 33
- overview 1
- Quick Start sample project 203

J

Java heap space errors, avoiding 243

K

key-value pair loads
- about 116
- configuring Add KVPs connector 118
- configuring Edge metadata for graph 119
- configuring Reader for graph 117
- input format 116
- running graph 121

KVP loads 116

L

languages, loading stemming files for 96

loading DDRs
- configuring Reader and Edge components 69
- overview 67

loading managed attribute metadata
- configuring WebServiceClient component 73

loading PDRs
- configuring Add/Update Records connector 64
- configuring Edge component 56
- configuring Reader component 56
- overview 53

M

managed attribute name for taxonomy, specifying 127

managed attributes, default values for 10

managed values, loading 122

mdexType Custom properties, creating 14

metadata
- configuring for deleting data 194
- configuring for full index load 41
- configuring for loading index configuration 85
- configuring for loading PDRs 56
- configuring for loading taxonomy 128
- configuring for transaction graph 26
- supported data types 8

multi-assign data
- about 35
- configuring for Add/Update Records connector 51
- configuring for Bulk Add/Replace Records connector 47
- errors from misconfiguration 246

N

Named Entities extraction, Text Enrichment 131

New Project from a Flat File Wizard 205

O

order of loading data 8
outer transaction
about 22
error scenarios 245
performance impact 31
use with existing graphs 23
when to use in Integrator graphs 7
OutOfMemory errors, avoiding 243

P

PDRs, loading 53
Phase configuration property for components 240
precedence rules
about 102
configuration input file 104
configuring Reader component 106
graph components 105
primary key
about 35
configuring for Add/Update Records connector 51
configuring for Bulk Add/Replace Records connector 46
project, creating 33

Q

Quick Start sample project 203
Quotation extraction, Text Enrichment 131

R

record schema load
See attribute schema load
record spec property
See primary key
Record Store Reader
about 163
adding to graph 164
component configuration properties 226
component reference details 226
configuring 165
running graph 170
Wizard 167
Reformat component
baseline loads 42
loading precedence rules 108
loading standard attribute schema 58
restoring managed attribute schema 70
restoring views 187
regular expressions for Text Tagger 148
Reset Data Store connector
configuration properties 228
reference details 228
Rollback Transaction graph 29

S

Salience Engine version for Text Enrichment. 132
Sentiment Analysis, Text Enrichment 130
Server, overview of Integrator 5
snapshots
create operation 198
graph 199
source data format
attribute schema 54
deleting data 192
full index load 34
incremental updates 48
key-value pair loads 116
managed attribute schema 67
precedence rules 104
taxonomy loads 123
Text Tagger Whitelist tag-rules source 156
transaction graph 24
SSL enablement
Add KVPs connector 216
Add Managed Values connector 219
Add/Update Records connector 214
Bulk Add/Replace Records connector 211
Delete Data connector 221
Export Config connector 223
for connectors 20
Import Config connector 225
Reset Data Store connector 229
Transaction RunGraph connector 238
standard attributes, default values for 9
stemming files
configuring WebServiceClient 99
loading 96

T

tag-rules source format, Text Tagger Whitelist 156
taxonomy loads
configuring Add Managed Values connector 127
configuring Reader component 125
creating graph 124
metadata configuration 128
overview 122
running graph 129
source input file 123
specifying managed attribute name 127
Text Enrichment
about 130
component configuration properties 230
component reference details 230
configuration 132
configuring component 142
configuring Reader component 140
Document Summary 131

graph 139
Named Entities 131
Quotations 131
Sentiment Analysis 130
supported features 130
Themes 131
Text Tagger Regex component
about 147
about regular expressions 148
configuration properties 231
configuring 151
configuring Reader component 150
graph 147
reference details 231
Text Tagger Whitelist component
about 147
configuration properties 233
configuring 159
configuring Readers 158
graph 155
reference details 233
tag-rules source format 156
Theme extraction, Text Enrichment 131
time zone for data, setting 17
transaction graph
creating 24
steps input file 24
TransactionRunGraph connector 234
configuration properties 236
configuring 27
transactions
configuring Edge for transaction graph 26
configuring Reader component for transaction graph 26
manually committing 29
requirements 22

U

UniversalDataReader component
adding to full index load graph 38
configuring for configuration document input 84
configuring for DDR input 69
configuring for deleting data 194
configuring for full index load 39
configuring for loading view definitions 184
configuring for PDR input 56
configuring for precedence rules 106
configuring for taxonomy loads 125
configuring for Text Enrichment 140
configuring for Text Tagger Regex 150
configuring for Text Tagger Whitelist 158
configuring for transaction graph input 26

V

versions of Web services, verifying 18
view definitions
adding components to graph 184
configuring Reader component 184
configuring Reformat component 187
overview of restoring 183
Visual configuration properties for connectors 238

W

WebServiceClient component
configuring for configuration document loads 92
configuring for GCR 95
configuring for managed attributes 73
configuring for precedence rules 111
configuring for snapshots 199
configuring for standard attributes 64
configuring for stemming files 99
Web services, verifying versions of 18
wizards
about 202
New Project from a Flat File 205
Quick Start 203
Record Store 167
word forms collection files 96
workspace.prm
OUTER_TRANSACTION_ID 13
parameters specific to Endeca 12