# Oracle® Endeca Server

Developer's Guide

Version 7.7.0 • January 2016

ORACLE

# Copyright and disclaimer

# Table of Contents

## Part III: Collections, Record Filters, and Records

# Part VI: Search Features

## Part VII: References

# Preface

Oracle® Endeca Server is a hybrid search-analytical engine that organizes complex and varied data from disparate sources. At the core of Endeca Information Discovery, the unique NoSQL-like data model and in-memory architecture of the Endeca Server create an extremely agile framework for handling complex data combinations, eliminating the need for complex up-front modeling and offering extreme performance at scale. Endeca Server also supports 35 distinct languages.

# About this guide

This guide describes the core features of the Oracle Endeca Server that you can access via applications built with Studio, or with other front-end applications that can communicate with the Oracle Endeca Server.

# Who should use this guide

This guide is intended for developers who are building applications based on the Oracle Endeca Server software and require information about the interfaces to the Oracle Endeca Server, as well as information about the features of the Dgraph process. The features include record and attribute search, record filters, and navigation on refinements.

# Conventions used in this guide

The following conventions are used in this document.

## Typographic conventions

This table describes the typographic conventions used when formatting text in this document.

| Typeface | Meaning |
|---|---|
| **User Interface Elements** | This formatting is used for graphical user interface elements such as pages, dialog boxes, buttons, and fields. |
| `Code Sample` | This formatting is used for sample code phrases within a paragraph. |
| *Variable* | This formatting is used for variable values.<br>For variables within a code sample, the formatting is *`Variable`*. |
| `File Path` | This formatting is used for file names and paths. |

## Symbol conventions

This table describes the symbol conventions used in this document.

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

### Path variable conventions

This table describes the path variable conventions used in this document.

| Path variable | Meaning |
|---|---|
| `$MW_HOME` | Indicates the absolute path to your Oracle Middleware home directory, which is the root directory for your WebLogic installation. |
| `$DOMAIN_HOME` | Indicates the absolute path to your WebLogic domain home directory. For example, if `endeca_server_domain` is the name of your WebLogic domain, then the `$DOMAIN_HOME` value would be the `$MW_HOME/user_projects/domains/endeca_server_domain` directory. |
| `$ENDECA_HOME` | Indicates the absolute path to your Oracle Endeca Server home directory, which is the root directory for your Endeca Server installation. |

## Contacting Oracle Customer Support

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

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

# Part I

## Overview and Concepts

Chapter 1

# Oracle Endeca Server Interfaces

The Oracle Endeca Server is equipped with the set of Web services that provide the interfaces to it. You use the Web services through Integrator ETL (or other data-loading clients) to load the data; and through Studio components (or other front-end applications for querying the data) to query the Oracle Endeca Server and manipulate the query results. You can also use these Web services directly.



## Oracle Endeca Server overview

The Oracle Endeca Server is a hybrid search-analytical engine that organizes complex and varied data from disparate sources. At the core of Endeca Information Discovery, the NoSQL-like data model and in-memory architecture of Endeca Server create an extremely agile framework for handling complex data combinations, eliminating the need for complex up-front data modeling and offering extreme performance at scale.

It is useful to recognize that the term "Endeca Server" may refer to the Endeca Server software package, and to the Endeca Server Java application hosted in the WebLogic Server. Whenever this distinction is needed, the documentation refers to the software package as "the Oracle Endeca Server", and to the Java application as the "Endeca Server Java application".

The Oracle Endeca Server is designed to host multiple Endeca data domains. The Oracle Endeca Server maintains the index of records for the data domain in memory, receives queries, executes them against the stored index, and returns the results.

Once the Endeca Server package is installed in the WebLogic Server, the WebLogic Server starts the Endeca Server Java application. The Endeca Server software exposes almost all of its APIs as SOAP Web services.

A ***data domain*** is a logical unit of data and metadata managed by the Endeca Server. Through its interfaces, the Endeca Server allows for the data loading, configuration, and querying of a data domain. A data domain may impose order on subsets of its data through collections (known in Studio as data sets). A data domain is the largest unit of data over which the Endeca Server allows queries to be expressed. It represents a discrete set of data and includes indexed data records and system records. (Applications wishing to correlate, join, or display data from multiple data domains must do so themselves.)

***The Dgraph*** is the name of the process created in the Oracle Endeca Server, for each data domain. Each Dgraph process handles requests made to the data domain. The Dgraph process of the Oracle Endeca Server uses proprietary data structures and algorithms that allow it to provide real-time responses to client requests. The Dgraph process stores the index created after your source data is ingested. After the index is stored, it

receives client requests via the application tier, queries the data files, and returns the results through the Oracle Endeca Server. The communication between the Endeca Server and the Dgraph process is secure by default.

Once a data domain is created, you only need to use the name of the data domain to manage it. You don't need to know which port the Dgraph processes for the data domain are running on, as the Endeca Server keeps track of that information. This name-only reference to the data domains makes it much easier to enable and disable them and perform other data domain management operations.

The Endeca Server includes a set of commands, available through `endeca-cmd`, with which you create and control data domains. Optionally, you can use the Web services of the Endeca Server for this purpose.

The Oracle Endeca Server is designed to be stateless. This design requires that a complete query be sent to it for each request, for each Endeca data domain hosted in the Endeca Server. The stateless design facilitates the addition of multiple Oracle Endeca Server instances for load balancing and redundancy — any instance of an Oracle Endeca Server cluster hosting a data domain can reply to queries independently of other instances, utilizing a shared data domain index.

Consequently, for each Endeca data domain, configuring additional Dgraph processes as nodes in a data domain cluster increases availability of request processing for the data domain. If a node in the data domain cluster goes down, at least one of the Dgraphs running in the cluster continues to reply to queries.

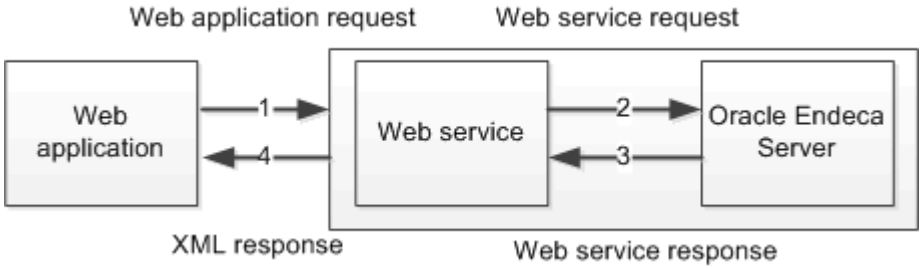
# Data flow

The Oracle Endeca Server communicates with the front-end Web application using its Web services.

A typical solution that utilizes the Oracle Endeca Server consists of the following parts:

- The Oracle Endeca Server, which processes query requests. The Oracle Endeca Server Web services are used for sending and receiving requests.
- A front-end Web application in the form of a set of application modules, which receive client requests and pass them to the Oracle Endeca Server.

The following diagram illustrates the data flow between these parts for a typical front-end application that uses the Oracle Endeca Server:




In this diagram, the following actions take place:

1. A client browser makes a request. The Web application server receives the request and passes it to the Oracle Endeca Server, using its Web services.
2. The Oracle Endeca Server processes the query and returns its results.
3. The Web services retrieve and manipulate the query results and transfer them in XML format to the Web application. The application formats the query results and returns them to the client browser, via the Web application server.

# Full list of Web services

The Oracle Endeca Server is installed with a set of versioned SOAP Web services for loading, configuring, and querying the data, as well as for managing data domains and administering the Endeca Server cluster. These Web services provide the interface to the Oracle Endeca Server.

## Web services available for data domains

Once you install the Oracle Endeca Server and create a data domain in it, you can use the following Web services to send requests:

- Configuration Web Service, `config` (see *Configuration Web Service Interface on page 52*)
- Conversation Web Service, `conversation` (see *Conversation Web Service Interface on page 62*)
- Transaction Web Service, `transaction` (see *Transaction Web Service Interface on page 78*)
- Data Ingest Web Service, `ingest` (documented in the *Oracle Endeca Server Data Loading Guide*)
- Cluster Web Service, `cluster` (documented in the *Oracle Endeca Server Cluster Guide*)
- Manage Web Service, `manage` (documented in the *Oracle Endeca Server Cluster Guide*)

In addition to these listed Web services, additional Web services are available with the Oracle Endeca Server:

- The EQL Parser Web Service, `eql_parser` is used for parsing Endeca Query Language queries and filters. For information, see the *Oracle Endeca Server API References* and the *Oracle Endeca Server EQL Guide*.
- The Entity and Collection Configuration Web Service, `sconfig`, is used to create and manage collections (known in Studio as data sets), and entities (known in Studio as views). For information, see *Entity and Collection Configuration Web Service Interface on page 74*. Also refer to the *Oracle Endeca Information Discovery Studio User's Guide* for information on creating views with Studio.
- Several private Web services also exist in the Oracle Endeca Server. These interfaces are used for internal communication.

**Note:** Each Web service is assigned its own version, consisting of major and minor versions. The supported versions are listed in its WSDL document. If you are planning to use Web service calls directly or use client-side code created with stubs generated from a Web service, ensure that you use a supported version of the Web service. For detailed information on Web service versions, see *Web Service Versioning on page 87*.

In addition to these listed Web services, the Bulk Load Interface is also included. It does not use Web services technology. Together with the Data Ingest Web Service, the Bulk Load Interface loads the records into the Oracle Endeca Server. For more information on the Bulk Load Interface, see the *Oracle Endeca Server Data Loading Guide* (if you are planning to use this interface directly), or the *Oracle Endeca Information Discovery Integrator ETL User's Guide* (if you are planning to use components in Integrator ETL that utilize calls to the Bulk Load Interface).

## Flow for using the Web services

As you build your application, you use these Oracle Endeca Server Web services through various tools. The usage pattern is as follows:

1. Use the Cluster Web Service to configure an Endeca Server Cluster that can host multiple Endeca data domains. For information on the Endeca Server Cluster, see the *Oracle Endeca Server Cluster Guide*.
2. Use the Manage Web Service to create and manage Endeca data domains hosted in the Endeca Server cluster. For information, see the *Oracle Endeca Server Cluster Guide*.
3. Use the Data Ingest Web Service and the Configuration Web Service to load data and configuration into the data domain hosted by the Oracle Endeca Server.
4. Use the Transaction Web Service to group individual requests from other Web services into a single outer transaction. Typically, the Transaction Web Service is useful for sending multiple data loading requests in a single outer transaction.
5. Use the Configuration Web Service to configure the records schema and Oracle Endeca Server features.
6. Use the Conversation Web Service and the Configuration Web Service to send requests and obtain results from the Oracle Endeca Server for your data domain. These results can subsequently be rendered in the front-end application used by the end users.

## How each Web service interacts with the Oracle Endeca Server

Each Web service can be described in the context of how it interacts with the Oracle Endeca Server:

| Web service | Function |
|---|---|
| Data Ingest Web Service | Used to load data into the data domain hosted in the Oracle Endeca Server. It serves as the basis for various batch processes, and is designed for easy integration with ETL tools. |
| Entity and Collection Configuration Web Service | Used to create and manage entities, collections, and filter rules. |
| Configuration Web Service | Supports the process of refining the records schema for the data domain and adjusting your configuration in the development environment. |
| Conversation Web Service | Used to query the index stored by the Oracle Endeca Server for each data domain and to provide summarizations. |
| Transaction Web Service | Controls outer transactions in the Oracle Endeca Server, allowing you to send multiple Web service requests (such as, for data loading), inside a single outer transaction. |
| Manage Web Service | Used to create and manage data domains hosted in the Endeca Server cluster. |
| Cluster Web Service | Used to configure an Endeca Server Cluster that can host multiple data domains. |

### Performance and interaction with the front-end application

Performance of the Endeca Server depends on the number of distinct query requests sent to it through its various web service interfaces, including updating and non-updating query requests, such as those sent from Studio or other front-end clients (if they are used instead of Studio). Performance is not affected by the processing that occurs in the front-end client. For example, the number of front-end application's pages generated to create a query has no impact on performance of underlying queries processed by the Endeca Server.

## About the Oracle Endeca Server API References

The *Oracle Endeca Server API References* represent a collection of automatically generated reference documentation for each of the SOAP Web services and the Bulk Ingest Interface that are packaged with the Oracle Endeca Server.

The *Oracle Endeca Server API References* are generated from the two types of files that describe a Web service:

- A WSDL document
- An XML Schema definition (XSD)

In addition to the WSDL documentation for the packaged Web services, the *Oracle Endeca Server API References* also include the documentation for the Bulk Load Interface.

The *Oracle Endeca Server API References* are located in the `doc` directory of the Oracle Endeca Server installation. They are complemented by the *Oracle Endeca Server Developer's Guide* (this guide) which discusses how to use the Conversation Web Service, the Transaction Web Service, and the Configuration Web Service.

The *Oracle Endeca Server Data Loading Guide* discusses how to use the Data Ingest Web Service and the Bulk Ingest Interface.

Finally, the *Oracle Endeca Server Cluster Guide* discusses how to use the Cluster and Manage Web Services.

## About the Java client examples

The Oracle Endeca Server includes a collection of Java client examples for sending queries to a data domain. Consider using these examples as one possible way to build your own front-end client code.

The examples are based on Java stubs generated with JRF 11.1.1.6.

Even though these examples represent Java classes and methods, they do not represent the supported interfaces. The Web services and the Bulk Load Interface represent the supported interfaces to the Oracle Endeca Server.

**Important:** The Java client examples are not intended to be extensible to real tasks and are not intended to be run in a secure production environment. Use them as demonstrations of how to interact with the generated code and how to create your own client code in Java for sending queries to the Oracle Endeca Server.

Do not use the Java client examples as your reference for the supported interfaces. To learn about the capabilities of each of the supported interfaces, use the Web services themselves, their corresponding WSDL documents, and the documentation generated from them and for the Bulk Load Interface. The automatically-

generated documentation for these interfaces, collectively known as the *Oracle Endeca Server API References*, is included in the installation of the Oracle Endeca Server. In addition, the Oracle Endeca Server documentation provides information about the interface capabilities and describes how to use them.

The Java client examples are installed in the `$ENDECA_HOME/apis/examples` directory of the Oracle Endeca Server package, and include the following top-level directories:

| Directory | Description |
|---|---|
| `generated-sources` | Contains the result of generating stubs using JRF. |
| `src` | Contains sample code for accomplishing basic tasks using these stubs, such as configuring a data domain, adding individual records, and making basic queries.<br>In addition, the `src/tests` directory contains unit tests employing the simple routines in `src/main`, as well as wrappers around those unit tests. |
| `standalone_tests` | Includes a JAR file containing compiled versions of unit test wrappers, and a Perl script for running the tests. |

### Running the examples

All example tests run against a WebLogic instance that has its unencrypted port (7001) open, and that is not configured securely (will not use HTTPS internally).

The Perl script for running the example tests is located in the `/apis/examples/standalone_tests` directory. Before running the script:

- Verify that the Endeca Server is running.
- Download and install Perl, Java, and TestNG packages on your machine, and ensure that your PATH environment variable includes the full paths to Perl and Java packages. The script depends on these packages and logs errors if any of them are not found.
- Verify that the script can locate the security configuration file `jpsconfig.xml`. The script checks for this file in MW_HOME. Alternatively, you can define the file's location by setting the path to it in the environment variable `oracle.security.jps.config`.

If you use the Perl script with no arguments, it issues a list of available options.

To run the Perl script, specify the name of the example test to run and the full path to the TestNG JAR file. The script runs as many tests as you specify on the command line.

# Dgraph configuration documents

You can use the Configuration Web Service interface to load and modify configuration documents.

You can modify the following configuration documents:

- `dimsearch_config`
- `recsearch_config`

- `relrank_strategies`
- `stop_words`
- `thesaurus`

Various features of the Oracle Endeca Server use these documents. See this guide for more information.

**Note:** In addition to letting you modify configuration documents, the Configuration Web Service is also used to modify the system records — Property Description Records, Dimension Description Records, and the Global Configuration Record. For more information, see *Configuration Web Service Interface on page 52*.

Chapter 2

# Oracle Endeca Server Concepts

This section introduces basic concepts associated with the schema of records in the Oracle Endeca Server, and describes how data is structured and configured in the Oracle Endeca Server data model for each hosted data domain.



## About the data model

The data model in the Oracle Endeca Server consists of records and attributes.

- Records are the fundamental units of data.
- Attributes are the fundamental units of the schema. For each attribute, a record may be assigned zero, one, or more attribute values.



### Records

Records are the fundamental units of data in the Oracle Endeca Server. Almost all information that is consumed by the Oracle Endeca Server, including raw data and the data schema, is represented by records.

In the context of applications powered by the Oracle Endeca Server, the following types of records are discussed:

| Record type | Description |
|---|---|
| Source records | Source records represent the data that is input into the application powered by the Oracle Endeca Server, for each data domain. Source records in a variety of formats are supported. |

| Record type | Description |
|---|---|
| Data records | In most applications, you are primarily concerned with data records. Data records are the business records of your data domain that you want to explore using the front-end application. |
| System records | System records represent the records schema in the index that is created by the user for each data domain. System records are created in the Endeca Server using the schema from the primordial records. You use these records for data modeling — changing these records controls the behavior of your records schema and thus affects your data model. System records are listed in the topic *Configurable system records on page 34*. |
| Primordial records | Primordial records are created automatically and used internally by the Endeca Server. They represent the most basic infrastructure of the data model. An overview of them is given in *Primordial records on page 30*. |

## Attributes

An ***attribute*** is the basic unit of a record schema. Assignments from attributes (also known as ***key-value pairs***) describe records in the Oracle Endeca Server.

For a data record, an assignment from an attribute provides information about that record. For example, for a list of book records, an assignment from the Author attribute contains the author of the book record.

Each attribute is identified by a unique name.

Each attribute on a data record is itself represented by a record that describes this attribute. Following the book records example, there is a record that describes the Author attribute. A set of these records that describe attributes forms a schema for your records. This set is known as system records. Each attribute in a record in the schema controls an aspect of the attribute on a data record. For example, an attribute on any data record can be searchable or not. This fact is described by an attribute in the schema record.

The term attribute collectively refers to both standard attributes and managed attributes:

- Standard attributes are described by system records known as Property Description Records (PDRs).
- Managed attributes are described by Property Description Records (PDRs) and Dimension Description Records (DDRs). Each managed attribute is described by one PDR and one DDR.

### Assignments on standard attributes

Records are assigned values from standard attributes. An ***assignment*** indicates that a record has a value from a standard attribute.

A record typically has assignments from multiple standard attributes. For each assigned attribute, the record may have one or more values. An assignment on a standard attribute is known as a ***key-value pair (KVP)***.

Not all standard attributes will have an assignment for every record. For example, for a publisher that sells both books and magazines, the "ISBN number" standard attribute would be assigned for book records, but not assigned (empty) for most magazine records.

Standard attributes may be single-assign or multi-assign:

- A single-assign attribute is an attribute for which each record can have at most one value. For example, for a list of books, the ISBN number would be a single-assign attribute. Each book only has one ISBN number.
- A multi-assign attribute is an attribute for which a single record can have more than one value. For the same list of books, because a single book may have multiple authors, the Author attribute would be a multi-assign attribute.

By default, all standard attributes are single-assign. To make a standard attribute multi-assign, you must update the attribute configuration.

## Primary keys

In the Oracle Endeca Server data model, several types of primary keys (also known as specs) are used, for identifying records, collections, and managed attribute values. This topic provides a summary of each of these primary keys.

## Record spec

For the Oracle Endeca Server to identify each record, it must have an assignment from exactly one primary-key attribute. This assignment is known as a record primary key, or record spec. You can use any attribute as a record primary key as long as the attribute is single-assign and guaranteed to be unique. That is, the PDR (Property Description Record) for the primary-key attribute must have both the `mdex-property_IsSingleAssign` and `mdex-property_IsUnique` attributes set to `true`. As a result, the attribute may be assigned only once in any record and a given attribute value may be assigned to at most one record (that is, no two records in a single Endeca data domain have the same value for this attribute).

The requirement for single assignment and uniqueness is enforced when you are adding initial records to the Endeca data domain using the Data Ingest Web Service or the Bulk Load interface. If you add a new record to the Oracle Endeca Server, it verifies that the record has an assignment for exactly one value from the primary-key attribute.

For more information on how the record spec (record primary key) is used when new records are added to the data domain, see the *Oracle Endeca Server Data Loading Guide*.

## Collection spec

For the Endeca Server to identify each collection, it must have an assignment from exactly one attribute, known as `uniquePropertyKey`. This attribute is known as a unique primary key for a collection (or a collection spec):

| Collection attribute | Description |
|---|---|
| `uniquePropertyKey` | Required. Sets the standard attribute that provides the unique key values for records in the collection. Once a `uniquePropertyKey` is configured for a collection, it cannot be used for any other collection. In addition, this standard attribute must not already be assigned on any record.<br>The PDR for the standard attribute must be created with:<br>• `mdex-property_IsSingleAssign` set to `true`<br>• `mdex-property_IsUnique` set to `true` |

You specify a unique property key for a collection when adding collections. For more information, see *Collection create operations on page 95*.

Note that the collection spec also serves as the record spec during the ingest of the records.

## Managed attribute value spec

For the Endeca Server to identify each managed attribute value, it must have an assignment from the managed attribute value spec, `mdex-dimension-value_Spec`. The managed attribute value spec is added when you create a managed attribute.

## Attribute types

The attribute type identifies the type of data allowed for the standard attribute value (key-value pair).

The Oracle Endeca Server supports the following standard attribute types:

| Attribute type | Description |
|---|---|
| `mdex:string` | XML-valid character strings. |
| `mdex:int` | A 32-bit signed integer. `mdex:int` values accepted by the Oracle Endeca Server on all platforms can be up to the value of 2,147,483,647. |
| `mdex:long` | A 64-bit signed integer. `mdex:long` values accepted by the Oracle Endeca Server on all platforms can be up to the value of 9,223,372,036,854,775,807. |
| `mdex:double` | A floating point value. |
| `mdex:time` | Represents the hour and minutes of an instance of time, with the optional specification of fractional seconds. The time value can be specified as a universal (UTC) date time or as a local time plus a UTC time zone offset. |

| Attribute type | Description |
|---|---|
| `mdex:dateTime` | Represents the year, month, day, hour, minute, and seconds of a time point, with the optional specification of fractional seconds. The dateTime value can be specified as a universal (UTC) date time or as a local time plus a UTC time zone offset. |
| `mdex:duration` | Represents a duration of the days, hours, and minutes of an instance of time. |
| `mdex:boolean` | A Boolean. Valid Boolean values are `true` (or `1`, which is a synonym for `true`) and `false` (or `0`, which is a synonym for `false`). |
| `mdex:geocode` | A latitude and longitude pair. The latitude and longitude are both double-precision floating-point values, in units of degrees. |

## XML representation of records and attributes

In XML, each record is represented as a collection of attribute value assignments (key-value pairs).

In all of the Oracle Endeca Server Web service interfaces, a record is represented in XML as a record element. The record element contains attribute elements (these attributes should not be confused with the term "attribute" used in the XML standard set of terms). Each attribute element contains the attribute values for the specified attribute.

If a record does not have a value for an attribute, the attribute is not included for that record.

If a record has multiple values for an attribute, there is a separate attribute element for each value.

The following XML represents a single data record with three standard attributes (`ProductID`, `BikeType`, and `Color`):

```
<Record>
  <attribute name="ProductID" type="mdex:int">12345</attribute>
  <attribute name="BikeType" type="mdex:string">Road Bikes</attribute>
  <attribute name="Color" type="mdex:string">Red</attribute>
</Record>
```

## Examples of records and standard attributes

The following examples of records demonstrate different configurations of standard attributes and their values (key-value pairs).

### About these examples

In the examples, each row in the table represents a single record, in this case, a bicycle. The column headings are standard attributes, and each cell contains a standard attribute value (key-value pair).

### Example 1: all records have a single assignment from each attribute

In this example:

- The ProductID attribute is the primary key, and is therefore both unique and single-assign. Each record has exactly one assignment on the ProductID attribute, and the ProductID attribute value for a given record is unique across the data set.
- All other attributes are single-assign, which means that no records have multiple assignments from a given attribute.
- Every record has an assignment for every attribute.

| Name | Bike Type | ProductID | Size Range | Color | Number Sold | Price |
|---|---|---|---|---|---|---|
| Road-450 | Road Bikes | 4038 | 42-46 CM | Red | 171 | 1457.99 |
| Road-550-W | Road Bikes | 5213 | 38-40 CM | Yellow | 455 | 1000.48 |
| Touring-1000 | Touring Bikes | 8765 | 54-58 CM | Blue | 117 | 2384.07 |
| Touring-3000 | Touring Bikes | 4035 | 48-52 CM | Yellow | 221 | 742.35 |
| Mountain-300 | Mountain Bikes | 3421 | 38-40 CM | Black | 223 | 1079.99 |
| Mountain-500 | Mountain Bikes | 4821 | 38-40 CM | Silver | 176 | 564.99 |

The XML representation of the Road-450 record may look similar to the following example:

```
<Record>
  <attribute name="Name" type="mdex:string">Road-450</attribute>
  <attribute name="ProductID" type="mdex:int">4038</attribute>
  <attribute name="BikeType" type="mdex:string">Road Bikes</attribute>
  <attribute name="SizeRange" type="mdex:string">42-46 CM</attribute>
  <attribute name="Color" type="mdex:string">Red</attribute>
  <attribute name="NumSold" type="mdex:int">171</attribute>
  <attribute name="Price" type="mdex:double">1457.99</attribute>
</Record>
```

Notice the primary key attribute, which in this case is the `ProductID` attribute. This primary key attribute is used by the Oracle Endeca Server to uniquely identify this record. At the data loading stage, you decide which of your standard attributes is going to be the primary key attribute.

### Example 2: records with no assignments or multiple assignments on an attribute

This example uses the same data as the previous example, but adds a Review Score attribute. For the Review Score attribute, some records have multiple assignments and some have no assignments.

For example, the Road-450 record has multiple review scores and the Touring-3000 record has no review scores.

| Name | Bike Type | ProductID | Size Range | Color | Review Score | Price |
|---|---|---|---|---|---|---|
| Road-450 | Road Bikes | 4038 | 42-46 CM | Red | 35, 45, 60 | 1457.99 |
| Road-550-W | Road Bikes | 5213 | 38-40 CM | Yellow | 80, 82 | 1000.48 |
| Touring-3000 | Touring Bikes | 4035 | 48-52 CM | Yellow | | 742.35 |
| Mountain-500 | Mountain Bikes | 4821 | 38-40 CM | Silver | 76 | 564.99 |

The XML representation of the Road-450 and Touring-3000 bikes may look similar to the following example:

```
<Record>
  <attribute name="Name" type="mdex:string">Road-450</attribute>
  <attribute name="ProductID" type="mdex:int">4038</attribute>
  <attribute name="BikeType" type="mdex:string">Road Bikes</attribute>
  <attribute name="SizeRange" type="mdex:string">42-46 CM</attribute>
  <attribute name="Color" type="mdex:string">Red</attribute>
  <attribute name="ReviewScore" type="mdex:int">35</attribute>
  <attribute name="ReviewScore" type="mdex:int">45</attribute>
  <attribute name="ReviewScore" type="mdex:int">60</attribute>
  <attribute name="Price" type="mdex:double">1457.99</attribute>
</Record>
<Record>
  <attribute name="Name" type="mdex:string">Touring-3000</attribute>
  <attribute name="ProductID" type="mdex:int">4035</attribute>
  <attribute name="BikeType" type="mdex:string">Mountain Bikes</attribute>
  <attribute name="SizeRange" type="mdex:string">48-52 CM</attribute>
  <attribute name="Color" type="mdex:string">Yellow</attribute>
  <attribute name="Price" type="mdex:double">742.35</attribute>
</Record>
```

The XML for the Road-450 record contains three `ReviewScore` elements, one for each score. Because the Touring-3000 record does not have any review scores, it does not include a `ReviewScore` element.

## Managed attributes

***Managed attributes*** are similar to standard attributes in that they describe the records in your data set.

Unlike standard attributes, (which only provide a way to assign values to records in your data set), managed attributes allow you to capture additional characteristics that may be present in your data. Once captured and loaded into a running Dgraph, these characteristics become part of that data domain.

Managed attributes allow you, as a data architect, to capture the following characteristics of your records:

- **A set of predefined allowed values**. Some attributes on your data records may have a requirement to have assignments only from a predefined set of allowed values. For example, an attribute `currency` may have a predefined set of values (`dollars` and `euros`). A managed attribute allows you to define a set of specific values that are allowed on a standard attribute.
- **Hierarchy**. Some attributes on your data records may benefit from being organized in a hierarchy. For example, an attribute representing a ProductCategory type of Clothing may have different types of clothing items underneath it, each represented by a standard attribute (such as Caps, Gloves, Jerseys, and so on). A managed attribute allows you to define the hierarchy of standard attributes.

- **Additional metadata on attribute values**. Finally, your source data records may have attribute values that could include additional metadata, such as text descriptions. Managed attributes allow you to capture these additional metadata on attribute values.

Managed attributes are often used to support hierarchical navigation. In other words, associating a managed attribute with a standard attribute enables hierarchical navigation of records based on the standard attribute values. For example, you can navigate a collection of books using the Library of Congress Classification standard attribute, and refine by **Literature > American > 19th century**.

(Note that while managed attributes can capture the hierarchy of your attributes, they are not required to contain hierarchy information.) When you create a managed attribute whose purpose is to represent a hierarchy, you load a taxonomy definition that enumerates a hierarchy where each standard attribute value (in a key-value pair for the standard attribute) is a node in the hierarchy (called a ***managed attribute value, or mval***).

Managed attributes are described by system records — PDRs and DDRs.

# Primordial records

Primordial records are created automatically and should not be modified by the user.

When an Endeca Server data domain is created, a group of system records, called primordial records, is created automatically within the Dgraph. These primordial records are used internally by the Dgraph and their configuration should not be changed by the user. Changing their configuration may cause parts of the system to fail to work or to work incorrectly.

The purpose of these primordial records is to define system properties that will, in turn, be used to define user-created system records, such as PDRs and CDRs. For example, the `mdex-property_Type` record defines the configuration for the `mdex-property_Type` schema attribute, which is one of the various properties that define a PDR for a standard attribute.

The primordial records in the system are:

| Primordial record | Purpose |
|---|---|
| **CDR (defines the schema for collection):**<br>`mdex-collection_Key`<br>`mdex-collection_PropertyKey`<br>`mdex-collection_UniquePropertyKey`<br>`system-collection_Description`<br>`system-collection_DisplayName` | Define properties of a Collection Description record (CDR). A CDR, in turn, defines the characteristics of collections, such as their spec. |

| Primordial record | Purpose |
|---|---|
| **GCR (defines global configuration):**<br>`mdex-config_Key`<br>`mdex-config_MergePolicy`<br>`mdex-config_SearchChars`<br>`mdex-config_SpellingDValMaxWordLength`<br>`mdex-config_SpellingDValMinWordLength`<br>`mdex-config_SpellingDValMinWordOccur`<br>`mdex-config_SpellingRecordMaxWordLength`<br>`mdex-config_SpellingRecordMinWordLength`<br>`mdex-config_SpellingRecordMinWordOccur`<br>`mdex-config_SystemRecordVersion` | Define properties of a Global Configuration Record (GCR). A GCR, in turn, defines global configuration parameters, such as spelling configuration. |
| **DDR (together with a PDR, defines a managed attribute):**<br>`mdex-dimension_Key`<br>`mdex-dimension-value_Spec`<br>`mdex-dimension-value_Parent`<br>`mdex-dimension_EnableRefinements`<br>`mdex-dimension_IsDimensionSearchHierarchical`<br>`mdex-dimension_IsRecordSearchHierarchical`<br>`system-dimension_DefaultValue` | Define a Dimension Description Record (DDR). A DDR, together with a PDR, in turn, defines properties for managed attributes. For example, a DDR defines a primary key for the managed attribute, its spec and its parent, and whether it should enable refinements to be displayed. |
| **MAVDR (defines a managed attribute value):**<br>`mdex-dimension-value_Dimension`<br>`mdex-dimension-value_Name`<br>`mdex-dimension-value_Rank`<br>`mdex-dimension-value_Synonyms` | Define properties of a Managed Attribute Value Description Record (MAVDR). A MAVDR, in turn, defines characteristics of managed attribute values, such as their display name, as well as rank and synonyms (if they are present). |
| **PRDR (defines a precedence rule):**<br>`mdex-precedenceRule_Key`<br>`mdex-precedenceRule_IsLeafTrigger`<br>`mdex-precedenceRule_TargetAttributeKey`<br>`mdex-precedenceRule_TriggerAttributeKey`<br>`mdex-precedenceRule_TriggerAttributeValue` | Define properties of a description record that defines precedence rules. This record, in turn, defines characteristics of precedence rules, such as the spec for the precedence rule, its trigger's spec and value, its target spec and value. |
| `mdex-property_ForeignKey` | Not used (reserved for future use). |

| Primordial record | Purpose |
|---|---|
| **PDR (defines a standard attribute and a managed attribute):**<br>`mdex-property_Key`<br>`mdex-property_DisplayName`<br>`mdex-property_IsPropertyValueSearchable`<br>`mdex-property_IsSingleAssign`<br>`mdex-property_IsTextSearchable`<br>`mdex-property_IsUnique`<br>`mdex-property_Language`<br>`mdex-property_TextSearchAllowsWildcards`<br>`mdex-property_Type`<br>`system-navigation_Select`<br>`system-navigation_ShowRecordCounts`<br>`system-navigation_Sorting`<br>`system-property_DefaultValue` | Define properties of a Property Description Record (PDR). A PDR represents characteristics for standard attributes. Note that:<br>• PDRs, in addition to defining standard attributes, also define managed attributes (together with DDRs).<br>• `mdex-property_Key` is also used to add standard attributes to attribute groups. |
| **EDR (defines an entity):**<br>`mdex-semanticEntity_Key`<br>`mdex-semanticEntity_Attributes`<br>`mdex-semanticEntity_Definition`<br>`mdex-semanticEntity_Description`<br>`mdex-semanticEntity_DisplayName`<br>`mdex-semanticEntity_Groups`<br>`mdex-semanticEntity_IsActive`<br>`mdex-semanticEntity_Metrics`<br>`system-group_DynamicContent`<br>`system-group_DynamicOrder` | Define properties of an entity description record, which, in turn, defines characteristics for entity records. |

| Primordial record | Purpose |
|---|---|
| **FRDR (defines a filter rule):**<br>`system-filterRule_Key`<br>`system-filterRule_DisplayName`<br>`system-filterRule_IsActive`<br>`system-filterRule_SourceCollectionKey`<br>`system-filterRule_SourcePropertyKey`<br>`system-filterRule_TargetCollectionKey`<br>`system-filterRule_TargetPropertyKey` | Define properties of a Filter Rule Description Record (FRDR), which in turn defines filter rules. |
| **GDR (defines an attribute group configuration):**<br>`system-group_Key`<br>`system-group_DisplayName` | Define properties of a Group Description Record (GDR), which, in turn defines attribute group configuration. |
| **GMDR (defines attribute group membership):**<br>`system-group-membership_Key`<br>`system-group-membership_GroupKey`<br>`system-group-membership_Position`<br>`system-group-membership_PropertyKey` | Define properties of a Group Membership Description Record (GMDR), which in turn defines attribute group members. |
| `system-navigation_RecordSource` | Used in the definition of various types of primordial records. |
| `system-parentPipeline*`<br>`system-pipeline*` | Define the various schema records for the Data Enrichment module. The Data Enrichment module is packaged with the Endeca Server and is used with Studio. |

## Listing primordial records

You can list the primordial records with:

- `RecordKind` filter with the `nondata` value (see *Filtering data and non-data records on page 136*)
- `listProperties` operation of the Configuration Web Service (see *Other list attribute methods on page 194*)
- `PropertyListConfig` summarization type (see *Obtaining a list of available attributes on page 193*)

# Configurable system records

Some system records that store configuration information can be modified by the user.

The next three topics describe how you can configure the following system records, all of which are user-created except for the GCR:

- **Property Description Record (PDR)** for a standard attribute. PDRs are used to define the format and behavior of standard attributes and managed attributes.
- **Dimension Description Record (DDR)** for a managed attribute. DDRs are used to define managed attributes and thus, among other characteristics, enable the creation of hierarchical attribute values.
- **Global Configuration Record (GCR)**, used to control various aspects of the global configuration. The GCR is automatically created when the data domain is created, but it can be changed afterwards by the user.

In addition, other configurable system record types exist. They are also described in this guide:

- **Managed Attribute Value Description Records (MAVDR)** for a user-created managed attribute value. For operations to create managed attribute values, see *Operations for Managed Attribute Values on page 56*.
- **Collection Description Record (CDR)** for a collection. For the operations to create and modify collections, see *Collection operations on page 94*.
- **Filter Rule Description Record (FRDR)** for a filter rule. For the operations to create and modify filter rules, see *Filter rule operations on page 112*.
- **Precedence Rule Description Record (PRDR)** for a precedence rule. For the operations to create and modify precedence rules, see *Precedence Rules on page 224*.
- **Entity Description Record (EDR)** for an entity. For the operations to create and modify entities, see *Working with Entities on page 231*.
- **Group Description Record (GDR)** for the configuration of an attribute group. For the `updateGroupConfigs` operation to update the config, see *Updating the group configuration on page 217*.
- **Group Membership Description Record (GMDR)** for the membership of an attribute group. For the operations to create and modify the attribute group membership rules, see *Attribute Groups on page 209*.

To avoid naming collisions with customer-created records and attributes, the keys for system records use reserved prefixes, such as `mdex-property`.

*Property Description Record (PDR)*

*Dimension Description Record (DDR)*

*Global Configuration Record (GCR)*

*Updates to schema and configuration*

# Property Description Record (PDR)

A ***Property Description Record (PDR)*** is a system record that defines a record for a standard attribute in an Endeca data domain.

## About PDRs

The Oracle Endeca Server uses a PDR to store metadata about the standard attribute, and must have a PDR created in order to build a schema for your data records. In addition, to create a managed attribute, both one PDR and one DDR are required.

As records, PDRs themselves have required attributes, and can also have arbitrary, user-defined attributes.

For each standard attribute, the attributes in the associated PDR define the attribute's characteristics, including:

- Name and type
- Display name
- Language
- Configuration parameters. For example, whether an attribute is searchable.
- Navigability settings. For example, whether to show record counts for available refinements, whether to enable multi-select, and how to sort refinements.

## Creating and updating PDRs

When an Endeca data domain acquires a new record, it stores it and constructs a PDR for any attributes that it finds in the record. Updating a PDR immediately changes the navigation behavior of the Oracle Endeca Server. To create or change a PDR, you can use the Data Ingest Web Service, or Integrator ETL.

## Required schema attributes of a PDR

PDRs have the attributes in the following table. (The word "required" means that if you do not specify them for the PDR, their default values are used). Only the key is the required attribute when creating a PDR. If you do not specify other attributes, their default values are used.

| Schema attribute | Type | Description |
|---|---|---|
| `mdex-property_Key` | `string` | The key name of the standard attribute. This attribute is mandatory.<br>The key name must be an NCName. The NCName format is defined in the W3C document Namespaces in XML 1.0 (Second Edition), located at this URL: *http://www.w3.org/TR/REC-xml-names/#NT-NCName* Although hyphens are valid in an NCName, it is recommended to use key names without hyphens. The reason is that if a hyphenated standard attribute is used in an EQL statement, it must be enclosed in quotes.<br>You can modify the name of the standard attribute. No restrictions on these changes exist. |
| `mdex-property_DisplayName` | `string` | The name of the standard attribute in an easy-to-understand format. The display name can use a non-NCName format.<br>You can modify the display name of an attribute. No restrictions on these changes exist. |

| Schema attribute | Type | Description |
|---|---|---|
| `mdex-property_Type` | `string` | The data type of the standard attribute. The possible values are:<br>• `mdex:boolean`<br>• `mdex:dateTime`<br>• `mdex:duration`<br>• `mdex:double`<br>• `mdex:geocode`<br>• `mdex:int`<br>• `mdex:long`<br>• `mdex:string`<br>• `mdex:time`<br>The default is `mdex:string`.<br>The data type cannot be modified. |
| `mdex-property_Language` | `string` | The language of the standard attribute (set as an RFC-3066 language code).<br>The language ID is either the special string `unknown` (the default if not changed) or (if changed) the RFC-3066 language code set by the Configuration Web Service's `setPropertyDefaultLanguage` operation.<br>This schema attribute can be modified, but changing its value has a performance cost. |

| Schema attribute | Type | Description |
|---|---|---|
| `mdex-property_IsSingleAssign` | `boolean` | If set to `true` (the default), each record can have at most one value for an attribute assignment (i.e., the attribute is single assign).<br>If set to `false`, each record may have more than one value for an attribute assignment (i.e., the attribute is multi-assign).<br>The setting can be modified as follows:<br>• A single-assign setting can be changed to a multi-assign setting only if the PDR is non-unique (i.e., `mdex-property_IsUnique` is set to `false`).<br>• A multi-assign setting can be changed to a single-assign setting only if all assignments of this standard attribute are single assign. If at least one record has multi-assignments of this standard attribute, an attempted change will fail. |
| `mdex-property_IsUnique` | `boolean` | If set to `true`, no two records can have the same value for the attribute. Note that a setting of `true` requires that `mdex-property_IsSingleAssign` must also be set to `true`.<br>If set to `false` (the default), multiple records can have the same value for this attribute.<br>You can change the setting (from `true` to `false` or vice-versa) only if no assignments for this attribute have been made (i.e., no record can have an assignment for this attribute). |

| Schema attribute | Type | Description |
|---|---|---|
| `mdex-property_IsTextSearchable` | `boolean` | If set to `true`, then the standard attribute is enabled for text search.<br>If set to `false`, the standard attribute does not support text search.<br>The default is `false`.<br>This schema attribute can be changed only for the standard attributes of type string.<br>Making this change on an attribute that has many assignments on records has a performance cost due to re-indexing. |
| `mdex-property_TextSearchAllowsWildcards` | `boolean` | If set to `true`, then wildcard search is enabled for this standard attribute.<br>If set to `false`, then wildcard search is not enabled.<br>If this is set to `true`, then `mdex-property_IsTextSearchable` must be set to `true`.<br>The default is `false`.<br>Changing the value for this attribute has a performance cost. A restriction for modifying it is that `mdex-property_IsTextSearchable` must also be set to `true`. |
| `mdex-property_IsPropertyValueSearchable` | `boolean` | If set to `true`, the standard attribute is enabled for value search. Note that only attributes of type `mdex:string` can be enabled for value search.<br>If set to `false`, the attribute is not value-searchable.<br>The default is `true`.<br>This schema attribute can be changed only for the standard attributes of type string. Modifying the value for this schema attribute has a performance cost.<br>This schema attribute does not apply for managed attributes, for which value search is always enabled and cannot be disabled. |

| Schema attribute | Type | Description |
|---|---|---|
| `system-navigation_Select` | `string` | Configures the multi-select feature for a standard attribute. The values of this attribute supply defaults for query parameters, which you can override. The allowed values are:<br>• `single`. Users can select only one refinement from this attribute.<br>• `multi-and`. Users can select multiple refinements from the attribute. The returned records must have assignments from all of the selected refinements (from A AND B). Selecting this value only makes sense for those attributes that are multi-assign.<br>• `multi-or`. Users can select multiple refinements from this attribute. The returned records must have assignments from at least one of the selected refinements (from A OR B).<br>The default is `single`.<br>No restrictions on modifying the values for this schema attribute exist. |
| `system-navigation_Sorting` | `string` | The order in which to display refinements in the navigation menu. The allowed values are:<br>• `lexical` sorts refinements alphabetically or by number.<br>• `record-count` sorts refinements in descending order, by the number of records available for each refinement.<br>The default is `record-count`.<br>No restrictions on modifying the values for this schema attribute exist. |

| Schema attribute | Type | Description |
|---|---|---|
| `system-navigation_ShowRecordCounts` | `boolean` | Whether to show record counts for a refinement.<br>If set to `true`, the record counts are shown.<br>If set to `false`, the record counts are not shown.<br>The default is `true`.<br>No restrictions on modifying the values for this schema attribute exist. |

### User-defined schema attributes of a PDR

You can add assignments on other, user-defined attributes to PDRs to display various aspects of how your data records are organized. Note that the names of these attributes must not begin with `mdex`, or `system`.

## Dimension Description Record (DDR)

A ***Dimension Description Record (DDR)***, together with a PDR, defines a managed attribute.

### About DDRs

The Dimension Description Record has the same name as the associated standard attribute. It is used to enable the creation of hierarchical standard attribute values and managed attribute values, to provide a list of predefined allowed values, and also as a placeholder for metadata on the attribute values.

### Required schema attributes of a DDR

A Dimension Description Record has the following required schema attributes. (The word "required" means that if you do not specify them for the DDR, their default values are used).

| Schema attributes | Type | Description |
|---|---|---|
| `mdex-dimension_Key` | `string` | The primary key (spec) of the managed attribute. It cannot be modified. This attribute is mandatory and must be user-specified. |

| Schema attributes | Type | Description |
|---|---|---|
| `mdex-dimension_EnableRefinements` | `boolean` | If set to `true`, then refinements are displayed.<br>If set to `false`, refinements are not displayed. In other words, the managed attribute is hidden.<br>The default is `true`.<br>You can modify whether to enable the display of refinements on managed attributes. No restrictions on this change exist. |
| `mdex-dimension-value_Spec` | `string` | The primary key (spec) of the managed attribute value.<br>It cannot be modified.<br>If, when adding a managed attribute, you do not specify this attribute for the DDR, the default value is used, in this format: `mdex-dimension_<key>_Spec`, where <*key*> is the key of the attribute you are adding. |
| `mdex-dimension-value_Parent` | `string` | The key of the parent managed attribute value.<br>It cannot be modified.<br>If, when adding a dimension, you do not specify this attribute for the DDR, the default value is used, in this format: `mdex-dimension_<key>_Parent`, where <*key*> is the key of the attribute you are adding. |

| Schema attributes | Type | Description |
|---|---|---|
| `mdex-dimension_IsDimensionSearchHierarchical` | `boolean` | If set to `true`, then during value searches, the search matches both the assigned values and the ancestors of those values.<br>If set to `false`, then the search matches only the assigned values.<br>The default is `false`.<br>The hierarchy setting for attribute search can be modified only before loading records. |
| `mdex-dimension_IsRecordSearchHierarchical` | `boolean` | If set to `true`, then during record searches, the search matches records with both the assigned values and the ancestors of those values.<br>If set to `false`, then the search only matches records with the assigned values.<br>The default is `false`.<br>The hierarchy setting for record search can be modified only before loading records. |

### User-defined schema attributes of a DDR

You can add assignments on other, user-defined attributes to DDRs to display various aspects of how your data records are organized. Note that the names of these attributes must not begin with `mdex`, or `system`.

## Global Configuration Record (GCR)

The ***Global Configuration Record*** (GCR) is a single record used to identify and store global configuration information for a specific Endeca data domain.

### Definition

The Global Configuration Record is created automatically in the Endeca data domain, and can be modified if needed. This information persists if you restart that data domain.

During record updates, the Dgraph validation process validates the configuration of the Global Configuration Record, and returns errors if its requirements are not met. The requirements are as follows:

- The `mdex-config_Key` attribute must be unique and single-assign. The value must be `global`.

- The Global Configuration Record must contain valid allowable values for all of its attributes. None of its attributes can be omitted.
- The Global Configuration Record cannot have any arbitrary, user-defined attributes.

The Global Configuration Record controls the following areas of the Endeca data domain configuration:

| Area of global configuration | What you can do... |
|---|---|
| Wildcard search enablement | Specify whether wildcard search should be enabled or disabled for value search in the Endeca data domain. By default, it is disabled. |
| Search characters | List which characters you want to identify as search characters for queries. |
| Merge policy | Optionally, change the policy that the Endeca data domain uses in the background to merge its generations of data files. The default policy – `balanced` – is recommended, and is optimized for best performance. |
| Spelling correction settings | Control which words are eligible for the spelling dictionary by specifying the following parameters:<br>• Minimum word occurrence<br>• Minimum word length<br>• Maximum word length<br>  **Note:** If you change the spelling settings in the Global Configuration Record, you must run the `updateSpellingDictionaries` operation of the Data Ingest Web Service in order for them to take effect. |

## Modifying the settings in the Global Configuration Record

To change the Global Configuration Record settings, use the Configuration Web Service's `putGlobalConfigRecord` operation or one of the data ingest interfaces (such as the Data Ingest Web Service).

## Required attributes of the GCR

The Global Configuration Record has required attributes, but it cannot have arbitrary, user-defined attributes. The required attributes are:

| Attribute | Type | Description |
|---|---|---|
| `mdex-config_Key` | `String` | The only value for this attribute is `global`.<br>This attribute is unique and single-assign. It cannot be modified. |

| Attribute | Type | Description |
|---|---|---|
| mdex-config_EnableValueSearchWildcard | Boolean | If set to true, then wildcard search is enabled for value search.<br>If set to false, then wildcard search is disabled.<br>The default value is false.<br>While you can change these values without restrictions, changing them has a performance cost. |
| mdex-config_MergePolicy | String | The allowed values are balanced or aggressive.<br>The default is balanced.<br>The value for this attribute can be modified if needed. |
| mdex-config_SearchChars | String | The characters to use as search characters in the Oracle Endeca Server.<br>The allowed values are strings that are listed sequentially and are not separated by commas or spaces.<br>Each string is a search character.<br>While you can values for this attribute without restrictions, changing them has a performance cost. |
| mdex-config_SystemRecordVersion | String | The version of the system records in the Oracle Endeca Server.<br>This attribute is used by the Oracle Endeca Server and should not be modified. |

| Attribute | Type | Description |
|---|---|---|
| `mdex-config_SpellingRecordMinWordOccur` | `Int` | The minimum number of times a word must occur in a standard attribute value (record assignment on a standard attribute, in a key value pair) for it to affect the configuration for spelling correction.<br>The default value is 4.<br>You can modify various spelling settings. The values for spelling must not be negative. |
| `mdex-config_SpellingRecordMinWordLength` | `Int` | The minimum number of characters that a word can contain in a standard attribute value for it to affect the configuration for spelling correction.<br>The default value is 3.<br>You can modify various spelling settings. The values for spelling must not be negative. |
| `mdex-config_SpellingRecordMaxWordLength` | `Int` | The maximum number of characters that a word can contain in a standard attribute value for it to affect the spelling correction.<br>The default value is 16.<br>You can modify various spelling settings. The values for spelling must not be negative. |
| `mdex-config_SpellingDValMinWordOccur` | `Int` | The minimum number of times a word must occur in a managed attribute value for it to affect the spelling correction.<br>The default value is 1.<br>You can modify various spelling settings. The values for spelling must not be negative. |

| Attribute | Type | Description |
|---|---|---|
| mdex-config_SpellingDValMinWordLength | Int | The minimum number of characters that a word must contain in a managed attribute value for it to affect the spelling correction.<br>The default value is 3.<br>You can modify various spelling settings. The values for spelling must not be negative. |
| mdex-config_SpellingDValMaxWordLength | Int | The maximum number of characters that a word may contain in a managed attribute value for it to affect the spelling correction.<br>The default value is 16.<br>You can modify various spelling settings. The values for spelling must not be negative. |

## Updates to schema and configuration

Some parts of your existing data domain's schema and configuration can be updated on a running data domain without performance impact, while other modifications may have performance implications. Additionally, a few aspects of your data domain's configuration and schema can only be modified before the data records are loaded, or cannot be modified at all. This topic addresses these categories, to help you make decisions about loading data and configuring your data domains.

This topic does not provide an exhaustive list of which properties on system records (or other aspects of configuration) can be modified and under which conditions. Instead, it explains the basic principles that help answer the following questions:

- When are updates on a running data domain supported? Are there any restrictions, such as, are modifications allowed only before data records are loaded?
- What is the impact (including performance impact) of an update and when is it reflected in the index? For example, does this modification cause delays in obtaining query results while the Endeca Server is re-indexing the data domain?

To answer these questions, it helps to understand how the Endeca Server maintains the knowledge of your data domain's records configuration. The following aspects control the various characteristics of your records loaded into the Endeca Server:

- **Index**. Once you load source records into the Endeca Server, it creates an internal index. At the initial indexing time, the source records utilize primordial records and system records. Together, these records form the schema for the loaded source records — each primordial record serves as the basis for a system record, and each attribute in a system record represents a particular configuration setting. Thus, once the

index is created based on source records, it already contains inside it settings that control various aspects of these records.

Changing the items that affect the schema causes the Endeca Server to re-index its records. Examples of modifications that cause records re-indexing include modifying which attributes in your records are searchable, modifying the language value, and making changes to search characters. If these settings are changed, the Endeca Server must recreate the index for the data domain.

- **Additional configuration settings**. Certain aspects of configuration do not affect the index, but may have an impact on how a query is processed, such as whether a record's attribute returned in the results has a display name (in addition to its internal value). Changes that fall into this category do not cause re-indexing and thus do not have performance costs. These changes affect subsequent queries — these are queries processed after the change was submitted to the Endeca Server.

1. **Changes that have performance costs**.

   This table includes some examples of items whose modifications have performance costs associated with a partial or complete re-indexing of loaded records:

| Item | Comments and restrictions on modifying (if exist) |
|---|---|
| `mdex-property_IsTextSearchable` | Specifies whether values from this attribute are searchable. No restrictions on these changes exist, except that they are only allowed for attributes of type `string`. Making such a change on an attribute that has many assignments on records has a performance cost due to re-indexing. |
| `mdex-property_TextSearchAllowsWildcards` | Affects whether wildcards are allowed in text search on attributes. Has a performance cost. A restriction for modifying this setting is that `mdex-property_IsTextSearchable` must also be set to `true`. |
| `mdex-property_IsPropertyValueSearchable` | Affects value search on standard attributes. Has a performance cost. Changing the value for this attribute is only allowed on attributes of type string. |
| `mdex-property_Language` | Specifies the language ID on an attribute. Has a performance cost. Changes are only allowed for supported language codes (see *Supported languages on page 154*). No other restrictions on these changes exist. |
| `mdex-config_EnableValueSearchWildcard` | This setting is part of the Global Configuration Record. It affects whether wildcards are allowed on value search. Has a performance cost. No restrictions on these changes exist. |
| `mdex-config_SearchChars` | Specifies search characters. Has a performance cost. No restrictions on these changes exist. |

2. **Changes that have no performance costs**.

   Examples of modifications that do not have performance costs include:

| Item | Restrictions on modifying (if exist) |
|---|---|
| `system-navigation_Sorting`<br>`system-navigation_ShowRecordCounts`<br>`system-navigation_Select` | You can modify how the records are sorted, or whether record counts are displayed, or how refinements can be selected (one, or many). No restrictions on these changes exist. |
| `mdex-property_DisplayName`<br>`mdex-property_Key` | You can modify the display name of an attribute or the name of the standard attribute. No restrictions on these changes exist. |
| `mdex-dimension_EnableRefinements` | You can modify whether to enable the display of refinements on managed attributes. No restrictions on this change exist. |
| `mdex-config_Spelling*` | You can modify various spelling settings. The values for spelling must not be negative. For the changes to take effect, you must run the `updateSpellingDictionaries` operation to update the spelling dictionaries. |
| `mdex-precedenceRule_*` | You can modify various precedence rules settings. No restrictions on these changes exist. |
| `mdex-dimension-value_Name`<br>`mdex-dimension-value_Rank` | You can modify the display name and the ranking of the managed attribute value records. No restrictions on these changes exist. (Note, for managed attribute value records, these are the only two items that you can modify. All other characteristics of the managed attribute value records — their spec and their parent managed attribute value, as well as the associated managed attribute, and their synonyms — cannot be modified at all. Instead, you need to delete and add a new set of managed attribute values.) |

3. **Exceptions**. A few exceptions exist to the first two categories. These are various aspects of the schema or configuration that for various reasons can either be modified only before assignments on the system

records exist, cannot be changed at all, or cannot be deleted once they are added. These exceptions include the following (this list is not guaranteed to be exhaustive):

| Item | Restrictions on modifying (if exist) |
|---|---|
| `mdex-dimension_IsDimensionSearchHierarchical`<br>`mdex-dimension_IsRecordSearchHierarchical` | The hierarchy setting for record search and attribute search can be modified only before loading records. |
| `mdex-dimension-value_Spec` and `mdex-dimension-value_Parent` , which belong to the DDR of the associated managed attribute.<br>`mdex-dimension-value_Dimension` and `mdex-dimension-value_Synonyms` , which belong to the MAVDR (Managed Attribute Value Description Record). | These items, which together define the characteristics of the managed attribute value records, cannot be modified once they are added. If you delete the associated managed attribute, you can delete them too. |
| `mdex-property_IsUnique` | Affects whether an assignment on this attribute is unique across the entire corpus. You can change the setting (from `true` to `false` or vice-versa) only if no assignments for this attribute have been made (that is, if no record can have an assignment for this attribute). Note that a unique attribute must also be single-assign (that is, if you set it to `true`, `mdex-property_IsSingleAssign` must also be set to `true`). |
| `mdex-property_IsSingleAssign` | Sets a standard attribute to be either single-assign (only one assignment from the attribute can be made on a record) or multi-assign (multiple assignments can be made on a record).<br>The setting can be modified as follows:<br>• A single-assign setting can be changed to a multi-assign setting only if the PDR is non-unique (that is, `mdex-property_IsUnique` is set to `false`).<br>• A multi-assign setting can be changed to a single-assign setting only if all assignments of this standard attribute are single assign. If at least one record has multi-assignments of this standard attribute, an attempted change will fail. |

| Item | Restrictions on modifying (if exist) |
| --- | --- |
| `mdex-config_SystemRecordVersion`<br>`mdex-property_Type`<br>`mdex-dimension_Key`<br>`mdex-config_Key` | These items, which include the data type of attributes, can never be modified. |

# Part II

## Web Services

Chapter 3

# Configuration Web Service Interface

This section describes the operations of the Configuration Web Service.



## About the Configuration Web Service

The Configuration Web Service lets you create and change the records schema and configuration, for the data domain.

### Accessing the web service

The service is declared in its WSDL document. You can access the WSDL at this URL:

```
<host>:<port>/endeca-server/ws/config/<dataDomain>?wsdl
```

where the host and port represent the Oracle Endeca Server, and the `dataDomain` is the name of the data domain created on the server.

### Operation description

A request to the Configuration Web Service consists of a `configTransaction` element, which contains a series of operations that read the configuration and schema and also update it. Operations can be combined arbitrarily in a single service request; each of the operations can appear at most once. The operations perform actions on PDRs (Property Description Records), DDRs (Dimension Description Records), groups, the GCR (Global Configuration Record), and on XML configuration documents.

The effect of a Configuration Web Service request that contains `put` operations is to add attributes, XML configuration documents, or the Global Configuration Record to the specified Endeca data domain:

- If a record with the specified key already exists in the data domain, it is replaced.
- If a record does not exist, it is created.

### Request

The input to the Configuration Web Service depends on the operation used. It can include attribute schema records (PDRs and DDRs), Global Configuration Record, groups, and a set of XML configuration documents.

Any request to the Configuration Web Service can contain an optional element `OuterTransactionId` that specifies the ID of an outer transaction (if it has been started by the Transaction Web Service). The following statements describe the interaction of configuration requests with outer transactions:

- If an outer transaction has been started by the Transaction Web Service, the configuration request may be run against either the latest version of the data files inside the transaction, or against the pre-transaction version of the data files:
  - To run a configuration request against the latest version, the `OuterTransactionId` element in your request must specify the ID issued by the Transaction Web Service when the transaction was started. This element must be the first element specified in your request.
  - To run against the published version (it could be the version published prior to the outer transaction, or the version published after the outer transaction has been committed or rolled back), the `OuterTransactionId` element must be empty or omitted.

It is incorrect to specify an outer transaction ID when an outer transaction is not in progress. All configuration requests with incorrectly specified outer transaction IDs fail with a SOAP fault.

### Response

Not all operations in the Configuration Web Service return data.

If the operation returns data, the response to the Configuration Web Service is a results element, within which each of the submitted operations produces an element showing its own results.

If any operation does not succeed, the whole Web service transaction returns a SOAP fault and none of the operations are applied. An operation may not succeed if an outer transaction has been started by a Transaction Web Service, but an incorrect ID has been specified within a request sent to the Configuration Web Service.

#### Example

The following example of a Configuration Web Service request is used to retrieve a list of attribute groups:

```
<config:configTransaction>
   xmlns:config="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:mdex="http://www.endeca.com/MDEX/config/XQuery/2009/09">
<config:listGroups/>
</config:configTransaction>
```

# Configuration Web Service operations

This topic lists the operations available in the Configuration Web Service.

A request to the Configuration Web Service consists of a `configTransaction` element.

## Operations for PDRs

The operations on Property Description Records (PDRs) are the following:

| Operation | Description |
|---|---|
| `exportProperties` | Return all Property Description Records (PDRs) for the specified Endeca data domain. |
| `listProperties` | Return the key of the PDRs for the standard attributes present in the schema of the Endeca data domain. |
| `getProperties` | Return PDRs for the specified attribute keys. Attribute keys are obtained from `listProperties`. |
| `putProperties` | Add the PDRs (specified as an argument) to the schema of the Endeca data domain. If an attribute with the same key exists, it is replaced. |
| `updateProperties` | Lets you add or modify specified assignments on the PDR.<br>As an argument, specify an attribute key associated with an existing PDR and zero or more assignments.<br>The operation replaces the assignment on the PDR with a new assignment if it is provided as an argument.<br>There is no requirement to specify the entire PDR to this operation; there is a requirement to specify the standard attribute key.<br>  **Note:** If you update an assignment on the PDR in a data set with a large number of existing records, this operation can increase the processing time and affects performance. |
| `setPropertyDefaultLanguage` | Sets the default language for new standard attributes (PDRs) that are created automatically by the Data Ingest Web Service (DIWS) or the Bulk Load Interface. The default language is also used if the `mdex-property_Language` property is not explicitly set during the creation of a PDR by DIWS or the Bulk Load Interface. (Note that PDRs created by the Configuration Web Service's `putProperties` and `import` operations must be fully and explicitly specified.)<br>The default language is specified with one of the language IDs listed in *Supported languages on page 154*. If a language ID is not specified, then the default PDR language will be set to `unknown`. |
| `getPropertyDefaultLanguage` | Returns the default language ID that is used for PDRs. The language ID will be either `unknown` (the default) or the language ID that was set by a previous `setPropertyDefaultLanguage` operation. |

## Operations for DDRs

The operations on Dimension Description Records (DDRs) for managed attributes are the following:

| Operation | Description |
|---|---|
| `exportDimensions` | Return all Dimension Description Records. |
| `listDimensions` | Return the key of each managed attribute present in an Oracle Endeca Server data domain. |
| `getDimensions` | Return DDRs for specified managed attribute keys. Managed attribute keys are obtained from `listDimensions`. |
| `putDimensions` | Add the DDRs (specified as arguments) to an Oracle Endeca Server data domain. If a managed attribute with the same key exists, it is replaced. |
| `updateDimensions` | Lets you add or modify one or more specified assignments (managed attributes) on the DDR.<br>As an argument, specify a managed attribute key associated with an existing DDR and zero or more assignments.<br>The operation replaces the assignment on the DDR with a new assignment if it is provided as an argument.<br>There is no requirement to specify the entire DDR to this operation; there is a requirement to specify the managed attribute key.<br>  **Note:** If you update an assignment on the DDR in a data set with a large number of existing records, this operation can increase the processing time and affects performance. |

## Operations for Managed Attribute Values

The operations on managed attribute values are the following:

| Operation | Description |
|---|---|
| `listManagedAttributeValues` | If a managed attribute is specified, return a `managedAttributeValues` element containing all managed attribute values for the specified managed attribute, including each managed attribute record's spec, parent managed attribute value, associated managed attribute, and, if specified, the name, rank and synonyms. The values are ordered by managed attribute.<br>If no managed attribute is specified, return all managed attribute values present in the index of the Endeca data domain. |

| Operation | Description |
|---|---|
| `putManagedAttributeValues` | Add or update managed attribute values. If a managed attribute value already exists with the given spec for the given managed attribute, it is updated. Otherwise, a new managed attribute value record is added. For the root managed attribute value, specify "/" as the parent spec. |

## Operations for Attribute Groups

The operations on attribute groups are the following:

| Operation | Description |
|---|---|
| `importGroups` | Remove any existing groups and add the groups with specified attributes. |
| `exportGroups` | Return the full representation of each group. |
| `listGroups` | Return a summary of each group. |
| `getGroups` | Return the specified groups.<br>This operation returns groups in the order in which you specify the keys for each group. This operation creates a summary of each existing group which includes the group key, the display name (if it exists), and the cardinality of the group.<br>This operation returns attributes for all user-specified groups and attributes that do not belong to any user-specified groups. To request all attributes that do not belong to any user-specified groups, specify the key `system-navigation_InternalGroup`. |
| `putGroups` | Add or replace each of the specified groups. |
| `deleteGroups` | Delete each of the specified groups. |
| `updateGroupConfigs` | Lets you add or modify specified assignments on the group description record. As an argument, specify a `system-group_Key` indicating which group to update, and zero or more assignments in the group description record. The operation replaces the assignment on the group description record with a new assignment if it is provided as an argument.<br>There is no requirement to specify the entire group description record to this operation; there is a requirement to specify the `system-group_Key` associated with the group. |

For examples of Configuration Web Service requests for groups and for information on how to retrieve group configuration information with the Conversation Web Service, see *Working with attribute groups using the API on page 211*.

## Operations for Global Configuration Record

The operations for Global Configuration Record are the following:

| Operation | Description |
|---|---|
| `getGlobalConfigRecord` | Obtain the Global Configuration Record from the data domain. |
| `putGlobalConfigRecord` | Replace the Global Configuration Record in the data domain. |

## Operations for Dgraph configuration documents

The operations for managing the Dgraph configuration documents are the following:

| Operation | Description |
|---|---|
| `listConfigDocuments` | Return the names of the Dgraph process configuration documents. |
| `getConfigDocuments` | Return the requested Dgraph process configuration documents. |
| `putConfigDocuments` | Add or replace each of the specified Dgraph process configuration documents. |

## Operations for precedence rules

The operations for managing precedence rule records are the following:

| Operation | Description |
|---|---|
| `listPrecedenceRules` | Return the names of the precedence rules configured for the data domain. |
| `putPrecedenceRules` | Add or replace each of the specified precedence rules. |
| `deletePrecedenceRules` | Take a list of precedence rule keys and completely delete each rule. |
| `exportPrecedenceRules` | Return the full representation of each precedence rule. |
| `importPrecedenceRules` | Remove any existing precedence rules and add the specified ones. |

### Global operations

The Configuration Web Service has the following global operations:

| Operation | Description |
|---|---|
| `export` | Export all attributes, groups, configuration documents, and the Global Configuration Record. |
| `import` | Import all attributes, groups, configuration documents, and the Global Configuration Record. |

## Loading an attribute schema

You can use the Configuration Web Service to load the schema (PDRs and DDRs) for your standard and managed attributes.

It is recommended to load your attribute schema before loading your source records, so that you can modify the resulting PDRs and DDRs as needed, without causing the Dgraph process to reindex the data set. After the PDRs and DDRs have been configured as desired, you can then use the Data Ingest Web Service to load your source records.

- The `putProperties` operation loads records that describe characteristics of standard attributes (these records are known as PDRs).
- The `putDimensions` operation loads records that describe characteristics of managed attributes (these records are known as PDRs and DDRs).

To illustrate the use of this operation, consider this `configTransaction` simple example from the Configuration Web Service:

```
<config-service:configTransaction
   xmlns:config="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:mdex="http://www.endeca.com/MDEX/config/XQuery/2009/09">
  <config-service:putProperties>
    <mdex:record>
      <mdex-property_Key>ProductID</mdex-property_Key>
      <mdex-property_DisplayName>Product ID</mdex-property_DisplayName>
      <mdex-property_IsSingleAssign>true</mdex-property_IsSingleAssign>
      <mdex-property_IsUnique>true</mdex-property_IsUnique>
      <mdex-property_IsTextSearchable>false</mdex-property_IsTextSearchable>
      <mdex-property_TextSearchAllowsWildcards>false</mdex-property_TextSearchAllowsWildcards>
      <mdex-property_IsPropertyValueSearchable>false</mdex-property_IsPropertyValueSearchable>
      <mdex-property_Type>mdex:int</mdex-property_Type>
      <mdex-property_Language>en</mdex-property_Language>

    </mdex:record>
    <mdex:record>
      <mdex-property_Key>BikeType</mdex-property_Key>
      <mdex-property_DisplayName>Bike Type</mdex-property_DisplayName>
      <mdex-property_IsSingleAssign>true</mdex-property_IsSingleAssign>
      <mdex-property_IsTextSearchable>true</mdex-property_IsTextSearchable>
      <mdex-property_IsUnique>false</mdex-property_IsUnique>
      <mdex-property_TextSearchAllowsWildcards>true</mdex-property_TextSearchAllowsWildcards>
      <mdex-property_IsPropertyValueSearchable>true</mdex-property_IsPropertyValueSearchable>
      <mdex-property_Type>mdex:string</mdex-property_Type>
      <mdex-property_Language>en</mdex-property_Language>
    </mdex:record>
  </config-service:putProperties>
```

```
  <config-service:putDimensions>
    <mdex:record>
      <mdex-dimension_Key>BikeType</mdex-dimension_Key>
      <mdex-dimension_EnableRefinements>true</mdex-dimension_EnableRefinements>
      <mdex-dimension_IsDimensionSearchHierarchical>true<
/mdex-dimension_IsDimensionSearchHierarchical>
      <mdex-dimension_IsRecordSearchHierarchical>true</mdex-dimension_IsRecordSearchHierarchical>
    </mdex:record>
  </config-service:putDimensions>
</config-service:configTransaction>
```

First, the example uses `putProperties` to create two standard attributes (`ProductID` and `BikeType`). The `ProductID` attribute is configured as a single-assign, unique attribute, so that it can be used as a primary key for records. The `BikeType` attribute is the standard attribute record used for the creation of the `BikeType` managed attribute. Next, this example uses `putDimensions` to create the `BikeType` managed attribute.

To load an attribute schema into the Endeca data domain:

1. Make sure that the Oracle Endeca Server and the data domain are running. Access the Configuration Web Service for the data domain: `http://localhost:<port>/ws/config/dataDomain?wsdl`.
2. Make a SOAP request to the Configuration Web Service as shown above.

If the request is successful, the response will look like this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header/>
 <soapenv:Body>
   <config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3/0"/>
 </soapenv:Body>
</soapenv:Envelope>
```

# Loading configuration documents

You can use the Configuration Web Service to load the XML configuration documents to the data domain's configuration.

If you load your configuration documents before loading your source records, you can modify them as needed. After they have been configured as desired, you can then use the Data Ingest Web Service to load your source records.

You can load the following configuration documents:

- `dimsearch_config`
- `recsearch_config`
- `relrank_strategies`
- `stop_words`
- `thesaurus`

For more information on the syntax of these documents, see the section *Dgraph Configuration Reference on page 333*.

The operations for managing the XML configuration documents are the following:

| Operation | Description |
|---|---|
| `listConfigDocuments` | Return the names of the Dgraph process configuration documents. |
| `getConfigDocuments` | Return the requested Dgraph process configuration documents. |
| `putConfigDocuments` | Add or replace each of the specified Dgraph process configuration documents. |

The following example illustrates the use of the `putConfigDocuments` operation to load the `RECSEARCH_CONFIG` configuration document. The `RECSEARCH_CONFIG` document creates one search interface (named `All`) that consists of the `ProductType`, `Region`, and `Description` attributes:

```
<soap:Envelope>
<soap:Body>
<config:configTransaction>
<config:putConfigDocuments>
    <mdex:configDocument name="recsearch_config">
      <RECSEARCH_CONFIG>
        <SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="ALWAYS" NAME="All">
          <MEMBER_NAME RELEVANCE_RANK="3">ProductType</MEMBER_NAME>
          <MEMBER_NAME RELEVANCE_RANK="2">Region</MEMBER_NAME>
          <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
        </SEARCH_INTERFACE>
      </RECSEARCH_CONFIG>
    </mdex:configDocument>
  </config:putConfigDocuments>
</config:configTransaction>
</soap:Body>
</soap:Envelope>
```

To load the XML configuration documents into the data domain of the Oracle Endeca Server:

1. Make sure that the Oracle Endeca Server and the data domain are running. Access the Configuration Web Service for the data domain as in this example: `http://<host>:<port>/ws/config/<DataDomain>?wsdl`.
2. Make a SOAP request to the Configuration Web Service as shown above.

If the request is successful, the response contains a SOAP success message, similar to the following:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header/>
 <soapenv:Body>
   <config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3/0"/>
 </soapenv:Body>
</soapenv:Envelope>
```

# Performance impact of schema and configuration changes

Changes to the schema and configuration can be made on an empty index (before data is loaded), or after the records have already been loaded into the Dgraph. Even though it is possible to change the schema for your records after the records have been loaded, such a change is associated with performance impact.

You can use the `updateProperties` or `updateDimensions` operations of the Configuration Web Service to update the PDRs and DDRs (the records schema) after the records have already been added to the Dgraph

process index. However, changes to the schema on an existing index are not recommended for performance reasons, since such changes cause the Dgraph to reindex the data set, which is associated with an increase in CPU and memory usage.

Note also that not all changes to the PDRs and DDRs can be done on an existing data domain containing records.

In addition, changes to the configuration, such as to the settings of the GCR, while they can be done on a running data domain that contains records, are also associated with performance impact in cases when the data domain contains a large number of records.

# Using the Configuration Web Service in Integrator ETL

The Configuration Web Service lets you perform operations with the schema and configuration documents. All of these operations are supported in Integrator ETL, which uses the Configuration Web Service requests.

You can load record-based configuration files using either one of its connectors, or the **WebServiceClient** component. For example, you can use a dedicated component of Integrator ETL to load schema records, or records representing precedence rules. You can also use the **WebServiceClient** component of Integrator ETL to load the Global Configuration Record and configuration documents.

For more information on Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

Chapter 4

# Conversation Web Service Interface

This section describes the role and operations of the Conversation Web Service.



## About the Conversation Web Service

The Conversation Web Service provides the primary means of querying data in the Oracle Endeca Server.

### Overview

This Web service interface can be used by any front-end application powered by the Oracle Endeca Server. The interface is used by Studio to send queries (such as navigation or search queries) to the Oracle Endeca Server. You can also use it on its own. It is a WS-I compliant SOAP/HTTP Web service that also supports the wrapped-document/literal pattern of binding.

The service supports fundamental Oracle Endeca Server behavior, such as:

- Guided navigation
- Record search
- Value search
- Communication between the front-end application client and the Oracle Endeca Server
- A range of summarizations

The Conversation Web Service is declared in `conversation.wsdl`, and uses several library helper modules.

To view the WSDL document for the Conversation Web Service, issue the following command:

```
http://host:<port>/endeca-server/ws/conversation/dataDomain?wsdl
```

where `host` and *`port`* represent the machine on which an instance of the Oracle Endeca Server is running, and *`dataDomain`* is the name of the Endeca data domain.

The service's version is listed in one of its namespaces included in the WSDL, as shown in the following example (the version in this example may not match the version of the service you have installed):

```
xmlns:cs_v3_0="http://www.endeca.com/MDEX/conversation/3/0"
```

In this example, 3 is the major version; 0 is the minor version. If more than one minor version is supported, it is listed in its own namespace in the WSDL.

For reference information on the Conversation Web Service operations and for schema elements, see the *Oracle Endeca Server API References*.

# Conversation Web Service operations

The Conversation Web Service interface provides operations that query the Oracle Endeca Server.

## Operation description

At a high level, the Conversation Web Service facilitates a dialog with users about data. A typical request consists of the filter state and one of the content element configurations. Here is an abbreviated example:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   ...
   <State>
   ...
   </State>
    <RecordListConfig>
    ...
   </RecordCountConfig>
</Request>
```

- The state is represented by a `State` element in the request. It describes the records to be selected by the request. The state may be empty or may contain components that change the filter state or reconfigure content elements, typically as a result of user actions. A request must contain at least one state. If the request contains multiple states, then each state must have a name.
- A summarization information about the records, included in one of the elements that are types of the `ContentElementConfig` element from the Conversation Web Service. Note that in any request, the `ContentElementConfig` base type itself is not included. Instead, its sub-type is included, such as `RecordListConfig` type in the example. From now on, the Endeca Server documentation refers to `ContentElementConfig` as "content element config". Any content element config provides summarization information about the records or other information specified in the state. The request may contain multiple `ContentElementConfig` elements.

The Conversation Web Service, unlike other Web services of the Endeca Server, implies that you create a series of related requests, thus creating a "dialog" with your data. The sequence of actions in the Conversation Web Service dialog is as follows:

1. A user issues a query using the front-end application.

   In the Conversation Web Service, the request is reflected in the `Request` complex type (in the WSDL, all complex types are listed as `ComplexType`).

   This query is used to construct an initial filter state (which may be empty or may contain one or more filters, as well as a collection name), and one or more content element configs. These filter state and content element configurations are sent in a Conversation Web Service request.
2. The response to this initial request returns a `Results` type, which contains the original state and the records (or other information) in one or more content element config elements.
3. When the user chooses a particular action, the front-end application submits a new request through the Conversation Web Service. Typically, this subsequent request is constructed from the results returned by the previous invocation of the Conversation Web Service request. Specifically, the request sends in the state returned by the previous response and any new filters or content element configs that correspond to the user actions.

4. The response returns a transformed query along with new state and the new content element configs.

Note that during this conversation, the user may request a completely different type of action, which will then require a new and different request.

## Request

The `Request` operation is a complex type with the following schema:

```
<complexType name="Request">
  <sequence>
    <element  minOccurs="0"  name="OuterTransactionId"  type="cs_v2_0:NonEmptyString" />
    <element  default="en"  minOccurs="0" name="Language"  type="cs_v2_0:NonEmptyString" />
    <element  maxOccurs="unbounded"  minOccurs="0"  name="State"  type="cs_v2_0:State" />
    <group    maxOccurs="unbounded"  minOccurs="0"  ref="cs_v2_0:ContentElementConfig" />
    <element  minOccurs="0" name="PinDataVersion"  type="cs_v2_0:NonEmptyString" />
    <element  minOccurs="0" name="DataVersionRequested"  type="cs_v2_0:NonEmptyString" />
  </sequence>
</complexType>
```

Each request may specify:

| Element | Description |
|---|---|
| `OuterTransactionId` | Optional. If used, must be the first element in the request. It must be specified only if the request runs within an outer transaction. For details on outer transactions, see *Transaction Web Service Interface on page 78*. |
| `Language` | Optional. Specifies a language code for error messages generated during parsing of EQL statements. For details on this element and its supported language codes, see *Language codes for EQL error messages on page 127*. |
| `State` | Required. Contains inputs that affect the set of records to operate on. For example, a state may contain record filters (such as a record search filter, a selected refinement filter, and EQL record filters) and the name of a collection to search. A request can have multiple states (in which case, each state must be named). An unnamed state can exist only if it is the only state in the request. |

| Element | Description |
|---|---|
| `ContentElementConfig` | Optional. Represents summarization configuration information relative to the records returned from a specific state. Different types of `ContentElementConfig` exist. Types can describe, for example, a summarization of a filter state or the data therein, such as a set of breadcrumbs, a navigation menu, or the data for a grid or chart. The types are:<br>• `AttributeGroupListConfig`<br>• `AvailableSearchKeysConfig`<br>• `BreadcrumbConfig`<br>• `EQLConfig`<br>• `NavigationMenuConfig`<br>• `PropertyListConfig`<br>• `RecordCountConfig`<br>• `RecordDetailsConfig`<br>• `RecordListConfig`<br>• `SearchAdjustmentsConfig`<br>• `ValueSearchConfig` |
| `PinDataVersion` | Optional. Specifies a timeout value during which the Endeca Server should hold on to the current data version. This data version becomes pinned and is maintained in memory for the duration of the timeout. The timeout value specified with this element must fall within minimum and maximum values listed in the `EndecaServer.properties` file. The pinned version number is returned in the `X-Endeca-Served-Data-Version` header of the response to a request that uses `PinDataVersion`. |
| `DataVersionRequested` | Optional. Specifies the number of the pinned version. The request that includes `DataVersionRequested` must be issued within the timeout period specified when the version was pinned. This resets the timeout. If the request is issued after this timeout period, the pinned version may have expired. |

## Response

The `Request` operation outputs a `Results` response, which includes the `State` and, optionally, one or more of the `ContentElement` types that resulted from the request's `ContentElement` (for example, a `RecordList` is returned from a `RecordListConfig`).

Each response from the Conversation Web Service includes in its header two version numbers — the served version number and the latest version number:

```
X-Endeca-Served-Data-Version
X-Endeca-Data-Version
```

The `X-Endeca-Served-Data-Version` also represents the number of the pinned data version (if you pin it).

### Error example

On failure, the SOAP fault is thrown. Its `faultstring` element contains information about the request that caused the error, and the `detail` element includes pointers to the location of errors in the request.

*State elements*

*Content element config summarizations*

## State elements

The `State` type contains inputs that determine the set of records to operate on.

Every request must contain at least one state, each one defining a navigation state for associated ContentElementConfigs.

The format of the `State` type is:

```
<State>
   <Name>?</Name>
   <CollectionName>?</CollectionName>
   <SelectedRefinementFilter Name="?" Spec="?" Id="?">
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
   </SelectedRefinementFilter>
   <TextSearchFilter Key="?" RelevanceRankingStrategy="?" Mode="?"
      EnableSnippeting="?" SnippetLength="?" Language="?">?</TextSearchFilter>
   <RecordKind>?</RecordKind>
   <DataSourceFilter Id="?">
      <filterAST>
         <typ:filter/>
      </filterAST>
      <filterString>?</filterString>
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
   </DataSourceFilter>
   <SelectionFilter Id="?">
      <filterAST>
         <typ:filter/>
      </filterAST>
      <filterString>?</filterString>
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
   </SelectionFilter>
</State>
```

The meanings of the major elements are:

| Element | Min/Max Occurrences | Description |
|---|---|---|
| `Name` | min=0 max=1 | The name of a `State`.<br>• In a request with multiple states, each state must use the `Name` element and each name must be distinct from other state names in that request.<br>• In a request with only one state, it is optional for the state to be named. |
| `CollectionName` | min=0 max=1 | The name of an existing collection to operate on. Note that collection names are case sensitive. |
| `SelectedRefinementFilter` | min=0 max=unbounded | Creates a refinement navigation query from a specific refinement. For details, see *SelectedRefinementFilter on page 192*. |
| `TextSearchFilter` | min=0 max=unbounded | Performs a keyword search against specific attribute values assigned to records. For details on its format, see *Record search filter on page 247*. |
| `DataSourceFilter` | min=0 max=unbounded | Uses EQL syntax to filter the corpus of records before any other processing is done. For details, see *EQL Record Filters on page 117*. |
| `SelectionFilter` | min=0 max=unbounded | Uses EQL syntax to provide selection criteria for the final record result set. Typically used in conjunction with `DataSourceFilter`. For details, see *EQL Record Filters on page 117*. |

## Rules for State usage

The rules for states in a request are:

- A state in a request is defined by the `State` type.
- Every request must have at least one state (i.e., one `State` type).
- A request can have more than one state. If multiple states are included, each state must have a unique name.
- If only one state is in the request, the state can be unnamed (i.e., the `Name` element is not used in the `State` definition). If the state is unnamed, then there can be one (and only one) state in the request.
- If the request contains only one unnamed state, the state can be empty (that is, `<State/>`). By definition, an empty state is an unnamed state.

- A state can have multiple filters that use the `Id` element to provide an identifier (such as `SelectionFilter Id="SalesSearch"`). In this case, each filter identifier must be unique within the state. If they are not unique, Endeca Server throws an exception.

In addition, if a request has multiple states, then each config must reference one (and only one) state. If the request has only one named state, then it is optional as to whether the config references that state (as the state will be used in any event in the config).

## Example of a multi-state request

This simple example shows a request with two named states (FlavorSearch and WineSearch), each of which is performing a record search on a different attribute. The results are summarized by two `RecordCountConfig` types, each of which uses the `StateName` element to associate it with one of the named states:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <Name>FlavorSearch</Name>
      <TextSearchFilter Key="Flavors" Mode="AllPartial" Language="en">oak</TextSearchFilter>
   </State>
   <State>
      <Name>WineSearch</Name>
      <TextSearchFilter Key="Wine" Mode="AllPartial" Language="en">merlot</TextSearchFilter>
   </State>
   <RecordCountConfig Id="FlavorRecs">
     <StateName>FlavorSearch</StateName>StateName>
   </RecordCountConfig>
   <RecordCountConfig Id="WineRecs">
     <StateName>WineSearch</StateName>StateName>
   </RecordCountConfig>
</Request>
```

The results of the search look like this example:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
         xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <State xmlns="http://www.endeca.com/MDEX/conversation/3/0"
         xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
         xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <Name>FlavorSearch</Name>
      <TextSearchFilter Key="Flavors" Mode="AllPartial" Language="en">oak</TextSearchFilter>
   </State>
   <State xmlns="http://www.endeca.com/MDEX/conversation/3/0"
         xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
         xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <Name>WineSearch</Name>
      <TextSearchFilter Key="Wine" Mode="AllPartial" Language="en">merlot</TextSearchFilter>
   </State>
   <cs:RecordCount Id="FlavorRecs">
      <cs:NumRecords>6381</cs:NumRecords>
   </cs:RecordCount>
   <cs:RecordCount Id="WineRecs">
      <cs:NumRecords>3073</cs:NumRecords>
   </cs:RecordCount>
</cs:Results>
```

As the results show, the record search from the FlavorSearch state returns 6381 records, while the record search from the WineSearch state returns 3073 records.

## Content element config summarizations

The content element configs describe which summarizations should be performed on the resulting data.

Each Conversation Web Service request should specify one the following content element configs:

| Content element config | Description |
|---|---|
| `AttributeGroupListConfig` | Retrieves a list of attribute groups in a data domain. For details, see *Retrieving lists of groups on page 212*. |
| `AvailableSearchKeysConfig` | Retrieves a list of the searchable attributes and search interfaces available in the data domain. For details, see *Obtaining the available search keys on page 246*. |
| `BreadcrumbConfig` | Retrieves breadcrumbs (explicitly-selected refinements) in the current navigation state. For details, see *BreadcrumbConfig on page 221*. |
| `EQLConfig` | Allows arbitrary EQL statements to be evaluated. If the request has one or more named states, this config must reference one (and only one) of the states. For details, see *EQLConfig requests on page 135*. |
| `NavigationMenuConfig` | Contains inputs that define what is returned in the navigation menu. For details, see *NavigationMenuConfig on page 185*. |
| `PropertyListConfig` | Returns all the attributes in a data domain. For details, see *Obtaining a list of available attributes on page 193*. |
| `RecordCountConfig` | Returns the number of records in the state. For details, see *Displaying record counts on page 146*. |
| `RecordDetailsConfig` | Configures the detail aspects of a returned record. For details, see *Displaying record details on page 144*. |
| `RecordListConfig` | Contains inputs that define what is returned in a list of records. For details, see *Configuring a record list on page 138*. |
| `SearchAdjustmentConfig` | Retrieves automatic spelling corrections and suggested corrections (Did You Mean) information. For details, see *Retrieving spelling corrections and DYM in query results on page 299*. |
| `ValueSearchConfig` | Controls the behavior of a single value search. For details, see *Value search query format on page 260*. |

### Usage rules

The rules for using content element configs in a request are:

- Each request may include multiple content element configs.

- Each content element config may reference a maximum of one state.
- If the request has one (and only one) unnamed state, then each content element config cannot have a state reference (because the state has no name).
- If the request has one or more named states, each content element config must reference one (and only one) of the states. The exceptions are the `AttributeGroupListConfig`, `AvailableSearchKeysConfig`, and `PropertyListConfig` types, where specifying a state is optional.

# Pinning data versions

The Dgraph process of the Endeca Server maintains knowledge about loaded records using the index. Since the index can be updated continuously, at any given time, several data versions of the index are maintained in memory. In some cases, it may be useful for the front-end applications (such as Studio), to issue queries against a specific data version. This process is known as *pinning a data version*.

Using two optional operations in the Conversation Web Service, and the settings for timeouts in the `EndecaServer.properties` file, you can:

- Pin a data version for a specified timeout. Once the timeout expires, the data version is dismissed by the Endeca Server.
- Issue queries while requesting a previously pinned data version. If the queries are issued within the timeout period, they use the pinned version. The timeout is renewed from the time of the last valid request that used the pinned version.
- Pin a version, wait and pin it again with a longer timeout period that is still less than the maximum timeout listed in `EndecaServer.properties`. In this case, the timeout is renewed to a longer timeout you specify.

*Timeout default, maximum and minimum values*

*Holding on to a data version*

*Requesting a pinned data version in a query*

## Timeout default, maximum and minimum values

Each data version you would like to pin can be held only for the duration of the specified timeout. The timeout default value, and its maximum and minimum are maintained in the `EndecaServer.properties` file in `$DOMAIN_HOME/config`.

You can modify the file to change these values. if you do so, and run an Endeca Server cluster, the values must be modified on all machines hosting an Endeca Server instance. The `EndecaServer.properties` file has the following default values related to maintaining specific data versions in memory:

| Parameter | Description |
|---|---|
| `endeca-ds-pin-timeout-min` | The default minimum value is 60000 milliseconds (ms). This is the lowest timeout value the Dgraph can use when `PinDataVersion` request is issued. |

| Parameter | Description |
|---|---|
| `endeca-ds-pin-timeout-max` | The default maximum value is 300000 ms. This is the highest timeout value the Dgraph process can use when `PinDataVersion` request is issued. |
| `endeca-ds-pin-timeout-default` | The default value for the timeout is 120000 ms. This is the default timeout value used by the Dgraph if you do not specify the value in the `PinDataVersion` request. (You typically use this request of the Conversation Web Service to pin a data version). |

This topic lists only those values that are related to data version pinning. For a full list of parameters in the `EndecaServer.properties` file, see the *Endeca Server Administrator's Guide*.

## Holding on to a data version

Endeca Server lets you hold on to (or pin) a specific version of the data in memory. Holding on to a version is useful, for example, when the front-end application (such as Studio), needs to return consistent page results. It is also useful to hold on to a specific version if you want to export a large number of records from Endeca Server.

You can issue a Conversation Web Service request to hold a version in memory, so that a subsequent request can ask for this version when performing query processing. The following statements describe the behavior of Endeca Server:

- When you issue a request to hold a data version, a data version is kept in memory (or is "pinned") for a period of time you specify.
- In the request, you can optionally specify the value for the timeout. Timeout is the time after which the version is dismissed by Endeca Server. It is counted from the time of the most recent request that uses this data version. If you do not specify a timeout, the default timeout from `EndecaServer.properties` file is used.
- If you specify a timeout value that is less than the minimum, or greater than the maximum values, the request returns an error.
- If you continue issuing requests against the pinned version, the timeout is renewed.
- The pinned version expires after a timeout, and is removed by Endeca Server if you stop issuing queries against this version.
- If you issue multiple requests in succession to hold on to a data version, each time specifying a different timeout value within the minimum and maximum boundaries, the greater of the timeout values is used. Consider this example, where you have already pinned a version by specifying a timeout of 10 minutes, and your maximum timeout value is 20 minutes:
    - If you issue a new request with the 15 minutes timeout, Endeca Server increases the timeout to this value, because it is greater than the existing timeout of 10 minutes. it also checks the new value is less than the maximum of 20 minutes in `EndecaServer.properties`.
    - If you issue a new request with the 7 minutes timeout, Endeca Server checks how much time is left in the existing timeout. If, for example, 7 minutes are left in the existing timeout of 10 minutes, this

timeout is not reset. Similarly, if 8 minutes are left, the current timeout is not reset because 8 > 7. However, if 5 minutes are left, the timeout is increased to 7 minutes.

- A version is pinned by Endeca Server on a specific Dgraph in the data domain. To guarantee this, Endeca Server lets you pin a data version only if your data domain uses session affinity. Session affinity guarantees that a request is routed to the same Dgraph in the data domain, if the Dgraph is available. (If you use a default data domain profile, session affinity is enabled by default.) For details on configuring session affinity, see the *Endeca Server Administrator's Guide*.

To pin a data version:

1. Check that the data domain profile uses session affinity. Issue the following commands in succession:

   ```
   endeca-cmd get-dd <data_domain_name>
   endeca-cmd get-dd-profile <data_domain_profile_name>
   ```

   The `get-dd` command returns the details of your data domain, including its profile. The `get-dd-profile` command returns the characteristics of the data domain profile.

   The following lines in the profile indicate that session affinity is specified for the data domain:

   ```
   session-id-type: header
   session-id-key: X-Endeca-Session-ID
   ```

   In this example, session affinity uses `header` (other methods are possible for session affinity. See the *Endeca Server Cluster Guide*).

2. On the host and port of your Endeca Server deployment, issue a Conversation Web Service request that specifies a timeout, as in this abbreviated example:

   ```
   ...
   <ns:Request>
    <!--Optional:-->
     <ns:PinDataVersion>optional_pin_timeout</ns:PinDataVersion>
   <ns:State>
     ....
   </ns:State>
   ```

   where *`optional_pin_timeout`* can be any integer value in milliseconds that is within the boundaries of the minimum and maximum timeout values specified in the `EndecaServer.properties` file in `$DOMAIN_HOME/config`. In that file, the defaults are as follows: the minimum timeout is 60000 milliseconds (1 minute), and the maximum timeout is 300000 ms (5 minutes). The default listed in the file is 120000 ms (2 minutes). If the timeout is not specified in the request, it uses the default from the file.

If successful, the Endeca Server returns an empty Conversation Web Service response, similar to the following:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
            xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
    <cs:State/>
</cs:Results>
```

The header of the response includes:

- The `X-Endeca-Served-Data-Version` — this is the version that you pinned.
- The `X-Endeca-Data-Version` — this is latest data version.

The latest data version may or may not differ from the pinned data version. Here is an example of a header, in which the served data version (the one you pinned), and the latest data version are the same, which is a typical case:

```
X-Endeca-Served-Data-Version 61
X-Endeca-Data-Version 61
```

Once you obtain the pinned data version, you can then use it in subsequent requests issued within the specified timeout. To use the pinned version, include `DataVersionRequested` element in subsequent requests, while staying within the timeout.

## Requesting a pinned data version in a query

If you have previously pinned a particular data version, then within the specified timeout, you can issue a request to Endeca Server that will be processed against this version of the data. To do so, you specify a previously pinned data version with `DataVersionRequested` element in your request.

The `<DataVersionRequested>` element is optional, meaning that you can include it into any Conversation Web Service request. (Although, you cannot issue a request that contains both `<PinDataVersion>` and `<DataVersionRequested>`). A request with `<DataVersionRequested>` can also optionally include an outer transaction ID, so that all other queries are processed within this transaction, while using the same data version.

Issuing a query request against a specific data version is useful when, for example, you would like to obtain a large number of records from Endeca Server (such as, to export them in bulk), or, when the user interface of your front-end application needs to present consistent results to the user. For example, in Studio, various components issue requests to Endeca Server, and it is desirable for them to represent the same state of the data, even if the data is being updated in real time, while end users continue issuing queries.

Before you request a specific version of the data with `<DataVersionRequested>`, use `PinDataVersion` in a previous request, to indicate to Endeca Server that you would like to hold on to a version for a specified timeout period. The response to the version-pinning request returns the number of the pinned data version (and a current data version number, which may differ from the pinned version number), in the header. You can then use the pinned data version number in any subsequent requests. These requests will be processed against the pinned version, if they are issued within the timeout period.

To request a previously pinned data version:

1. Issue a Conversation Web Service request that includes the following element:

   ```
   <DataVersionRequested>VersionNum</DataVersionRequested>
   ```

   where *`VersionNum`* is the number of the version you asked Endeca Server to hold. It was returned in the header of the `PinDataVersion` request.

   If you continue issuing requests with this version number within the specified timeout period, the timeout is renewed. If you attempt to issue a request with this version after the timeout has expired, the request returns an error. If you request a version that no longer exists (or never was pinned), the request also returns an error. To check the values for the timeout, see the `EndecaServer.properties`.

Chapter 5

# Entity and Collection Configuration Web Service Interface

This section describes the operations of the Entity and Collection Configuration Web Service.



## About the Entity and Collection Configuration Web Service

This Web Service lets you create and manage entities, collections, and filter rules.

### Overview

The Entity and Collection Configuration Web Service is a WS-I compliant SOAP/HTTP Web service that also supports the wrapped-document/literal pattern of binding. The service is declared in `sconfig.wsdl`.

To view the WSDL document for the service, issue the following command:

```
http://host:<port>endeca-server/ws/sconfig/dataDomain?wsdl
```

where `host` and *`port`* represent the host and port of Oracle Endeca Server, and *`dataDomain`* is the name of the data domain for which entities, collections, or filter rules will be managed.

The service's version is listed in one of its namespaces included in the WSDL, as shown in the following example (the version in this example may not match the version of the service you have installed):

```
xmlns:v3_0="http://www.endeca.com/endeca-server/sconfig/3/0"
```

In this example, 3 is the major version; 0 is the minor version. If more than one minor version is supported, it is listed in its own namespace in the WSDL document.

For reference information on the operations and for schema elements, see the *Oracle Endeca Server API References*.

### Operation description

A request to the Entity and Collection Configuration Web Service depends on the operation. The operations perform actions on entities, collections, and filter rules.

The effect of an Entity and Collection Configuration Web Service request that contains `put` operations is to add entities, collections, or filter rules for this data domain. After creation, each entity, collection, or filter rule is represented as a single logical record in the Endeca data domain. The on-disk storage of these records means that they persist across restarts of the Endeca data domain, as they are loaded into the Dgraph process at start-up time.

### Request

The input to the Entity and Collection Configuration Web Service depends on the operation used. For example, it can include a key and an EQL statement that defines an entity, for `put` operations; it can include the key only, for `deleteEntities` operation; or it can include the definition of the entity, for `validate` operations.

Any request to the Entity and Collection Configuration Web Service can contain an optional element `OuterTransactionId` that specifies the ID of an outer transaction (if it has been started by the Transaction Web Service). The following statements describe the interaction of configuration requests with outer transactions:

- If an outer transaction has been started by the Transaction Web Service, the request may be run against either the latest version of the index files inside the transaction, or against the pre-transaction version of the index files:
  - To run a request against the latest version, the `OuterTransactionId` element in your request must specify the ID issued by the Transaction Web Service when the transaction was started. This element must be the first element specified in your request.
  - To run against the published version (it could be the version published prior to the outer transaction, or the version published after the outer transaction has been committed or rolled back), the `OuterTransactionId` element must be empty or omitted.

It is incorrect to specify an outer transaction ID when an outer transaction is not in progress. All configuration requests with incorrectly specified outer transaction IDs fail with a SOAP fault.

### Response

Not all operations in the Entity and Collection Configuration Web Service return data.

If the operation returns data, the response is a results element, within which each of the submitted operations produces an element showing its own results.

If any operation does not succeed, the whole Web service transaction returns a SOAP fault, and none of the operations are applied. An operation may not succeed if an outer transaction has been started by a Transaction Web Service, but an incorrect ID has been specified within a request sent to the Entity and Collection Configuration Web Service.

## Operations in the Entity and Collection Configuration Web Service

This topic lists the operations of the Entity and Collection Configuration Web Service.

### Entity operations

| Entity operation | Description |
|---|---|
| `putEntity` | Add an entity with the specified entity key and definition to the data domain. |
| `putEntities` | Add multiple entities with the specified entity keys and definitions to the data domain. |

| Entity operation | Description |
|---|---|
| `validateEntity` | Validate an entity (either active or inactive) with the specified entity key and definition. |
| `validateEntities` | Validate multiple entities (either active or inactive) with specified definitions. |
| `listEntities` | List the entities that exist in the data domain. |
| `deleteEntities` | Delete multiple entities for which entity keys are specified. |
| `deleteAllEntities` | Delete all entities that exist in the data domain without specifying any of their entity keys. |

For details on using the entity operations, see *Working with Entities on page 231*.

## Collection operations

| Collection operation | Description |
|---|---|
| `putCollection` | Add a collection with the specified collection key and unique property key to the data domain. |
| `putCollections` | Add multiple collections with the specified collection keys and unique property keys to the data domain. |
| `updateCollections` | Update the configuration of an existing collection. |
| `listCollections` | List the collections that exist in the data domain. |
| `deleteCollections` | Delete one or more collections, as specified by their collection keys. |
| `deleteAllCollections` | Delete all collections that exist in the data domain without specifying any of their collection keys. |

For details on using the collection operations, see *Collections on page 92*.

## Filter rule operations

| Filter rule operation | Description |
|---|---|
| `putFilterRule` | Add a filter rule with the specified filter rule key and rules to the data domain. |
| `putFilterRules` | Add multiple filter rules with the specified filter rule keys and rules to the data domain. |
| `listFilterRules` | List the filter rules that exist in the data domain. |

| Filter rule operation | Description |
|---|---|
| `deleteFilterRules` | Delete one or more filter rules, as specified by their filter rule keys. |
| `deleteAllFilterRules` | Delete all filter rules that exist in the data domain without specifying any of their filter rule keys. |

For details on using the filter rule operations, see *Filter Rules on page 109*.

## Language ID for EQL parsing error messages

The operations have an optional `Language` element that sets the language for error messages that result from EQL parsing. For example, the general syntax of the `putEntity` operation is:

```
<putEntity>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
   <semanticEntity key="?" displayName="?" isActive="?">
    ...
   </semanticEntity>
</putEntity>
```

For details on this element and its supported language codes, see the description of the `Language` element in EQL *Language codes for EQL error messages on page 127*.

Chapter 6

# Transaction Web Service Interface

This section describes the operations and behavior of the Transaction Web Service.



## About outer transactions

An ***outer transaction*** is a set of operations performed in an Endeca data domain 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 index.

If any of the changes made within an outer 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 in the index do not occur.

**Note:** If an outer transaction fails to complete successfully due to a Dgraph failure, then it is not applied (and does not need to be explicitly rolled back).

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 the Endeca Server is used for controlling outer transactions.

# When to use outer transactions

This topic discusses outer transactions and provides recommendations for when it is useful to issue queries and updates inside an outer transaction as opposed to running individual queries for various tasks.

Typically, you use Integrator ETL or another data loading mechanism to load data and configuration into Endeca Server, for a specific data domain. You load data by making Web service requests to the Data Ingest Web Service or requests to the Bulk Load Interface. Each Web service request represents its own set of operations in the data domain, 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 domain succeed and others fail, the resulting data domain 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 set of data-updating requests to the data domain hosted in Endeca Server either complete or fail as a unit, so that the resulting set of index files represents an entirely updated data domain. You may also want to make sure that end users do not access intermediate states of the data in the front-end application, but instead can only have access to the pre-update state of the index files (while the data-updating graph completes), and then seamlessly transition to querying the data domain that has been fully updated.

To guarantee that your updates either completely succeed or fail, make your requests inside an outer transaction.

An ***outer transaction*** is a set of operations performed in the data domain 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 data domain's index.

To run an outer transaction, you can either use Integrator ETL or issue requests with the Transaction Web Service. This way, you can run inner transactions inside an outer transaction. Typically, running inner transactions (each of which represents a request to the Dgraph) inside an outer transaction is useful for running updates. Once such an outer transaction completes, an update to your records is guaranteed to be fully committed to the index of your data domain.

# About the Transaction Web Service

The Transaction Web Service provides a versioned interface for controlling one or more inner transactions on a particular data domain, inside a single outer transaction.

Each Web service request to Endeca Server represents an inner transaction. If the request completes successfully, the transaction is automatically committed. If the request fails, the transaction is rolled back.

In addition to using these inner transactions that are sent as independent Web service requests, you can also nest inner transactions inside a single outer transaction run by the Transaction Web Service.

The Transaction Web Service is a versioned Web service declared in its WSDL document, which you can access at this URL:

```
http://host:<port>endeca-server/ws/transaction/<DataDomain>?wsdl
```

where `host` and *`port`* represent Endeca Server, and the *`DataDomain`* is the name of the data domain hosted on the server. The WSDL that is returned contains a version number for the service.

Using outer transactions depends on whether you want to group multiple updates (which, together with other web service requests, typically represent simple inner transactions) into a single outer transaction.

The Transaction Web Service enables you to isolate updates within a single outer transaction, while non-updating queries continue to be processed against the pre-transaction version of the data domain's index.

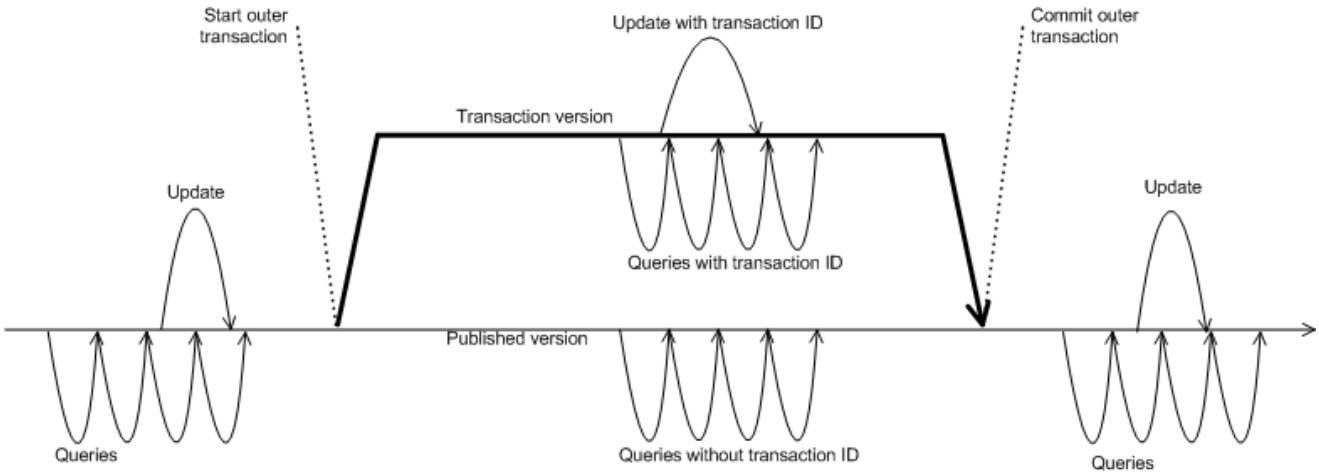
Although you can issue requests to the Transaction Web Service with any Web service tool, such as soapUI, you can use the **Transaction RunGraph** in Integrator ETL.

# Outer transactions and queries

Endeca Server processes two types of queries — non-updating (or read-only) queries, and updating queries.

- The purpose of non-updating queries is, typically, to obtain query results from the index, based on search or navigation selections made by the end users in the front-end application. Non-updating queries represent read-only requests to the index and do not attempt to change them.
- The purpose of updating queries is to change the index or other settings in the data domain or the Dgraph process. Updating queries represent "write" requests to the index.

The following diagram shows how both updating and non-updating queries are processed by the Endeca Server in view of outer transactions. It illustrates that, to be processed within an outer transaction, updating queries must specify its ID. Non-updating queries, depending on whether they specify the outer transaction ID, are processed against different versions of the index:




The following statements describe the actions in this diagram in detail, starting from the left side of the diagram:

- **Stage 1: Before an outer transaction is started**. If no outer transaction is in progress, then all queries (updating and non-updating) are processed against the most recent published version of the index. If no outer transaction is in progress, the queries do not need to specify any outer transaction ID.
- **Stage 2: The outer transaction has been started**. Queries that specify a correct outer transaction ID (also known as queries sent inside the transaction):
  - All such queries (updating and non-updating) are guaranteed to run against the most recent internal version of the index available to the transaction. This version is not published and not available to queries outside of the outer transaction until the transaction commits. This version is published after the transaction commits.

Queries that do not specify an outer transaction ID (also known as queries sent outside of the transaction):

- Non-updating queries run against the most recent published (pre-transaction) version of the index.
- Updating queries wait until the outer transaction is committed (or rolled back), and are computed based on the published version of the index available at that time.

If the outer transaction has been started and requests sent to the Endeca Server specify an incorrect ID, the requests fail.

- **Stage 3: After the outer transaction has been committed**. All queries (updating and non-updating) are processed against the most recent published version of the index.

# Transaction Web Service operation description

This topic describes the logic of the Web service's operations, as well as its request and response structure.

## Operation description

Here is the logic of the Web service's operations:

- Before an outer transaction starts, the pre-transaction version of the index is available; it is known as the last published version of the index.
- When a `startOuterTransaction` operation is issued, it starts an outer transaction. This transaction, because it encapsulates inner transactions within it, is referred to as an outer transaction. Only one outer transaction can be running at a time.

  You can supply the ID for the transaction in the `startOuterTransaction` operation. If you do not specify it, the Dgraph process issues a unique outer transaction ID in the response.

  While the outer transaction is in progress, update requests to the index that reference the outer transaction ID are processed within the outer transaction. These updates can be made through any of the available interfaces that can issue requests to the server, including the Data Ingest Web Service, the Configuration Web Service, and the Bulk Load Interface. Updating requests from all interfaces except the Bulk Load that do not reference the ID wait until the outer transaction is committed (or rolled back), and are computed based on the published version of the index. (Requests from the Bulk Load interface that don't specify the ID are rejected while the outer transaction is in progress).
- Once an outer transaction starts, the index files are internally updated. Internal versions of the index might become available to qualifying requests (those that reference the ID) within an outer transaction. These versions are known as transaction versions.
- For the duration of the outer transaction, the Dgraph answers updating queries only if they specify the transaction ID. These updating queries are answered against a transaction version of the index that is not published until the transaction is committed. (This transaction version reflects the most recent writes to the index that occurred within the outer transaction until this point.) All non-updating queries are answered either against the last published version of the index (if they do not specify the ID), or against the transaction version (if they specify the ID).
- The outer transaction is committed with the Transaction Web Service `commitOuterTransaction` operation.
- Once the transaction commits, its version of the index is published. Updating requests that were waiting in the queue (and that did not specify the ID) are now processed against this version.

- If an outer transaction fails to commit, it remains open. You cannot start another outer transaction until you commit or back the outer transaction that failed to commit. You can manually issue a commit or rollback operation to recover from a failed transaction without restarting the Dgraph process for the data domain. (This statement has one exception — if an outer transaction fails to commit because the Dgraph fails, the transaction is not applied and does not require to be committed or rolled back).

  To manually end a transaction that failed to commit and roll back the changes, you can issue the `rollBackOuterTransaction` operation, specifying the ID of this outer transaction. If you roll back an outer transaction, then updating requests that didn't specify the ID and that were waiting in the queue are processed once the transaction is successfully rolled back.

  For information on connectors that support transactions, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

### Request

The input to the Web service depends on the operation used and can include any of its operations for starting, committing, or rolling back a transaction, or for listing the outer transaction ID.

In the `startOuterTransaction` operation, you can provide a transaction ID. If it is not provided, the Dgraph issues an ID automatically and returns it in the response. In addition, you can use the `listOuterTransaction` operation to obtain the ID from the Dgraph running in your data domain.

### Response

The response to the Transaction Web Service indicates whether each of the operations succeeded or failed.

If any operation does not succeed, the whole Web service transaction returns a SOAP fault and none of the operations are applied.

## Transaction Web Service operations

This topic lists the operations available in the Transaction Web Service.

A request to the Transaction Web Service consists of a `Request` element.

The operations are the following:

| Operation | Description |
|---|---|
| `startOuterTransaction` | An operation to begin an outer transaction. You can optionally provide a transaction ID in the `OuterTransactionId` element.<br><br>If this operation succeeds, it starts the outer transaction and returns a transaction ID, and the Dgraph enters transaction mode.<br><br>If this operation does not succeed, the Dgraph does not start an outer transaction, and does not return a transaction ID.<br><br>While an outer transaction is in progress, the following actions take place:<br><br>• All queries that reference the transaction ID are processed within the transaction. Updating queries that do not reference the transaction ID wait until the outer transaction is committed (or rolled back) and are computed based on the published transaction version of the index.<br><br>• Read-only queries that do not reference the transaction ID are not rejected — they are processed against the published version of the index.<br><br>Updates applied within the outer transaction do not become a published version of the index until another operation, `commitOuterTransaction`, returns successfully. |
| `listOuterTransaction` | An operation to request an ID of a running outer transaction. If an outer transaction is in progress, this operation returns its ID. |
| `rollBackOuterTransaction` | An operation to roll back an outer transaction with the ID specified in the `OuterTransactionIdToRollBack` element. |

| Operation | Description |
|---|---|
| `commitOuterTransaction` | An operation to end an outer transaction.<br><br>If an outer transaction with the specified ID is in progress, then if the operation succeeds, the Dgraph commits the transaction and exits transaction mode. The Dgraph resumes accepting unqualified queries. The version of the index that is propagated to all nodes becomes the last published version.<br><br>If the operation does not succeed, the outer transaction is not committed. The Dgraph does not apply any updates that referenced the transaction ID. All queries continue to use the pre-transaction version of the index.<br><br>  **Note:** If the outer transaction fails to commit, it remains open, and you cannot start another outer transaction before committing or rolling back the one that failed. Without stopping the Dgraph, you can manually commit the transaction and roll back any changes using the `rollBackOuterTransaction` operation, specifying the ID of the transaction. If, in another possible scenario, the outer transaction fails to commit because the Dgraph fails, the transaction is not applied and does not need to be rolled back manually. |

# Rolling back an outer transaction

The `rollBackOuterTransaction` operation is useful in operational environments that use outer transactions. In case a running outer transaction fails, this operation lets you roll back to the previously committed version of the index and commit the transaction.

You can run this operation directly with a tool such as soapUI or Integrator ETL. In Integrator ETL, you can use one of two options: either through the **Transaction RunGraph** component and its **rollback** option, or by using the **WebServiceClient** component, by configuring it to access the Transaction Web Service on the particular data domain, and specifying the `rollBackOuterTransaction` operation to it.

The following statements describe the rollBackOuterTransaction operation:

- If you are running this operation through a tool such as soapUI, ensure that the tool accesses the Transaction Web Service as follows:

  ```
  http://localhost:<port>/ws/transaction/<DataDomain>?wsdl
  ```

  Use this operation only if an outer transaction has been started on the node, referencing a transaction ID in the `OuterTransactionIdToRollBack` element, as in the following example:

  ```
  <rollBackOuterTransaction>
     <OuterTransactionIdToRollBack>myID</OuterTransactionIdToRollBack>
  </rollBackOuterTransaction>
  ```

   **Note:** The transaction ID can be either specified to the Transaction Web Service when you start a transaction, or, if you do not specify it, the Web service generates the ID automatically. The operation `listOuterTransaction` lists the ID. Also, if you are using the **Transaction**

**RunGraph** connector for running transactions, this connector automatically uses the ID string `"transaction"`.

- If you issue this operation with the outer transaction ID that does not match the ID of the currently running outer transaction, an error message notifies you of the transaction ID that is in progress.
- If you are using this operation directly (such as in soapUI) and not in the context of Integrator ETL, you can issue it at any point during a running outer transaction. Once issued, this operation ensures that inner transactions running within the outer transaction are rolled back.

  If you have updating requests sent to the server that did not specify the outer transaction ID, these requests wait for the outer transaction to finish (be committed or rolled back). If you roll back the outer transaction, these requests start being processed by the server based on the published version of the data files that is available after the rollback operation.
- If you are running a data domain cluster hosted in the Endeca Server cluster, and issue this operation, it is routed to the Endeca Server hosting the leader Dgraph node for the data domain.

  Once the operation completes, it stops the outer transaction, and the leader resumes serving queries on the last version of the index available before the start of the outer transaction.
- Only one `rollBackOuterTransaction` operation can be processed at a time.

# Notes about inner transactions

This topic discusses the treatment of requests and operations that occur within a single outer transaction.

***Inner transactions*** are operations that occur within a single outer transaction. They represent either Web service requests or administrative operations that run within an outer transaction. A few considerations about inner transactions are useful to note:

- All non-updating inner transactions that specify the outer transaction ID are processed against the most recent internal version of the index available within the outer transaction.
- Inner transactions with updates are always serial —they are applied in the order they are received.
- Read-only inner transactions are processed in parallel.

# Request processing in the presence of transactions

This topic describes how requests sent from various Web services, administration URL requests, and the Bulk Load Interface are treated by Endeca Server, in the presence of outer transactions.

Requests from the Endeca Server interfaces can be updating and non-updating (read-only).

For read-only requests, if the correct outer transaction ID is specified as the first element in the request, they run inside the outer transaction against the most recent version of the index. If no transaction ID is specified, the requests run against the pre-transaction version.

For updating requests, if the correct ID is specified as the first element in the request, the requests run; otherwise, the requests wait until the outer transaction is committed (or rolled back), and are computed based on the published version of the index that becomes available at that time.

An incorrect ID in any request results in a SOAP fault.

# Transaction Web Service and Integrator ETL

In Integrator ETL, you can start and commit an outer transaction using the **Transaction RunGraph** connector.

For information on how to run outer transactions in Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

# Performance impact of transactions

Running an outer transaction does not affect performance of 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 7

# Web Service Versioning

This section discusses versions used for the Web service interfaces in Endeca Server.



## How version numbers are assigned

Interface version numbers are assigned each time the particular interface is updated.

A version number for an interface consists of major and minor versions:

- Major version is used to track changes to the service that are not backward-compatible.
- Minor version is used to track backward-compatible changes to the service.

For example, if a version number for a Web service is 3.0, its major version is 3, and its minor version is 0.

All interface changes result in a version number increase:

- Major version numbers are increased only when changes that are not backward-compatible are introduced in the interface. These changes include removal of operations, elements, or attributes; addition of new required attributes to existing operations; or cases when services are split or combined, or operations are moved from one service to another.

  Changes that are not backward-compatible are introduced with the following deprecation policy. When any of the interface's artifacts change, new artifacts are added, but old ones are not removed in the new version. Instead, old artifacts are deprecated and retained for a period of time.

  The *Oracle Endeca Server Migration Guide* lists the following:

  - Which service versions are shipped with the particular Oracle Endeca Server release.
  - Which operations and/or services have been deprecated.
  - The upgrade impact for each of the changes.
- Minor version numbers are increased when backward-compatible changes are introduced. Backward-compatible changes include new operations that may be added, or new operations with new types.

Interface version numbers for each of the Web service interfaces may differ and depend on the changes to that interface.

Interface version numbers do not correspond to the version number of the product that is being released.

After upgrading to a new version of the product, it is recommended to check the version numbers of each of the interfaces and ensure that your clients also have the corresponding versions.

# Obtaining a version number for a Web service

To request a version number for a Web service interface, obtain the WSDL document for the web service from a running data domain in Endeca Server. The WSDL document indicates the version of a Web service.

You can obtain a WSDL document for a specific Web service once an Endeca data domain is running.

Each Web service has its own version. If a Web service has more than one minor version (for example, 3.0 and 3.1), then the newer version is backward-compatible with the older version, and all these versions are listed in the WSDL document.

To obtain the version number from a Web service:

1. Issue a request to Endeca Server for the WSDL document of the web service on a specific data domain, using this syntax:

   ```
   http://host:port/endeca-server/ws/WSName/<dataDomain?wsdl
   ```

   where *host* and *port* represent Endeca Server, *WSName* is the name of the Web service, and *<dataDomain>* is the name of the data domain created on the server.

   For example:

   ```
   http://localhost:7001/endeca-server/ws/config/books?wsdl
   ```

   In the WSDL document, the major and minor versions are displayed in one of the required namespaces, as follows:

   ```
   xmlns:config-service-v3_0="http://www.endeca.com/MDEX/config/services/config/3/0"
   ```

   In this example, 3 is the major version, and 0 is the minor version.

   If there is more than one minor version, then all of them are supported and listed in the WSDL document. The most recent version is backward-compatible with the other minor versions listed in the WSDL document.

2. Repeat the process for any other Web service whose version you need to verify, using the name of any of the available Web services.

# Using version numbers in requests

Version numbers are specified in Web service requests as a required namespace.

The following statements describe how versions of the Web service interfaces affect interaction with them:

- When the client sends a version in its request, the server sends an API version in its response.
- The version of a client (such as a set of Java methods generated from contacting a service) must be compatible with the version of the Web service.

If clients contact a service whose version is incompatible with the version in their stubs, they receive a SOAP fault. Specifically, the following cases are possible:

- If version A is specified in the client stubs, but version B is used in the Web service, and version B is backward-compatible with version A, the request is processed normally without any messages.
- If no version is specified in the client stubs, but a version exists in the Web service, this is interpreted as a parsing error: `Unable to parse version for <operation_name> in namespace <namespace_name>`.
- If version A is specified in the client stubs, but version B is used in the Web service, and version B is not backward-compatible with version A (that is, it represents a major version change), the Dgraph issues an error indicating that the version used is invalid; Endeca Server issues an error indicating that it could not find the specified version A in the web service WSDL available to the server.

**Important:** In all cases, to fix the client's incompatibility with the current Web service version, generate new client stubs and use them with the front-end application.

### Examples of specifying the version numbers in requests

These examples illustrate how to specify the version number in requests to various Web services. The principle for specifying the version is the same, but the syntax differs slightly depending on the type of the Web service.

In this example, the request is sent to the Configuration Web Service, with specified values for major and minor versions in the namespace (3.0):

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
   ...
   </soapenv:Body>
</soapenv:Envelope>
```

In this example, the request is sent to the Conversation Web Service:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>
      ...
      </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The response also contains version numbers.

# Backward-compatibility of Web service versions

Changes between minor versions are backward-compatible. Changes between major versions are not.

If a minor version change has occurred for an interface, you can continue using the previous minor version, if this version is listed in the current WSDL document (this indicates the backward-compatibility of a particular minor version).

However, a good practice is to keep the versions used by clients updated to the latest versions of the Web services installed with Endeca Server.

# Resolving incompatibility of Web services and client stubs

The client stubs generated from the Web services must be installed in various other front-end applications that communicate with the Oracle Endeca Server. When you upgrade Endeca Server, to ensure compatibility with the newer versions of the interfaces, regenerate the client stubs.

To resolve incompatibility of Web service versions and client stubs:

1. Install new Web service versions.

   This is typically done as part of an upgrade to the Endeca Server software.

2. Query each interface for its version to check which interfaces have major number changes (these changes are not backward-compatible).

   Read the *Migration Guide* for the upgraded version of the Oracle Endeca Server to learn about changes to the Web service interfaces.

3. Regenerate the client stubs. If the major version had changed for any interface, you must regenerate the client stubs. If only the minor version had changed, it is still recommended to regenerate the client stubs, although you can continue to use your existing clients generated against the previous minor versions.

   When the stubs are compiled, the new version of the interface is read from the Web service's WSDL.

   **Note:** If you are upgrading from an interface without a version to an interface with a version, the initial upgrade to the client stubs requires changing all import statements in your client code, similar to the following example.

   If the import statement in the client code using stubs generated from an unversioned web service looked similar to this example:

   ```
   import com.endeca.www.mdex.transaction._2011.startOuterTransactionDocument;
   ```

   change the import statement to indicate the versions:

   ```
   import com.endeca.www.mdex.transaction._2._0.startOuterTransactionDocument;
   ```

   The namespaces for each operation indicate which versions are supported for this operation. In this case, the `startOuterTransaction` operation is supported in version 2.0.

4. Start using the new stubs in your front-end application to send requests to your Endeca data domain.

# Part III

## Collections, Record Filters, and Records

Chapter 8

# Collections

This section describes how to create, manage, and use collections.



## About collections

In the Endeca Server, collections represent a data model in which source records are partitioned into named collections, according to their unique key assignments, and then loaded into the Endeca Server.

Collections allow you to divide the data in a given data domain into multiple organized groupings (known in Studio as data sets). You can therefore build an Endeca Server application with multiple collections of records, all comprising records in a single data domain's index.

Collections have a practical meaning. It is very common to have multiple different kinds of data that users want to search through: products for sale alongside how-to articles; vehicles alongside warranty claims; structured HR records of employee changes alongside satisfaction surveys; and so on. These different kinds of records are typically related and relevant to each other, but are often not useful to see mixed up with each other. It would not be very useful, for example, to see a results list in a UI where some of the rows represented products for sale and others represented data sheets. Collections allow you to load and organize data, for each data domain, inside them, by their logical groupings.

Keep in mind that collections are optional in your Endeca Server application. Endeca records are not required to be members of a collection, but you can use collections if this approach represents your data model.

**Note:** At least one collection (which is called a data set in Studio) must be present in an Endeca data domain before a Studio application can be configured to connect to that data domain. For this reason, you should always ingest your source data into one or more collections in a Studio environment.

### Sample use case

Looking at the application from a high level, the Conversation Web Service has knowledge of multiple collections of data in a single Endeca Server data domain. This means that the query state is tracked per-collection, and most content elements (such as `NavigationMenu` and `RecordList`) can operate on a single focal collection.

For example, consider an application for exploring product sales and online reviews. Imagine that Sales and Reviews are two collections of records. You might have two different tabs in your application: one for exploring Reviews, and the other for exploring Sales. Further, you might make different filters on the two tabs, so that

you can narrow your Review records to only five-star reviews, while Sales records are limited by a text search on product description. As you switch between tabs, the single State includes the separate selections for both kinds of records, so selections for both kinds of records are not lost.

On each tab, you see only data relevant to the collection in question: on the Reviews tab, you see a menu of available refinements containing values that are useful refinements for narrowing your Reviews (but not, notably, attributes that only appear on Sales records). You also see a record list with a page of just Review records without Sales records being mixed in.

### Cross filtering

When switching between navigating over the Reviews records and navigating over the Sales records, it can be useful to carry some filters between the two. For example, though a selection on 5-star Reviews only should not filter Sales, other attributes may actually be shared. For example, if user Jane has filtered her Reviews to only reviews written by John Doe, she may also want to automatically narrow her Sales to only purchases made by John Doe. Endeca Server supports these cross filtering scenarios with filter rules. For more information, see *Filter Rules on page 109*.

Filter rules allow you to tie the Reviewer attribute on your Reviews records to the Buyer attribute on your Sales records. Any filter that narrows and makes a selection on one of those attributes will implicitly make an additional selection on the other attribute. This means that if you narrow to only Reviews written by John Doe on the Reviews tab, and then switch to your Sales view, the selection of "Buyer=John Doe" will have been automatically added to your Sales breadcrumbs also. Cross-filtering on multiple collections thus enables an important value proposition for Endeca applications: discovery across multiple data sets.

## Collection operations

This section describes the operations used to create and manage collections.

All of the collection operations have optional `outerTransactionId` and `language` elements, as in this syntax for the `listCollections` operation:

```
<listCollections>
   <!--Optional:-->
   <outerTransactionId>?</outerTransactionId>
   <!--Optional:-->
   <language>en</language>
</listCollections>
```

Because their functionality is identical across all the collection operations, the two elements are described here instead of in the descriptions for each operation.

### outerTransactionId element

The optional `outerTransactionId` element specifies the ID of an outer transaction (if it has been started by the Transaction Web Service). It is incorrect to specify an outer transaction ID when an outer transaction is not in progress. All configuration requests with incorrectly specified outer transaction IDs fail with a SOAP fault.

Some of the operations (such as `updateCollections`) can work on multiple collections with the same request. These operations, however, are atomic, which means that either the operation is successful (i.e., all collections are successfully affected), or the entire operation fails if at least one of the collections cannot be modified). To prevent leaving the collection records in a undeterministic state, you can first open an outer transaction, run the operation with the `outerTransactionId` element, and then roll back the transaction if the operation fails.

### language element

The optional `language` element sets the language for error messages that result from EQL parsing. The default language for error messages is `en` (English). For details on this element and its supported language codes, see the description of the `Language` element in EQL: *Language codes for EQL error messages on page 127*.



## Collection create operations

The SConfig Web Service has two operations to create collections.

The `putCollection` operation creates a single collection, while the `putCollections` operation can create multiple collections at once. Both operations result in a CDR (Collection Definition Record) being created in the Dgraph.

### putCollection syntax

The `putCollection` operation creates one collection with this syntax:

```
<putCollection>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
   <collection collectionKey="?" displayName="?" uniquePropertyKey="?">
      <description>?</description>
      <property key="?">?</property>
   </collection>
</putCollection>
```

The meanings of the collection attributes are as follows:

| Collection attribute | Description |
|---|---|
| `collectionKey` | Required. A unique identifier for the collection. The key name does not have to be in the NCName format, but must be unique among collection names. The name is case-sensitive when used with other collection operations or in queries. |
| `displayName` | Optional. Defines the display name which may be used by the front-end application, such as Studio. The display name can be any arbitrary string. |

| Collection attribute | Description |
|---|---|
| `uniquePropertyKey` | Required. Sets the standard attribute that provides the unique key values for records in the collection. Once a `uniquePropertyKey` is configured for a collection, it cannot be used for any other collection. In addition, this standard attribute must not already be assigned on any record.<br>The PDR for the standard attribute must be created with:<br>• `mdex-property_IsSingleAssign` set to `true`<br>• `mdex-property_IsUnique` set to `true`<br>Note that the this standard attribute will also serve as the primary key (record spec) for the records in this collection. |
| `description` | Optional. Provides descriptive text about the collection. |
| `property key` | Optional. Lets you associate a string metadata property for the collection (for example, locale information). The key name must be in the NCName format |

## putCollection example

The following `putCollection` soapUI example creates a collection named Products:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:putCollection>
         <ns:language>en</ns:language>
         <ns:collection collectionKey="Products" displayName="Product data" uniquePropertyKey
="ProductID">
            <ns:description>product records for the region</ns:description>
            <ns:property key="Region">New England</ns:property>
         </ns:collection>
      </ns:putCollection>
   </soapenv:Body>
</soapenv:Envelope>
```

The existing ProductID standard attribute is used as the collection's unique property key. The collection also has an associated metadata that defines the Region property metadata as the New England region.

## putCollections operation

The `putCollections` operation lets you create multiple collections at once, using this syntax:

```
<putCollections>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
   <collection collectionKey="?" displayName="?" uniquePropertyKey="?">
      <description>?</description>
      <property key="?">?</property>
   </collection>
</putCollections>
```

You can supply multiple `collection` elements, which have the same syntax and meanings as in the single-use `putCollection` operation.

### putCollectionResponse

The response for a successful `putCollections` operation looks like this example:

```
<putCollectionResponse xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <collectionAdditionInformation numCollectionsAdded="1" numCollectionsReplaced="0"/>
</putCollectionResponse>
```

The response shows that one new collection was added.

## Collection update operation

The `updateCollections` operation lets you update collection configurations.

You can update the configuration of a collection, with the following restrictions and behavior:

- The collection key (the `collectionKey` attribute) cannot be changed. Note that you must specify the collection key in the request so that Endeca Server can identify the collection to be updated.
- The display name (the `displayName` attribute) can be changed. If it is not used in the request, the previous value is kept.
- The collection unique property key (the `uniquePropertyKey` attribute) cannot be changed. Note that you do not have to specify the collection unique property key in the request.
- The description (the `<description>` element) can be changed. If it is not used in the request, the previous value is kept.
- All metadata properties are first deleted by the operation, and then any that are specified (via the `<property key>` element) are added. This means that if you have a metadata property that you do not want to change but want to retain, you must re-specify it in the request.

Note that you cannot create new collections with this operation; you can only update existing collections.

The `updateCollections` operation lets you update multiple collections with the same request. To do so, use multiple `collectionUpdate` elements within the `updateCollections` structure. Keep in mind, however, that an `updateCollections` operation that updates multiple collections is atomic, which means that either the entire operation is successful, (that is, all collections are successfully updated), or the entire operation fails if at least one of the updates fail (that is, none of the collections is updated).

### updateCollections syntax

The `updateCollections` syntax is:

```
<updateCollections>
    <outerTransactionId>?</outerTransactionId>
    <language>en</language>
   <collectionUpdate collectionKey="?" displayName="?" uniquePropertyKey="?">
      <description>?</description>
      <property key="?">?</property>
   </collectionUpdate>
</updateCollections>
```

The `outerTransactionId` and `language` attributes are optional.

The meanings of the collection attributes are:

| collectionUpdate attribute | Description |
|---|---|
| `collectionKey` | Required. The name of the collection to be updated. Note that you cannot change the name of the collection key. |
| `displayName` | Optional. If this attribute is used, the display name is changed to the specified value. If omitted, the last assignment is kept. |
| `uniquePropertyKey` | Optional. The name of the collection unique property key. If this attribute is omitted or left blank, the last assignment is kept. Note that you cannot change the name of the unique property key. |
| `description` | Optional. If this attribute is used, the collection's description is changed to the specified value. If omitted, the last assignment is kept. |
| `property key` | Optional. You can use this attribute as follows:<br>• If no `property key` is used, then all existing metadata properties are deleted from the collection record.<br>• If a `property key` is used, the metadata property is added to the collection record. |

## updateCollections example

The following `updateCollections` soapUI example updates an existing collection named SalesRecs:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:updateCollections>
         <ns:language>en</ns:language>
         <ns:collectionUpdate collectionKey="SalesRecs" displayName="Sales Order Records">
            <ns:description>All sales order records for the company</ns:description>
            <ns:property key="Locale">US East Coast</ns:property>
         </ns:collectionUpdate>
      </ns:updateCollections>
   </soapenv:Body>
</soapenv:Envelope>
```

The example updates the collection's display name and description, and adds the Locale metadata property. Note that the collection's `uniquePropertyKey` parameter is not used for the request.

## updateCollections response

The `numCollectionsUpdated` element in the `updateCollectionsResponse` message indicates the number of collections that were successfully updated, as shown in this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:updateCollectionsResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:numCollectionsUpdated>2</ns2:numCollectionsUpdated>
```

```
      </ns2:updateCollectionsResponse>
   </env:Body>
</env:Envelope>
```

In the example, two collections were updated. If the `updateCollections` operation was not successful, the `numCollectionsUpdated` element will have 0 (zero) as its value.

## Collection list operation

The `listCollections` operation returns a list of all the collections in a data domain.

### listCollections syntax

The syntax for the `listCollections` operation is:

```
<listCollections>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
</listCollections>
```

The `outerTransactionId` and `language` attributes are optional.

### listCollections example

The following is an example of making a `listCollections` call from soapUI:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:listCollections>
         <ns:language>en</ns:language>
      </ns:listCollections>
   </soapenv:Body>
</soapenv:Envelope>
```

### listCollections response

The list of returned collections might look like this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:listCollectionsResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:collectionRecord collectionKey="ProductRecs" displayName
="Product records" uniquePropertyKey="ProductId">
            <ns2:description>Collects product records in the system</ns2:description>
            <ns2:property key="PriceUnits">US dollars</ns2:property>
            <ns2:collectionAttributes/>
         </ns2:collectionRecord>
         <ns2:collectionRecord collectionKey="SalesRecs" displayName="Sales orders" uniquePropertyKey
="SalesOrderCol">
            <ns2:description>Collects the sales order records in the system</ns2:description>
            <ns2:property key="Locale">New England</ns2:property>
            <ns2:collectionAttributes/>
         </ns2:collectionRecord>
      </ns2:listCollectionsResponse>
   </env:Body>
</env:Envelope>
```

The operation shows that the data domain currently has two collections defined: ProductRecs and SalesRecs.

## Collection delete operations

There are two operations to delete collections.

The two delete operations are:

- `deleteCollections` deletes one or more collections, as specified by their collection keys.
- `deleteAllCollections` deletes all the collections at once, with no specification of their collection keys.

Note that there are no software pre-requirements for deleting a collection (that is, a collection can be deleted at any time). However, it is highly recommended that you first delete the records in a collection before deleting the collection itself, as described in *Deleting a collection and its records on page 101*.

### Deleting specific collections

The `deleteCollections` operation allows you to delete one or more collections as specified by their collection keys. The syntax is:

```
<deleteCollections>
    <outerTransactionId>?</outerTransactionId>
    <language>en</language>
    <collectionKey collectionKey="?"/>
</deleteCollections>
```

The `collectionKey` element specifies the collection key of the collection to be deleted (note that collection names are case sensitive). The `outerTransactionId` and `language` attributes are optional.

The `deleteCollections` operation lets you delete multiple collections with the same request. To do so, use multiple `collectionKey` elements within the `<deleteCollections>` structure. Keep in mind, however, that a `deleteCollections` operation that deletes multiple collections is atomic, which means that either the operation is successful (i.e., all collections are successfully deleted) or the entire operation fails if at least one of the deletes fail (i.e., none of the collections is deleted).

This soapUI example deletes two collections:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:deleteCollections>
         <ns:language>en</ns:language>
         <ns:collectionKey collectionKey="EmpRecs"/>
         <ns:collectionKey collectionKey="ArchivedRecs"/>
      </ns:deleteCollections>
   </soapenv:Body>
</soapenv:Envelope>
```

If the request is successful, the `numCollectionsDeleted` element in the `deleteCollectionsResponse` message indicates the number of collections that were successfully deleted, as shown in this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:deleteCollectionsResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:numCollectionsDeleted>2</ns2:numCollectionsDeleted>
      </ns2:deleteCollectionsResponse>
   </env:Body>
```

```
</env:Envelope>
```

In the example, two collections were deleted. If the `deleteCollections` operation was not successful, the `numCollectionsDeleted` element will have 0 (zero) as its value.

### Deleting all collections at once

The `deleteAllCollections` operation allows you to delete all your existing collections at once, without specifying their collection keys. The syntax for this operation is:

```
<deleteAllCollections>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
</deleteAllCollections>
```

The `outerTransactionId` and `language` attributes are optional.

If the request is successful, the `numCollectionsDeleted` element in the `deleteAllCollectionsResponse` message indicates the number of collections that were successfully deleted, as shown in this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:deleteAllCollectionsResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:numCollectionsDeleted>3</ns2:numCollectionsDeleted>
      </ns2:deleteAllCollectionsResponse>
   </env:Body>
</env:Envelope>
```

In the example, three collections were deleted. If the `deleteAllCollections` operation was not successful, the `numCollectionsDeleted` element will have 0 (zero) as its value.

### Deleting collections with entity dependencies

If you delete a collection that has an entity that depends on the collection, then the entity's `isValid` attribute will be set to `false`. In this case, you should update the configuration of the entity to remove the dependency on the deleted collection.

## Deleting a collection and its records

This procedure describes how you first delete a collection's records and then delete the collection itself.

The procedure below uses soapUI to make the various API calls. However, you can create an Integrator ETL graph that performs the same actions.

In the procedure, the actual deletion of the collection records is performed by the `deleteRecords` operation of the Data Ingest Web Service:

```
<ingestChanges>
   <deleteRecords>
      <recordSpecifier>ResellerKey IS NOT NULL</recordSpecifier>
   </deleteRecords>
</ingestChanges>
```

The EQL statement for the `recordSpecifier` element will use the collection's unique property key as a filter for the records in the collection. That is, all records that have a non-NULL assignment for the unique property key will be selected for deletion.

To delete a collection's records and then the collection itself:

1. Use the `listCollections` operation to obtain the name of the collection's unique property key.

   In this sample response, the ResellerKey attribute is the unique property key of the Resellers collection that is going to be deleted:

   ```
   <collectionRecord collectionKey="Resellers" displayName="reseller data" uniquePropertyKey
   ="ResellerKey">
   ```

2. Optionally, you can start an outer transaction for the delete operation, as in this example that uses the Transaction Web Service's `startOuterTransaction` operation:

   ```
   soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:ns="http://www.endeca.com/MDEX/transaction/1/0">
      <soapenv:Header/>
      <soapenv:Body>
         <ns:startOuterTransaction>
            <ns:OuterTransactionId>50</ns:OuterTransactionId>
         </ns:startOuterTransaction>
      </soapenv:Body>
   </soapenv:Envelope>
   ```

   If successful, the response should be similar to this example:

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header/>
      <soapenv:Body>
         <transaction:startOuterTransactionResponse xmlns:transaction="http://www.endeca.com
   /MDEX/transaction/1/0">
            <Started xmlns="http://www.endeca.com/MDEX/transaction/1/0">true</Started>
            <OuterTransactionId xmlns="http://www.endeca.com/MDEX/transaction/1/0">50<
   /OuterTransactionId>
         </transaction:startOuterTransactionResponse>
      </soapenv:Body>
   </soapenv:Envelope>
   ```

3. Run the `deleteRecords` operation of `ingestChanges` with the unique property key of the collection:

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns="http://www.endeca.com/MDEX/ingest/3/0"
        xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
      <soapenv:Header/>
      <soapenv:Body>
         <ns:ingestChanges>
            <ns:OuterTransactionId>50</ns:OuterTransactionId>
            <ns:Language>en</ns:Language>
            <ns:deleteRecords>
               <ns:recordSpecifier>ResellerKey IS NOT NULL</ns:recordSpecifier>
            </ns:deleteRecords>
         </ns:ingestChanges>
      </soapenv:Body>
   </soapenv:Envelope>
   ```

   If the delete operation was successful, the response should be similar to this:

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header/>
      <soapenv:Body>
         <ingest:ingestChangesResponse xmlns:ingest="http://www.endeca.com/MDEX/ingest/3/0">
            <ingest:numPropertiesCreated>0</ingest:numPropertiesCreated>
            <ingest:numRecordsAffected>0</ingest:numRecordsAffected>
            <ingest:numRecordsDeleted>334</ingest:numRecordsDeleted>
         </ingest:ingestChangesResponse>
      </soapenv:Body>
   </soapenv:Envelope>
   ```

4. Use the `deleteCollections` operation to delete the collection.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:deleteCollections>
         <ns:outerTransactionId>50</ns:outerTransactionId>
         <ns:language>en</ns:language>
         <ns:collectionKey collectionKey="Resellers"/>
      </ns:deleteCollections>
   </soapenv:Body>
</soapenv:Envelope>
```

If the `deleteCollections` operation was successful, the response should look like this:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns3:deleteCollectionsResponse xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
            xmlns:ns3="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns3:numCollectionsDeleted>1</ns3:numCollectionsDeleted>
      </ns3:deleteCollectionsResponse>
   </env:Body>
</env:Envelope>
```

5. Use the `deleteFilterRules` operation to delete the filter rules that reference the now-deleted collection:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:deleteFilterRules>
         <ns:outerTransactionId>50</ns:outerTransactionId>
         <ns:language>en</ns:language>
         <ns:filterRuleKey filterRuleKey="ResellerRule"/>
      </ns:deleteFilterRules>
   </soapenv:Body>
</soapenv:Envelope>
```

If the delete filter rule operation was successful, the response should look like this:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns3:deleteFilterRulesResponse xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
           xmlns:ns3="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns3:numFilterRulesDeleted>1</ns3:numFilterRulesDeleted>
      </ns3:deleteFilterRulesResponse>
   </env:Body>
</env:Envelope>
```

6. If you had started an outer transaction for the delete operation, use the Transaction Web Service's `commitOuterTransaction` operation to commit the transaction:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ns="http://www.endeca.com/MDEX/transaction/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:commitOuterTransaction>
         <ns:OuterTransactionId>50</ns:OuterTransactionId>
      </ns:commitOuterTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

If the commit was successful, the response should look like this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <transaction:commitOuterTransactionResponse xmlns:transaction="http://www.endeca.com
/MDEX/transaction/1/0">
         <Committed xmlns="http://www.endeca.com/MDEX/transaction/1/0">true</Committed>
         <OuterTransactionId xmlns="http://www.endeca.com/MDEX/transaction/1/0">50<
/OuterTransactionId>
      </transaction:commitOuterTransactionResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that procedure does not delete the PDRs of the record attributes. The PDRs do not take up much memory, and having them in the Dgraph will make it easier to re-ingest the source records if you later decide to rebuild the collection. However, it is a good idea to check your search interfaces and attribute groups and remove any attributes that are no longer used.

# Collection Definition Records

A CDR (Collection Definition Record) defines a specific collection in the Dgraph.

A collection is represented in the Dgraph by a Collection Definition Record (CDR). A CDR is created when the `putCollection` and the `putCollections` operations are successful. Although CDRs are not user-visible, the `listCollections` operation in effect returns a representation of the CDR via the `collectionRecord` element.

When the CDR is first created, it looks like this example when shown via the `listCollections` operation:

```
<collectionRecord collectionKey="Products" displayName="Product data" uniquePropertyKey="ProductID">
     <description>product records for the region</description>
     <property key="Region">New England</property>
     <collectionAttributes/>
</collectionRecord>
```

Note that the `collectionAttributes` field is empty, which means that no records are associated with this collection at this time.

After source records are ingested into the data domain, the attributes that are tagged on to collection-related records (i.e., records that have a `uniquePropertyKey` assignment) are added to the CDR with `collectionAttribute` attributes, as in this example from a `listCollections` operation:

```
<collectionRecord collectionKey="Products" displayName="Product data" uniquePropertyKey="ProductID">
   <description>product records for the region</description>
   <property key="Region">New England</property>
   <collectionAttributes>
      <collectionAttribute propertyKey="Class"/>
      <collectionAttribute propertyKey="Color"/>
      <collectionAttribute propertyKey="DaysToManufacture"/>
      <collectionAttribute propertyKey="DealerPrice"/>
      <collectionAttribute propertyKey="FinishedGoodsFlag"/>
      <collectionAttribute propertyKey="ListPrice"/>
      <collectionAttribute propertyKey="ModelName"/>
      <collectionAttribute propertyKey="ProductID"/>
      <collectionAttribute propertyKey="ProductLine"/>
      <collectionAttribute propertyKey="ProductSubcategoryKey"/>
      <collectionAttribute propertyKey="ReorderPoint"/>
      <collectionAttribute propertyKey="SafetyStockLevel"/>
      <collectionAttribute propertyKey="Size"/>
      <collectionAttribute propertyKey="SizeRange"/>
      <collectionAttribute propertyKey="SizeUnitMeasureCode"/>
```

```
      <collectionAttribute propertyKey="StandardCost"/>
      <collectionAttribute propertyKey="Status"/>
      <collectionAttribute propertyKey="Style"/>
      <collectionAttribute propertyKey="Weight"/>
   </collectionAttributes>
</collectionRecord>
```

Now the `collectionAttributes` field has been populated with the record attributes that comprise this collection.

### collectionRecord properties

The `collectionRecord` element has the properties listed in this table. The properties (with the exception of the `collectionAttribute` properties) are initially set by the user via the `collection` attributes of a `putCollection` or `putCollections` operation:

| collectionRecord Property | Description |
|---|---|
| `collectionKey` | The name of the collection, used to specify this collection in all Endeca Server APIs (including queries). Set via the `collection::collectionKey` attribute. |
| `displayName` | An arbitrary string value that can be used by applications, intended for use as a user-friendly display name of the collection. Set via the `collection::displayName` attribute. |
| `uniquePropertyKey` | The standard attribute that determines which records will be part of this collection. Set via the `collection::uniquePropertyKey` attribute. |
| `description` | An arbitrary string value that can be used by applications, intended for use as a description of the collection. Set via the `collection::description` attribute. |
| `property key` | String metadata that is associated with the collection. Set via the `collection::property key` attribute. |
| `collectionAttribute propertyKey` | A standard attribute that belongs to the collection. These properties are added by the Dgraph during a record ingest operation and cannot be modified by the user. |

# Procedure for creating collections in the data domain

This topic provides a high-level overview of the steps necessary to create a collection.

The procedure assumes that your are creating two collections: Products (with the ProductID standard attribute as its unique property key) and Sales (with the SalesID standard attribute as its unique property key). It also assumes that each source record (when ingested) will have an assignment of either the ProductID attribute (in which case it will belong to the Products collection) or the SalesID attribute (in which case it will belong to the Sales collection).

To create a collection:

1. Create an empty Endeca data domain.

   For example, you can use the Endeca Server `create-dd` command.

2. Load the attribute schema (i.e., the PDRs for the standard attributes) into the data domain. In particular, make sure you create the ProductID and SalesID standard attributes as single-assign, unique attributes.

3. Create the collections, as in this example that uses the `putCollections` operation:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:putCollections>
         <ns:language>en</ns:language>
         <ns:collection collectionKey="Products" displayName="Product data" uniquePropertyKey
="ProductID">
            <ns:description>product records for the region</ns:description>
            <ns:property key="Region">New England</ns:property>
         </ns:collection>
         <ns:collection collectionKey="Sales" displayName="Sales data" uniquePropertyKey
="SalesID">
            <ns:description>sales information</ns:description>
            <ns:property key="Currency">$</ns:property>
         </ns:collection>
      </ns:putCollections>
   </soapenv:Body>
</soapenv:Envelope>
```

4. Load the source records into the data domain.

   As mentioned above, each source record should have an assignment of either the ProductID or SalesID attribute, which will also serve as the record spec (primary key) for the records.

When the ingest operation has finished, the results of a `listCollections` operation would return the following list (which is abbreviated for ease of reading):

```
<listCollectionsResponse xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <collectionRecord collectionKey="Products" displayName="Product data" uniquePropertyKey
="ProductID">
      <description>Collection of Product information</description>
      <property key="Locale">US region</property>
      <collectionAttributes>
         <collectionAttribute propertyKey="Color"/>
         <collectionAttribute propertyKey="DealerPrice"/>
         ...
         <collectionAttribute propertyKey="Style"/>
         <collectionAttribute propertyKey="Weight"/>
      </collectionAttributes>
   </collectionRecord>
   <collectionRecord collectionKey="Sales" displayName="Sales data" uniquePropertyKey="SalesID">
      <description>Collection of Sales information</description>
      <property key="Currency">$</property>
      <collectionAttributes>
         <collectionAttribute propertyKey="FactSales_CurrencyKey"/>
         <collectionAttribute propertyKey="FactSales_CustomerPONumber"/>
         ...
         <collectionAttribute propertyKey="FactSales_TotalProductCost"/>
         <collectionAttribute propertyKey="FactSales_UnitPrice"/>
      </collectionAttributes>
   </collectionRecord>
</listCollectionsResponse>
```

The `listCollectionsResponse` shows the two Products and Sales collections that were created in step 3. It also shows that the Dgraph has populated the CDRs with the attributes from records that have an assignment from the unique property key of a collection.

# Using collections in queries

You can specify a collection in a query state.

The `CollectionName` element of the `State` type lets you specify the name of a collection to be used in a query. You can specify a maximum of one collection per state. However, you can use multiple states in a query, with each state potentially specifying a collection name.

Once you associate a collection with a state, then operations performed by that state will reference that collection. For example, a record search in a state with a collection name will search for records only in that collection. However, if the collection has one or more filter rules configured, then a `SelectionFilter` or `SelectedRefinementFilter` query will trigger that filter rule, which will make use of records in the target collection. For more information on filter rules (see *Filter Rules on page 109*).

The query syntax for specifying a collection name is:

```
<Request>
   ...
   <State>
      <Name>?</Name>
      <CollectionName>?<CollectionName>
      ...
   </State>
   ...
</Request>
```

where:

- `Name` is the name of the state.
- `CollectionName` is the name of an existing collection.

If you have a named state, you can then specify it in a `ContentElementConfig` in the query.

## Example of using collections in queries

This query has one state (SalesRecs) that specifies the Sales collection and a second state (Resellers) associated with the Resellers collection. In addition, there is a filter rule that has the Sales collection as the source collection and the Resellers collection as the target collection. The `RecordCountConfig` type is associated with the SalesRecs state and will list the number of records from the query:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <Language>en</Language>
   <State>
      <Name>SalesRecs</Name>
      <CollectionName>Sales</CollectionName>
      <SelectionFilter Id="SalesFltr">
         <filterString>FactSales_SalesAmount > 1000</filterString>
      </SelectionFilter>
   </State>
   <State>
      <Name>ResellerRecs</Name>
      <CollectionName>Resellers</CollectionName>
   </State>
   <RecordListConfig Id="Results">
      <StateName>SalesRecs</StateName>
   </RecordCountConfig>
```

```
</Request>
```

The results from the query may look like this abbreviated example:

```
<cs:Results ...>
   <State ...>
      <Name>SalesRecs</Name>
      <CollectionName>Sales</CollectionName>
      <DataSourceFilter>
         <filterString>"FactSales_ProductKey" IS NOT NULL</filterString>
      </DataSourceFilter>
      <SelectionFilter Id="SalesFltr">
         <filterString>FactSales_SalesAmount > 1000</filterString>
      </SelectionFilter>
   </State>
   <State ...>
      <Name>ResellerRecs</Name>
      <CollectionName>Resellers</CollectionName>
      <DataSourceFilter>
         <filterString>"DimReseller_ResellerKey" IS NOT NULL</filterString>
      </DataSourceFilter>
      <SelectionFilter>
         <filterString>"DimReseller_AnnualSales" > 1000</filterString>
         <AppliedFilterRule>
            <Source FilterId="SalesFltr">
               <StateName>SalesRecs</StateName>
            </Source>
            <TargetPropertyKey>DimReseller_AnnualSales</TargetPropertyKey>
         </AppliedFilterRule>
      </SelectionFilter>
   </State>
   <cs:RecordCount Id="Results">
      <cs:NumRecords>115</cs:NumRecords>
   </cs:RecordCount>
</cs:Results>
```

In the response, note that the ResellerRecs state shows that a `SelectionFilter` implicit filter was run on the Resellers collection. The `AppliedFilterRule` element lists the source (filter ID and state) of the implicit filter.

Note that if you specified a non-existent collection, the query will fail with this fault string:

```
<Fault>
   <faultcode>env:Server</faultcode>
   <faultstring>OES-000190: Collection 'Producs' does not exist.</faultstring>
   <detail>
      <Fault xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
             xmlns:ns3="http://www.endeca.com/MDEX/conversation/3/0"/>
   </detail>
 </Fault>
```

## Automatic addition of data source filter

During processing of a query, Endeca Server will automatically create and add a `DataSourceFilter` to all states that provide a collection name. The filter is similar to the example above:

```
<cs:Results ...>
   <State ...>
      <Name>ResellerRecs</Name>
      <CollectionName>Resellers</CollectionName>
      <DataSourceFilter>
         <filterString>"DimReseller_ResellerKey" IS NOT NULL</filterString>
      </DataSourceFilter>
      <SelectionFilter>
         <filterString>"DimReseller_AnnualSales" > 1000</filterString>
         <AppliedFilterRule>
```

```
            <Source FilterId="SalesFltr">
               <StateName>SalesRecs</StateName>
            </Source>
            <TargetPropertyKey>DimReseller_AnnualSales</TargetPropertyKey>
         </AppliedFilterRule>
      </SelectionFilter>
   </State>
   ...
</cs:Results>
```

The purpose of adding the `DataSourceFilter` with the collection's unique property key ("ProductKey" in the example) is to limit the search to the records in the collection.

## Using collections in EQLConfig queries

A `FROM` clause in an EQL query cannot directly reference a collection, but it can reference a state. However, if the state specifies a collection name, then the collection can be used via the state.

This simple EQL query uses the `EQLConfig` type:

```
<Request>
   <Language>en</Language>
   <State>
      <Name>TotalSales</Name>
      <CollectionName>Sales</CollectionName>
   </State>
   <ns:EQLConfig Id="EQLQuery">
      <StateName>TotalSales</StateName>
      <EQLQueryString>
         RETURN Results AS
         SELECT ARB(FactSales_SalesAmount) AS totalAmount
         FROM TotalSales
         GROUP BY FactSales_OrderQuantity
      </EQLQueryString>
   </EQLConfig>
</Request>
```

The `FROM` clause specifies the TotalSales state as the source. Because the TotalSales state specifies the Sales collection, the records from that collection are used.

Chapter 9

# Filter Rules

This section discusses how filter rules work, and how to create and apply them.



## About filter rules

When filter rules are enabled, filters present in the client request will cause the request to be processed as if it also included other filters that are automatically generated by the filter rules.

Filter rules express a relationship between a source attribute (in the context of a source collection), and a target attribute (in the context of a target collection). A filter rule implements a map from a filter on the source attribute in the source collection to a filter on the target attribute in the target collection. A filter rule is directional; that is, it maps only from the source attribute to the target attribute, but not backwards.

**Note:** In Studio, filter rules are called refinement rules.

Only the `SelectionFilter` and `SelectedRefinementFilter` components support filter rules. (The `DataSourceFilter`, `TextSearchFilter`, and `RecordKind` components do not support filter rules.)

Note that the Conversation Service does not provide a means to specify which filter rules should be applied. Instead, all currently configured filter rules that are applicable to a request will be applied to that request.

### Choosing the source and target attributes

When you refine a source collection by the source attribute of a filter rule, the target collection is also refined by the target attribute. Therefore, filter rules work best when the source and target attributes have the same or similar sets of values.

For example, assume that a Sales collection contains a BikeType attribute that lists the model of a bicycle that was sold. The Resellers collection also contains a BikeModel attribute that contains similar data. You then create a filter rule that uses the Sales BikeType attribute as the source attribute and the Resellers BikeModel attribute as the target attribute. When you then refine the Sales collection by its BikeType attribute, the Resellers collection will likewise be refined by its BikeModel attribute. Programmatically, this means:

```
// This explicit filter is first applied to the Sales collection.
<SelectionFilter Id="SalesFltr">
   <filterString>BikeType = 'mountain'</filterString>
</SelectionFilter>

// This implicit filter is then applied to the Resellers collection.
<SelectionFilter Id="SalesFltr">
   <filterString>BikeModel = 'mountain'</filterString>
```

```
</SelectionFilter>
```

The term "***explicit filter***" means that the `SelectionFilter` filter appears in the state that references the Sales collection, but is not present in the state for the Resellers collection. However, when the query is run, the filter in the Sales state (which uses the source attribute) is implicitly constructed for the Resellers collection (using the target attribute).

For these reasons, the data type of both source and target attributes must be the same (for example, both are `mdex:string`). If you are using the `SelectionFilter` component, then both source and target attributes must have the same `mdex-property_IsSingleAssign` setting (i.e., both must be single-assign attributes or both must be multi-assign attributes).

## Conditions for the application of filter rules

The following conditions must exist in order for a filter rule to be applied in a query:

- The query must be made via the `SelectionFilter` or `SelectedRefinementFilter` components.
- A filter rule must exist that references one source attribute (from as the source collection) and one target attribute (from the target collection).
- The filter rule must have an active status (that is, its `isActive` flag must be set to `true`).
- The query must have one state that references the source collection and a second state that references the target collection.
- The filter rule will be applied only to managed attribute and standard attribute refinement selections.
- A filter rule will be considered to match a filter if the single filter attribute matches the source attribute of the filter rule and the collection associated with the request component containing the filter matches the source collection of the filter rule.
- In a `SelectionFilter` query, only one attribute (managed or standard) can be used in the EQL query statement. The reason is that only one attribute from the explicit query can be substituted in the implicit query. (Note that the `SelectedRefinementFilter` syntax allows only one attribute to be specified.)

To elaborate on the last bullet item, these are valid and invalid examples of `SelectionFilter` queries for filter rules:

```
// Valid because only one attribute (SalesAmount) is used.
// The filter rule is applied.
<SelectionFilter Id="SalesFltr">
   <filterString>SalesAmount > 1000</filterString>
</SelectionFilter>

// Valid because only one attribute (SalesAmount) is used.
// The filter is applied.
<SelectionFilter Id="SalesFltr">
   <filterString>SalesAmount < 100 OR SalesAmount > 500</filterString>
</SelectionFilter>

// Invalid because two attributes (SalesAmount and QuantityOrder) are used.
// The query is processed, but the filter rule is not applied.
<SelectionFilter Id="SalesFltr">
   <filterString>SalesAmount > 1000 OR QuantityOrder > 500</filterString>
</SelectionFilter>
```

Keep in mind that when a filter rule is applied, the request returns two sets of records: one for each collection. It is up to the design of the front-end application as to how to handle the resulting record sets.

For information on how to construct a query that references collections and applies a filter rule, see *Using collections in queries on page 107*.

# Filter rule operations

This section describes the operations used to create and manage filter rules.

All of the filter rule operations have optional `outerTransactionId` and `language` elements, as in this syntax for the `listFilterRules` operation:

```
<listFilterRules>
   <!--Optional:-->
   <outerTransactionId>?</outerTransactionId>
   <!--Optional:-->
   <language>en</language>
</listFilterRules>
```

Because their functionality is identical across all the filter rule operations, the two elements are described here instead of in the descriptions for each operation.

### outerTransactionId element

The optional `outerTransactionId` element specifies the ID of an outer transaction (if it has been started by the Transaction Web Service). It is incorrect to specify an outer transaction ID when an outer transaction is not in progress. All configuration requests with incorrectly specified outer transaction IDs fail with a SOAP fault.

Most of the operations (such as `deleteFilterRules`) can work on multiple filter rules with the same request. These operations, however, are atomic, which means that either the operation is successful (i.e., all filter rules are successfully affected) or the entire operation fails if at least one of the filter rules cannot be modified). To prevent leaving the filter rule records in a undeterministic state, you can first open a transaction, run the operation with the `outerTransactionId` element, and then roll back the transaction if the operation failed.

### language element

The optional `language` element sets the language for error messages that result from EQL parsing. The default language for error messages is `en` (English). For details on this element and its supported language codes, see the description of the `Language` element for the Conversation Web Service in *Language codes for EQL error messages on page 127*.

*Filter rule create operations*

*Filter rules list operation*

*Filter rule delete operations*

## Filter rule create operations

There are two operations to create filter rules.

The `putFilterRule` operation creates or updates a single filter rule while the `putFilterRules` operation can create or update multiple filter rules at once. Both operations result in an FRDR (Filter Rule Definition Record) being created or updated in the Dgraph.

With both operations, if the filter rule already exists, then the operation will work in update mode rather than create mode.

## putFilterRule syntax

The `putFilterRule` operation creates one filter rule with this syntax:

```
<putFilterRule>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
   <filterRule filterRuleKey="?" displayName="?" sourceCollectionKey="?" sourcePropertyKey="?"
        targetCollectionKey="?" targetPropertyKey="?" isActive="?"/>
</putFilterRule>
```

The meanings of the filter rule attributes are as follows:

| Filter rule attribute | Meaning |
|---|---|
| `filterRuleKey` | A unique identifier for the filter rule to be created or updated. For a create mode, the key name does not have to be in the NCName format but must be unique among filter rule names. Required. |
| `displayName` | Defines the display name which may be used by the front-end application such as Studio. The display name can be any arbitrary string. Optional. |
| `sourceCollectionKey` | Specifies the name (a `collectionKey` attribute value) of an existing collection that will be the source collection. Required. |
| `sourcePropertyKey` | Specifies the name (an `mdex-property_Key` attribute value) of an existing PDR that will be the source property. The source property must belong to the source collection. Required. |
| `targetCollectionKey` | Specifies the name (a `collectionKey` attribute value) of an existing collection that will be the target collection. Required. |
| `targetPropertyKey` | Specifies the name (an `mdex-property_Key` attribute value) of an existing PDR that will be the target property. The target property must belong to the target collection. The target property and the source property must be the same data type (`mdex-property_Type`). For example, both can be an `mdex:string` data type. Note that if you make a `SelectionFilter` query, then both the target and source properties must have the same `mdex-property_IsSingleAssign` setting. Required. |
| `isActive` | Specifies a boolean value that determines whether this filter rule is active (`true`) or inactive (`false`). An Inactive filter rule is not used as part of a query. An inactive state thus allows you to save an incomplete filter rule (which you will later correct) or to save the filter rule for later use (at which time you will activate it). A filter rule must be active in order for it to be used in queries. Note that when saving an entity, `isActive` must be explicitly set. Required. |

## putFilterRule example

The following `putFilterRule` soapUI example creates a filter rule named ProductRule:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:putFilterRule>
         <ns:language>en</ns:language>
         <ns:filterRule filterRuleKey="ProductRule" displayName="filter for products and sales"
             sourceCollectionKey="Products" sourcePropertyKey="ProductName"
             targetCollectionKey="Sales" targetPropertyKey="SalesOrderNumber" isActive="true"/>
      </ns:putFilterRule>
   </soapenv:Body>
</soapenv:Envelope>
```

Because the `isActive` is set to `true`, the filter rule is active and can be used in queries.

### putFilterRuleResponse

The response for a successful `putFilterRule` operation looks like this example:

```
<putFilterRuleResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
   <ns2:filterRuleAdditionInformation numFilterRulesAdded="1" numFilterRulesReplaced="0"/>
</ns2:putFilterRuleResponse>
```

For a create operation, the `numFilterRulesAdded` attribute in the response shows the number of new filter rules. For an update operation, the `numFilterRulesReplaced` lists the number of filter rules that were updated.

### putFilterRules syntax

The `putFilterRules` operation lets you create multiple filter rules at once, using this syntax:

```
<putFilterRules>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
   <filterRule filterRuleKey="?" displayName="?"
      sourceCollectionKey="?" sourcePropertyKey="?"
      targetCollectionKey="?" targetPropertyKey="?" isActive="?"/>
</putFilterRules>
```

You can supply multiple `filterRule` elements, which have the same syntax and meanings as in the single-use `putFilterRule` operation.

## Filter rules list operation

The `listFilterRules` operation returns a list of all the filter rules in a data domain.

### listFilterRules operation

The syntax for the `listFilterRules` operation is:

```
<listFilterRules>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
</listFilterRules>
```

The `outerTransactionId` and `language` attributes are optional.

### listFilterRules example

The following is an example of making a `listFilterRules` call from soapUI:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:listFilterRules>
         <ns:language>en</ns:language>
      </ns:listFilterRules>
   </soapenv:Body>
</soapenv:Envelope>
```

### listFilterRules response

The result from a `listFilterRules` operation looks like this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:listFilterRulesResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:filterRule filterRuleKey="ProductRule" displayName="filter for products and sales?"
            sourceCollectionKey="Products" sourcePropertyKey="EnglishProductName"
            targetCollectionKey="Sales" targetPropertyKey="FactSales_SalesOrderNumber" isActive
="true"/>
      </ns2:listFilterRulesResponse>
   </env:Body>
</env:Envelope>
```

The `listFilterRulesResponse` shows that one filter rule (named ProductRule) has been created.

## Filter rule delete operations

There are two operations to delete filter rules.

The two delete operations are:

- `deleteFilterRules` deletes one or more filter rules, as specified by their keys.
- `deleteAllFilterRules` deletes all the filter rules at once, with no specification of their keys.

Note that there are no pre-requirements for deleting a filter rule (i.e., it can be deleted at any time).

### Deleting specific filter rules

The `deleteFilterRules` operation allows you to delete one or more filter rules as specified by their keys. The syntax is:

```
<deleteFilterRules>
    <outerTransactionId>?</outerTransactionId>
    <language>en</language>
    <filterRuleKey filterRuleKey="?"/>
</deleteFilterRules>
```

The `filterRuleKey` element specifies the key name of the filter rule to be deleted. The `outerTransactionId` and `language` attributes are optional.

The `deleteFilterRules` operation lets you delete multiple filter rules with the same request. To do so, use multiple `filterRuleKey` elements within the `<deleteFilterRules>` structure. Keep in mind, however,

that a deleteFilterRules operation that deletes multiple filter rules is atomic, which means that either the operation is successful (i.e., all filter rules are successfully deleted) or the entire operation fails if at least one of the deletes fail (i.e., none of the filter rules is deleted).

This soapUI example deletes two filter rules:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:deleteFilterRules>
         <ns:language>en</ns:language>
         <ns:filterRuleKey filterRuleKey="SalesRules"/>
         <ns:filterRuleKey filterRuleKey="RegionRules"/>
      </ns:deleteFilterRules>
   </soapenv:Body>
</soapenv:Envelope>
```

If the request is successful, the numFilterRulesDeleted element in the deleteFilterRulesResponse message indicates the number of filter rules that were successfully deleted, as shown in this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:deleteFilterRulesResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:numFilterRulesDeleted>2</ns2:numFilterRulesDeleted>
      </ns2:deleteFilterRulesResponse>
   </env:Body>
</env:Envelope>
```

In the example, two filter rules were deleted. If the deleteFilterRules operation was not successful, the numFilterRulesDeleted element will have 0 (zero) as its value.

## Deleting all filter rules at once

The deleteAllFilterRules operation allows you to delete all your existing filter rules at once, without specifying their filter rule keys. The syntax for this operation is:

```
<deleteAllFilterRules>
   <outerTransactionId>?</outerTransactionId>
   <language>en</language>
</deleteAllFilterRules>
```

The outerTransactionId and language attributes are optional.

If the request is successful, the numFilterRulesDeleted element in the deleteAllFilterRulesResponse message indicates the number of filter rules that were successfully deleted, as shown in this example:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:deleteAllFilterRulesResponse xmlns:ns2="http://www.endeca.com/endeca-server/sconfig/3/0">
         <ns2:numFilterRulesDeleted>3</ns2:numCollectionsDeleted>
      </ns2:deleteAllFilterRulesResponse>
   </env:Body>
</env:Envelope>
```

In the example, three filter rules were deleted. If the deleteAllFilterRules operation was not successful, the numFilterRulesDeleted element will have 0 (zero) as its value.

# Filter Rule Definition Records

An FRDR (Filter Rule Definition Record) defines a specific filter rule in the Dgraph.

A filter rule is represented in the Dgraph by a Filter Rule Definition Record (FRDR). An FRDR is created when the `putFilterRule` and `putFilterRules` operations are successful. Although FRDRs are not user-visible, the `listFilterRules` operation in effect returns a representation of the FRDR via the `filterRule` element.

The following is example from a `listFilterRules` operation:

```
<ns3:listFilterRulesResponse xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
   xmlns:ns3="http://www.endeca.com/endeca-server/sconfig/3/0">
   <ns3:filterRule filterRuleKey="ProductRule" displayName="filter for products" sourceCollectionKey
="Products"
      sourcePropertyKey="EnglishProductName" targetCollectionKey="Resellers" targetPropertyKey
="DimReseller_ProductLine"
      isActive="true"/>
   <ns3:filterRule filterRuleKey="SalesRule" displayName="filter for sales" sourceCollectionKey
="Sales"
      sourcePropertyKey="FactSales_SalesAmount" targetCollectionKey="Resellers" targetPropertyKey
="DimReseller_AnnualSales"
      isActive="true"/>
</ns3:listFilterRulesResponse>
```

Chapter 10

# EQL Record Filters

This section describes how to configure and use EQL record filters.



## About EQL record filters

EQL record filters let you define arbitrary subsets of the total record set, and dynamically restrict search and navigation results to these subsets.

The Conversation Web Service has two filtering components that allow you to use the Endeca Query Language (EQL) to provide filters for your query using EQL syntax:

- `DataSourceFilter`
- `SelectionFilter`

Both filters are used in the state of the query. The filter language for both filters is basically the record-filtering `WHERE` clause expression from EQL.

More details on the filter syntax and available EQL operators are provided in later topics.

### DataSourceFilter

The `DataSourceFilter` component filters the corpus of records before any other processing is done. In other words, this filter is applied first, and makes the universe of data that is visible to your query smaller. This means that filtered-out records will not contribute to spell correction, and will not be available as part of `AllBaseRecords` in EQL. The `DataSourceFilter` supports queries against collections (as specified in the state).

Because DataSourceFilter restricts the searchable records to a specified subset of the total records in the Dgraph, it can be used as a security filter to prevent users from obtaining records that they are not authorized to view. In EQL terms, AllBaseRecords corresponds to the records that pass the DataSourceFilter filter.

Note while processing a query, Endeca Server automatically adds a DataSourceFilter to any state that contains a collection name. For more information, see *Automatic addition of data source filter on page 108*.

### SelectionFilter

After the universe of records has been narrowed by DataSourceFilter, the SelectionFilter component is used for additional application-level filtering. It specifies the criteria for the final record result set. The results that are returned are the records that match all of the filters specified in the query.

The SelectionFilter supports queries against collections (as specified in the state). Any filter rules associated with those collections are automatically invoked in the query. If an implicit filter is generated by a request, the resulting information about that implicit filter is returned in the AppliedFilterRule element.

SelectionFilter also determines which data is available for refinement computation. NavStateRecords corresponds to the records that pass all filters (including SelectionFilter).

### Using multiple EQL filters in a query

A state can have multiple DataSourceFilter and/or SelectionFilter filters. In this case, each filter must have a unique name within the state.

If two or more filters are specified (for example, two SelectionFilter elements), the returned record set represents an intersection of the multiple searches. For example, if one SelectionFilter filter is searching for Color=red and the other SelectionFilter filter is searching for Color=blue, then each record in the resulting record set will have both "red "and "blue" assignments (rather than having a union of all "red" records and all "blue" records).

### Dgraph enablement

No Dgraph process configuration flags are necessary to enable EQL record filters.

## SelectionFilter format

The SelectionFilter type is a record filter that uses EQL query syntax.

There are two versions of SelectionFilter:

- filterString, in which the filter is expressed as an EQL string.
- filterAST, in which the filter is expressed as ExpressionBase sub-types.

When used with a collections that have filter rules, both versions will run the collection filter rule, which entails generating an implicit filter on the target collection.

### SelectionFilter filterString version

The syntax for the filterString version of the SelectionFilter is:

```
<SelectionFilter Id="?">
   <filterString>?</filterString>
   <AppliedFilterRule>
```

```
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
      <TargetPropertyKey>?</TargetPropertyKey>
   </AppliedFilterRule>
</SelectionFilter>
```

The meanings of the attributes are:

| Attribute | Meaning |
|---|---|
| `Id` | Optional. An identifier for this filter configuration. The identifier is needed only when using filter rules. The identifier must be unique among other filter identifiers in the state. |
| `filterString` | Required. Specifies a filter string using the EQL `WHERE` clause syntax. The `WHERE` expression uses one or more attributes whose values are to be tested, and one or more test conditions. For details on the syntax, see *EQL operators for filterString filters on page 125*. |
| `AppliedFilterRule` | Optional. Used for collections and filter rules. If present, indicates that the filter was implicitly generated and supplies relevant information. |
| `Source` | Optional. Information about the source of the implicit filter. If the implicit filter has been derived via a filter rule, the `Source` names the identifier of the filter and the State where it came from. |
| `FilterId` | Optional. The identifier of an implicit filter. |
| `StateName` | Optional. Specifies the name of a named state from which the implicit filter came. |
| `TargetPropertyKey` | Optional. The key that is derived from the filter rule. This is the target attribute that was used when the filter rule was applied. |

## Use of the AppliedFilterRule element

The `AppliedFilterRule` elements are used to display information about the source of an applied filter. If a filter was implicitly created via a filter rule, the `StateName` and `FilterId` elements indicate its origin

The `TargetPropertyKey` element (if present) indicates the name of the target attribute used in the generated filter. One use of this target attribute in your front-end application is in generating breadcrumbs reflecting the filter-rule-applied filter.

To illustrate these elements, assume that you have this environment:

- two collections named Sales and Resellers
- the Sales collection has an attribute named FactSales_SalesAmount, which is used for the `SelectionFilter` (which has an identifier of SalesFltr)
- a filter rule uses the Sales collection FactSales_SalesAmount attribute as the source property key and the Resellers collection DimReseller_AnnualSales attribute as the target property key

The request has two states (SalesQuery and Resell), one for each collection (note that some SOAP elements are removed for ease of reading):

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <Language>en</Language>
   <State>
      <Name>ProdRecs</Name>
      <CollectionName>Sales</CollectionName>
      <SelectionFilter Id="SalesFltr">
         <filterString>FactSales_SalesAmount > 1000</filterString>
      </SelectionFilter>
   </State>
   <State>
      <Name>ResellerRecs</Name>
      <CollectionName>Resellers</CollectionName>
   </State>
   <RecordCountConfig Id="NumRecs">
      <StateName>ProdRecs</StateName>
   </RecordCountConfig>
</Request>
```

The `Results` information might look like this:

```
<cs:Results>
   <State>
      <Name>SalesQuery</Name>
      <CollectionName>Sales</CollectionName>
      <DataSourceFilter>
         <filterString>"FactSales_ProductKey" IS NOT NULL</filterString>
      </DataSourceFilter>
      <SelectionFilter Id="SalesFlt">
         <filterString>FactSales_SalesAmount > 500</filterString>
      </SelectionFilter>
   </State>
   <State>
      <Name>Resell</Name>
      <CollectionName>Resellers</CollectionName>
      <DataSourceFilter>
         <filterString>"DimReseller_ResellerKey" IS NOT NULL</filterString>
      </DataSourceFilter>
      <SelectionFilter>
         <filterString>"DimReseller_AnnualSales" > 500</filterString>
         <AppliedFilterRule>
            <Source FilterId="SalesFlt">
               <StateName>SalesQuery</StateName>
            </Source>
            <TargetPropertyKey>DimReseller_AnnualSales</TargetPropertyKey>
         </AppliedFilterRule>
      </SelectionFilter>
   </State>
   <cs:RecordCount Id="NumRecs">
      <cs:NumRecords>180</cs:NumRecords>
   </cs:RecordCount>
</cs:Results>
```

The SalesQuery state in the results shows the following:

- A `DataSourceFilter` (using the Sales collection key FactSales_ProductKey) was implicitly applied to the Sales collection to filter out any non-Sales records for the explicit `SelectionFilter` query.
- The `SelectionFilter` (which has an identifier of SalesFlt) was explicitly applied to the FactSales_SalesAmount attribute of the Sales collection.
- This state does not have an `AppliedFilterRule` element because no filter rule was implicitly applied to it.

The Resell state in the results shows the following:

- A `DataSourceFilter` (using the Resellers collection key DimReseller_ResellerKey) was implicitly applied to the Resellers collection to filter out any non-Resellers records for the implicit `SelectionFilter` query.
- An `SelectionFilter` was implicitly applied to the DimReseller_AnnualSales attribute of the Resellers collection.
- The `Source` element shows that the implicit `SelectionFilter` came from the SalesQuery state and had an identifier of SalesFlt in that state.
- The `TargetPropertyKey` element shows that the DimReseller_AnnualSales attribute was the target property key of the filter rule.

The resulting record set will be from the state that is listed for the config (except for `EQLConfig`, which uses the records from all the states). In this example, the `RecordCountConfig` specifies the SalesQuery state, which means that the resulting 180 records are Sales records.

## SelectionFilter filterAST version

The syntax for the filterAST version of the `SelectionFilter` is:

```
<SelectionFilter Id="?">
   <filterAST>
      <typ:filter/>
   </filterAST>
   <AppliedFilterRule>
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
      <TargetPropertyKey>?</TargetPropertyKey>
   </AppliedFilterRule>
</SelectionFilter>
```

The `Id` and `AppliedFilterRule` attributes are the same as the filterString version.

An AST (Abstract Syntax Tree) is a way to represent an expression in the EQL language. Each node in an AST represents some expression. If that expression has sub-expressions on which it operates, those sub-expressions are children of the node in the AST. The AST filter itself is constructed from these `ExpressionBase` sub-types, such as the `AttributeRefExpression`, `ComparisonExpression`, and `DoubleLiteral` types used in this example:

```
<soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
      xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>
         <ns:Language>en</ns:Language>
         <ns:State>
            <ns:Name>SalesQuery</ns:Name>
            <ns:CollectionName>Sales</ns:CollectionName>
            <ns:SelectionFilter Id="SalesFlt">
              <ns:filterAST>
                   <typ:filter xsi:type="typ:ComparisonExpression" comparison=">">
                   <typ:leftOperand xsi:type="typ:AttributeRefExpression" attributeKey
="FactSales_SalesAmount"/>
                   <typ:rightOperand xsi:type="typ:DoubleLiteral" value="500"/>
                </typ:filter>
             </ns:filterAST>
            </ns:SelectionFilter>
```

```
         </ns:State>
         <ns:State>
            <ns:Name>Resell</ns:Name>
            <ns:CollectionName>Resellers</ns:CollectionName>
         </ns:State>
         <ns:RecordCountConfig Id="NumRecs">
            <ns:StateName>SalesQuery</ns:StateName>
         </ns:RecordCountConfig>
      </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that this query is basically the same as the filterString version, with the filter being specified in an AST format rather than a string.

# DataSourceFilter format

The `DataSourceFilter` type is similar in format to the `SelectionFilter` type.

The `DataSourceFilter` type is also available in filterString and filterAST versions.

Note that unlike the `SelectionFilter` type, the `DataSourceFilter` does not run filter rules associated with collections. It is for this reason that the `AppliedFilterRule` elements are not supported in a `DataSourceFilter`.

## DataSourceFilter filterString version

The syntax for the filterString version of the `DataSourceFilter` is:

```
<DataSourceFilter Id="?">
   <filterString>?</filterString>
</SelectionFilter>
```

The meanings of the attributes are:

| Attribute | Meaning |
|---|---|
| `Id` | Optional. An identifier for this filter configuration. The identifier must be unique among other filter identifiers in the state. |
| `filterString` | Required. Specifies a filter string using the EQL `WHERE` clause syntax. The `WHERE` expression uses one or more attributes whose values are to be tested, and one or more test conditions. For details on the syntax, see *EQL operators for filterString filters on page 125*. |

This example shows the use of both EQL filters:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
      xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>
         <ns:Language>en</ns:Language>
         <ns:State>
            <ns:Name>AmountQuery</ns:Name>
            <ns:DataSourceFilter Id="SourceFltr">
                <ns:filterString>COUNTRY_NAME = 'France'</ns:filterString>
```

```
            </ns:DataSourceFilter>
             <ns:SelectionFilter Id="AmntFltr">
                <ns:filterString>AMOUNT_SOLD > 1000</ns:filterString>
            </ns:SelectionFilter>
         </ns:State>
         <ns:RecordCountConfig Id="Results">
            <ns:StateName>AmountQuery</ns:StateName>
         </ns:RecordCountConfig>
      </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

In the example, the `DataSourceFilter` first filters out all records that do not have a value of "France" in their COUNTRY_NAME assignment. From the remaining records, the `SelectionFilter` then returns all records that have an AMOUNT_SOLD assignment with a value greater than 1000.

## DataSourceFilter filterAST version

The syntax for the filterAST version of the `DataSourceFilter` type is:

```
<DataSourceFilter Id="?">
   <filterAST>
      <typ:filter/>
   </filterAST>
</DataSourceFilter>
```

The meaning of the `Id` attribute is the same as the filterString version.

Like with the `SelectionFilter` type, the AST filter is constructed from `ExpressionBase` sub-types, such as `AttributeRefExpression`, `ComparisonExpression`, and `StringLiteral` types.

This example duplicates the one above, except for the use of a filterAST instead of a filterString:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
      xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:Request>
         <ns:Language>en</ns:Language>
         <ns:State>
            <ns:Name>AmountQuery</ns:Name>
            <ns:DataSourceFilter Id="SourceFltr">
              <ns:filterAST>
                   <typ:filter xsi:type="typ:ComparisonExpression" comparison="=">
                  <typ:leftOperand xsi:type="typ:AttributeRefExpression" attributeKey="COUNTRY_NAME"
/>
                   <typ:rightOperand xsi:type="typ:StringLiteral" value="France"/>
                 </typ:filter>
                  </ns:filterAST>
            </ns:DataSourceFilter>
            <ns:SelectionFilter Id="AmntFltr">
              <ns:filterAST>
                   <typ:filter xsi:type="typ:ComparisonExpression" comparison=">">
                   <typ:leftOperand xsi:type="typ:AttributeRefExpression" attributeKey="AMOUNT_SOLD"/>
                   <typ:rightOperand xsi:type="typ:DoubleLiteral" value="1000"/>
                   </typ:filter>
                  </ns:filterAST>
            </ns:SelectionFilter>
         </ns:State>
         <ns:RecordCountConfig Id="Results">
            <ns:StateName>AmountQuery</ns:StateName>
         </ns:RecordCountConfig>
      </ns:Request>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

In the example, note that the `rightOperand` of the `DataSourceFilter` is of type `StringLiteral` because COUNTRY_NAME is a string attribute, while the `rightOperand` of the `SelectionFilter` is of type `DoubleLiteral` because AMOUNT_SOLD is an attribute of type double.

# EQL operators for filterString filters

The filterString version of EQL record filters are specified with `WHERE` clause types of Boolean expressions.

The `WHERE` expression uses one or more attributes whose values are to be tested, and one or more test conditions. For example, the expression can use numeric and string value comparison operators, NULL value evaluation operators, and logical operators, as well as some functions. Note that unlike an EQL statement, the `WHERE` keyword itself is not used in the query string.

The following table lists the operators that can be used in an EQL record filter expression:

| Operator | Description | Example |
|---|---|---|
| = | Equal (tests the equality between two expressions) | COUNTRY_NAME = 'France' |
| <> | Not equal (tests the condition of two expressions not being equal to each other) | PROD_WEIGHT_CLASS <> 2 |
| > | Greater than (tests the condition of one expression being greater than the other) | PROD_MIN_PRICE > 1000 |
| < | Less than (tests the condition of one expression being less than the other) | QUANTITY_SOLD < 500 |
| >= | Greater than or equal (tests the condition of one expression being greater than or equal to the other expression) | PROD_MIN_PRICE >= 75 |
| <= | Less than or equal (tests the condition of one expression being less than or equal to the other expression) | PROMO_COST <= 1500 |
| BETWEEN low AND high | Specifies an inclusive range of values. Use AND to separate the low (starting) and high (ending) values. | FISCAL_YEAR BETWEEN 2000 AND 2006 |
| IS NULL | Specifies a search for NULL values in a single-assign attribute. | CUST_EMAIL IS NULL |
| IS NOT NULL | Specifies a search for values that are not NULL in a single-assign attribute. | PROD_STATUS IS NOT NULL |

| Operator | Description | Example |
|---|---|---|
| IS EMPTY | Specifies a search for an empty set (i.e., from a multi-assign attribute). | LOCALES IS EMPTY |
| IS NOT EMPTY | Specifies a search for a non-empty set (i.e., from a multi-assign attribute). | LOCALES IS NOT EMPTY |
| AND | Combines two conditions and evaluates to TRUE when both of the conditions are TRUE. | PROD_MIN_PRICE > 1000 AND COUNTRY_NAME = 'Spain' |
| OR | Combines two conditions and evaluates to TRUE when either condition is TRUE. | PROD_LIST_PRICE > 50 OR PROD_CATEGORY = 'Hardware' |
| NOT | Reverses the value of any Boolean expression. | NOT(COUNTRY_REGION = 'Europe' AND AMOUNT_SOLD > 1000) |

Note that you cannot use aggregating functions (such as `SUM`) in the query.

## Syntax for single-assign versus multi-assign attributes

The EQL syntax for the `WHERE` expression will depend on whether an attribute is configured as a single-assign or multi-assign attribute. For example, if the Flavors attribute is a single-assign string attribute, then this comparison syntax would work:

```
<filterString>Flavors = 'Peach'</filterString>
```

But if Flavors is a multi-assign attribute, then that syntax would fail with this error message:

```
Cannot compare mdex:string-set and mdex:string
```

The reason for the error is that in EQL a multi-assign attribute is treated as a set (of data type `mdex:string-set`), and a string (such as 'peach') cannot be compared to a string-set. (In other words, in EQL a single-assign attribute is of data type `mdex:string` while a multi-assign attribute is considered as being of data type `mdex:string-set`.) Thus, the expression would have to use the multi-assign syntax, such as these three examples:

```
<filterString>SOME i IN Flavors SATISFIES (i = 'Peach')</filterString>

<filterString>IS_MEMBER_OF('Peach', Flavors)</filterString>

<filterString>'Peach' IN Flavors</filterString>
```

This caveat for single-assign versus multi-assign attributes applies to all data types. For more information on working with multi-assign data in EQL, see the *Oracle Endeca Server EQL Guide*.

## Using single quotes with string values

When using string value comparison operators, make sure that you use single quotes around the text value field. For example, if the COUNTRY_NAME standard attribute is of type `mdex:string`, then the usage would be:

```
COUNTRY_NAME = 'Spain'  // Correct
```

```
COUNTRY_NAME = "Spain"  // Incorrect because double quotes are not allowed

COUNTRY_NAME = Spain  // Incorrect because the attribute stores string values
```

Also keep in mind that string comparisons are case-sensitive. Thus:

```
COUNTRY_NAME = 'spain'
```

would not match if all COUNTRY_NAME values were "Spain" (i.e., no "spain" values).

When using numeric value comparison operators, do not use quotes of any kind around the value field. For example, if the AMOUNT_SOLD standard attribute is of type `mdex:double`, then the usage would be:

```
AMOUNT_SOLD = 500  // Correct

AMOUNT_SOLD = "500"  // Incorrect because the attribute stores numeric values
```

### Escaping special XML characters

If you are making direct queries against the Conversation Web Service (for example, by using the soapUI tool), you may need to escape some XML characters to prevent parsing errors. For example, you should use the **<** escape character instead of the **<** (less than) character. Note that examples in this section will use the unescaped version for ease of reading.

## Language codes for EQL error messages

You can set the language to be used for EQL parsing error messages.

The `Request` complex type has an optional `Language` element that sets the language for error messages that result from EQL parsing. The supported languages and their corresponding language codes are:

- Chinese (simplified): `zh_CN`
- Chinese (traditional): `zh_TW`
- English: `en`
- French: `fr`
- German: `de`
- Italian: `it`
- Japanese: `ja`
- Korean : `ko`
- Portuguese: `pt`
- Spanish: `es`

If a language code is not specified, then `en` (English) is used as the default.

Note that this `Language` element serves a different purpose from the `Language` attribute in the `TextSearchFilter` type (for record search) and the `ValueSearchConfig` type (for value search).

### Example of setting a language code

The following example shows where in the request you would specify the `Language` element for EQL parsing error messages:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <Language>fr</Language>
   <State>
      <SelectionFilter Id="AmtFilter">
         <filterString>AMOUNT_SOLD > 1000</filterString>
      </SelectionFilte>
   </State>
   ...
</Request>
```

In this example, `fr` (French) is set as the language in which all EQL parsing error messages are returned.

# Range filters

EQL filters allow a user, at request time, to specify an arbitrary, dynamic range of values that are then used to limit the records returned for a navigation query.

The remaining refinement values for the records in the result set are also returned. For example, a range filter would be used if a user were querying for bicycle gloves within a specific price range, for example between $10 and $40.

It is important to remember that range filters are simply modifiers for a navigation query. Only records returned by the basic navigation request are considered when evaluating the range filter. You can use a range filter in a query on any of the record attributes.

*Between range filters*

*Less-than and greater-than range filters*

## Between range filters

Use the EQL `BETWEEN` operator to construct between range filter queries.

A between range filter query returns records with a standard or managed attribute value that falls between a lower bound and an upper bound.

### BETWEEN operator syntax

The syntax for `BETWEEN` is:

```
attribute BETWEEN lowerBound AND upperBound
```

where *`attribute`* is the attribute whose value will be tested.

`BETWEEN` is inclusive, which means that it returns TRUE if the value of *`attribute`* is greater than or equal to the value of *`lowerBound`* and less than or equal to the value of *`upperBound`*.

Supported data types for *`attribute`* and the range values are integer, double, dateTime, duration, time, string, and Boolean. With one exception, *`attribute`* must be of the same data type as *`lowerBound`* and *`upperBound`*. The exception is that you can use a mix of integer and double, because the integer is promoted to a double.

## Between single-assign examples

This first example shows a between range filter query for a single-assign attribute (UNIT_PRICE) of type double:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <Language>es</Language>
        <State>
          <DataSourceFilter Id="DataFlt">
             <filterString>COUNTRY_NAME = 'Spain'</filterString>
          </DataSourceFilter>
          <SelectionFilter Id="BtwFlt">
             <filterString>UNIT_PRICE BETWEEN 500 AND 1000</filterString>
          </SelectionFilter>
        </State>
        <RecordListConfig Id="Recs" MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
</soapenv:Envelope>
```

This example first limits the data source records to those that have a COUNTRY_NAME assignment of 'Spain' and then returns all records whose UNIT_PRICE value is between 500 and 1000. Because both bound elements are inclusive, the returned records include those with UNIT_PRICE values of 500 and 1000.

This second example shows a between range filter query for a single-assign attribute (COUNTRY_NAME) of type string:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <State>
          <DataSourceFilter Id="DataFlt">
             <filterString>AMOUNT_SOLD > 100</filterString>
          </DataSourceFilter>
          <SelectionFilter Id="BtwFlt">
             <filterString>COUNTRY_NAME BETWEEN 'Argentina' AND 'Japan'</filterString>
          </SelectionFilter>
        </State>
        <RecordListConfig Id="Recs" MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
</soapenv:Envelope>
```

This example first limits the data source records to those that have an AMOUNT_SOLD assignment of 100 or greater and then returns all records whose COUNTRY_NAME assignment value is a country name between Argentina and Japan (such as Canada and Italy).

## Between multi-assign example

This example shows a between range filter query for a single-assign attribute (Quantity) of type integer:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <Language>en</Language>
         <State>
```

```
            <SelectionFilter>
               <filterString>SOME i IN Quantity SATISFIES (i BETWEEN 10 AND 30)</filterString>
            </SelectionFilter>
         </State>
         <RecordListConfig Id="recs">
            <Column>Quantity</Column>
         </RecordListConfig>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The example would return all records whose Quantity assignment is between 10 and 30.

## Less-than and greater-than range filters

Two EQL operators allow you to make less-than and greater-than range filter queries.

You make these types of queries as follows:

- To make a **less-than** query, use only the < (less-than) operator. Because you are specifying only the upper bound of the range, all returned records will fall below this bound (i.e., be less than the upper bound).
- To make a **greater-than** query, use only the > (greater-than) operator. Because you are specifying only the lower bound of the range, all returned records will be above this bound (i.e., be greater than the lower bound).

Note that both operators are inclusive, so that records that are equal to the specified boundary value will be returned.

### Greater-than example

The following is an example of a greater-than query with two single-assign attributes:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <Language>de</Language>
        <State>
          <DataSourceFilter Id="DataFlt">
             <filterString>FISCAL_YEAR = 2002</filterString>
          </DataSourceFilter>
          <SelectionFilter Id="GtFlt">
             <filterString>AMOUNT_SOLD > 500</filterString>
          </SelectionFilter>
        </State>
        <RecordListConfig Id="RecList MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
</soapenv:Envelope>
```

This example first filters out all records except those from fiscal year 2002, and then returns all records that have sold 500 or more items.

An example of a less-than query would be the same except for the use of the < (less-than) operator.

For the multi-assign attribute (named AnnualRevenue in this example), the filter string would look like this:

```
<SelectionFilter Id="SalesFlt">
   <filterString>SOME i IN AnnualRevenue SATISFIES (i > 150000)</filterString>
```

```
</SelectionFilter>
```

This example returns every record in which an AnnualRevenue assignment is greater than 150000.

# Geocode filters

When used with a standard attribute of type geocode, the EQL `DISTANCE` function indicates a filter based on the distance of that geocode attribute from a given reference point.

The `DISTANCE` function returns the distance (in kilometers) between two geocodes. The syntax for `DISTANCE` is:

```
DISTANCE(geoAttribute, TO_GEOCODE(latitude,longtitude))
```

where *`geoAttribute`* is a standard attribute of type geocode.

The `TO_GEOCODE` function creates a geocode from a given latitude and longitude pair, both of which must be of type double:

- The latitude of the location is specified in whole and fractional degrees (positive values indicate north latitude, and negative values indicate south latitude).
- The longitude of the location in whole and fractional degrees (positive values indicate east longitude, and negative values indicate west longitude).

The distance limits in geocode filters are always expressed in kilometers. The records are filtered by the distance from the geocode reference point to the latitude/longitude pair.

Note that both `DISTANCE` and `TO_GEOCODE` operate only on single-assign geocode attributes.

## Between geocode filters

Use the `BETWEEN` operator to indicate that the distance from the geocode attribute to the reference point is between two bounds:

- The lower bound specifies a greater-than distance (in kilometers) from the geocode attribute to the reference point.
- The upper bound specifies a less-than distance (in kilometers) from the geocode attribute to the reference point.

The following example uses the `BETWEEN` operator:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <Language>en</Language>
        <State>
          <DataSourceFilterString>
            COUNTRY_NAME = 'United States of America'
          </DataSourceFilterString>
          <SelectionFilterString>
            DISTANCE(Location, TO_GEOCODE(40.758224, -73.917404)) BETWEEN 1 AND 500
          </SelectionFilterString>
        </State>
        <RecordListConfig Id="RecordList MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
```

```
</soapenv:Envelope>
```

The query returns only records whose location (in the Location property) is between 1 and 500 kilometers from the reference point.

### Less-than and greater-than geocode filters

You can make queries that return records that are less-than or greater-than a specific number of kilometers from the reference point:

- To make a **less-than** geocode query, use only the **<** (less-than) operator. Because you are specifying only the upper bound of the distance from the reference point, all returned records will fall below this bound.
- To make a **greater-than** geocode query, use only the **>** (greater-than) operator. Because you are specifying only the lower bound of the range, all returned records will be above this bound.

The following is an example of a greater-than geocode query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <State>
          <DataSourceFilterString>
            COUNTRY_NAME = 'United States of America'
          </DataSourceFilterString>
          <SelectionFilterString>
            DISTANCE(Location, TO_GEOCODE(40.758224, -73.917404)) > 200
          </SelectionFilterString>
        </State>
        <RecordListConfig Id="RecordList MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
</soapenv:Envelope>
```

The query returns only records whose location (in the Location property) is equal to or greater than 200 kilometers from the reference point.

An example of a less-than query would be the same except for the use of the **<** (less-than) operator.

# Managed attribute hierarchy filters

An EQL record filter can specify managed attribute values for the search criteria.

You can use two EQL hierarchy functions to specify managed attribute values:

| **Hierarchy function** | |
|---|---|
| `IS_ANCESTOR(`*`managedAttribute`*`, `*`valueSpec`*`)` | Include the record if the named attribute is the attribute specified or an ancestor. If the attribute is not a member of the specified hierarchy, it is a query-time error. |

| Hierarchy function | |
|---|---|
| IS_DESCENDANT(*managedAttribute*, *valueSpec*) | Include the record if the named attribute is the attribute specified or a descendant, and if the specified value spec matches. If the attribute is not a member of the specified hierarchy, it is a query-time error. |

For both functions, *managedAttribute* is the name of a managed attribute, and *valueSpec* (specified as a string) is the spec (not the value name) of the managed attribute value.

### Example

This example uses the IS_DESCENDANT function:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <State>
          <DataSourceFilter Id="DataFlt">
             <filterString>COUNTRY_NAME = 'United States of America'</filterString>
          </DataSourceFilter>
          <SelectionFilter Id="MavFlt">
             <filterString>IS_DESCENDANT(ProductCategory, '140')</filterString>
          </SelectionFilter>
        </State>
        <RecordListConfig Id="Recs" MaxPages="20">
           <RecordsPerPage>5</RecordsPerPage>
        </RecordListConfig>
      </Request>
    </soapenv:Body>
</soapenv:Envelope>
```

The search is filtered on the managed attribute value that has a spec of "140" and is a descendant of the ProductCategory managed attribute. Each returned record should have the following assignment:

```
<cs:Record>
   ...
   <cs:attribute name="ProductCategory" type="mdex:string" displayName="Endurance Racing">140<
/cs:attribute>
   ...
</cs:Record>
```

# Boolean attribute filters

Filtering by Boolean attribute assignments is supported.

You can specify the Boolean value as true or false (in either upper- or lower-case).

For example, this query filters on assignments of false in the Boolean single-assign attribute named AFFINITY_CARD:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
        <Language>fr</Language>
        <State>
          <DataSourceFilter>
```

```
            <filterString>COUNTRY_NAME = 'France'</filterString>
         </DataSourceFilter>
         <SelectionFilter Id="BoolFlt">
            <filterString>AFFINITY_CARD = false</filterString>
         </SelectionFilter>
       </State>
       <RecordListConfig Id="Recs" MaxPages="20">
          <RecordsPerPage>5</RecordsPerPage>
       </RecordListConfig>
     </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Each returned record should have the following assignment:

```
<cs:Record>
   ...
   <cs:attribute name="AFFINITY_CARD" type="mdex:boolean">false</cs:attribute>
   ...
</cs:Record>
```

# EQL filters with record and value searches

Either or both of the EQL record filters can be used with a record search or a value search.

The `DataSourceFilterString` component is especially useful (for example, as a security filter) to restrict the searchable records for these types of searches.

This example uses the `DataSourceFilter` EQL filter with a record search filter:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <Language>en</Language>
         <State>
            <Name>JulyRpts</Name>
            <DataSourceFilter Id="MonthFltr">
                <filterString>FISCAL_MONTH_NAME = 'July'</filterString>
            </DataSourceFilter>
            <TextSearchFilter Key="PROD_CATEGORY" RelevanceRankingStrategy="numFields"
              Mode="AllPartial" EnableSnippeting="false" Language="en">
              electronics
            </TextSearchFilter>
         </State>
         <RecordListConfig Id="Recs" MaxPages="30">
            <StateName>JulyRpts</StateName>
            <Column>PROD_CATEGORY</Column>
            <RecordsPerPage>5</RecordsPerPage>
         </RecordListConfig>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The `DataSourceFilter` filter first limits all the searchable records to those from the fiscal month of July. Then the `TextSearchFilter` uses the PROD_CATEGORY attribute for its record search.

# EQLConfig requests

The `EQLConfig` complex type allows you to send arbitrary EQL statements for evaluation.

Consider the following EQL statement:

```
RETURN SalesTransactions AS SELECT SUM(FactSales_SalesAmount)
WHERE (DimDate_FiscalYear=2008) AS Sales2008,
SUM(FactSales_SalesAmount)
WHERE (DimDate_FiscalYear=2007) AS Sales2007,
((Sales2008-Sales2007)/Sales2007 * 100) AS pctChange,
COUNTDISTINCT(FactSales_SalesOrderNumber)
AS TransactionCount
GROUP
```

To send it for processing to the Oracle Endeca Server, use the `EQLConfig` type, including the statement inside the `EQLQueryString` element, as in this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
  xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
<soapenv:Header/>
<soapenv:Body>
 <ns:Request>
  <ns:Language>en</ns:Language>
  <ns:State/>
  <ns:EQLConfig Id="EQLRequest">
     <ns:EQLQueryString>
      RETURN SalesTransactions AS SELECT SUM(FactSales_SalesAmount)
      WHERE (DimDate_FiscalYear=2008) AS Sales2008,
      SUM(FactSales_SalesAmount) WHERE (DimDate_FiscalYear=2007) AS Sales2007,
      ((Sales2008-Sales2007)/Sales2007 * 100) AS pctChange,
      countDistinct(FactSales_SalesOrderNumber)
      AS TransactionCount
      GROUP
     </ns:EQLQueryString>
  </ns:EQLConfig>
 </ns:Request>
 </soapenv:Body>
</soapenv:Envelope>
```

The contents of the `EQLQueryString` element must be a valid EQL statement.

The following abbreviated response returned from the Conversation Web Service contains the calculated results of the EQL statements:

```
<cs:EQL Id="EQLRequest">
 <cs:ResultRecords NumRecords="1" Name="SalesTransactions">
  <cs:DimensionHierarchy/>
   <cs:AttributeMetadata name="Sales2007" type="mdex:double"/>
   <cs:AttributeMetadata name="Sales2008" type="mdex:double"/>
   <cs:AttributeMetadata name="TransactionCount" type="mdex:long"/>
   <cs:AttributeMetadata name="pctChange" type="mdex:double"/>
      <cs:Record>
        <Sales2007 type="mdex:double">2.79216705182E7</Sales2007>
        <Sales2008 type="mdex:double">3.62404846965997E7</Sales2008>
        <TransactionCount type="mdex:long">3796</TransactionCount>
        <pctChange type="mdex:double">29.793397114178</pctChange>
     </cs:Record>
 </cs:ResultRecords>
</cs:EQL>
```

Chapter 11

# Working with Records

This section describes how to filter data and non-data records, display records, and their details and attribute values, configure a record list, and retrieve a large number of records.



## Filtering data and non-data records

The `RecordKind` filter is used to restrict records to a particular kind.

You use the `RecordKind` filter if you wish to perform some queries only against data records, or other queries only against PDRs or against all system records.

The `RecordKind` filter allows the following values:

| Value | Description |
|---|---|
| `data` | Restricts records to actual data records that you load to the data domain. |
| `properties` | Restricts records to those records that declare record attributes (PDRs). Note that default list includes the PDRs for the primordial records. |

| Value | Description |
|---|---|
| `nondata` | Restricts records to those records that do not represent data records, and instead represent these record types:<br>• GCR (Global Configuration Record)<br>• PDR (standard attribute, both primordial and user-created description records)<br>• DDR (managed attribute description records)<br>• MAVDR (managed attribute value description records)<br>• CDR (collection description records)<br>• FRDR (filter rule description records)<br>• PRDR (precedence rule description records)<br>• EDR (entity description records)<br>• GDR (attribute group configuration description records)<br>• GMDR (attribute group membership description records) |

When using the `RecordKind` filter, you can restrict the `State` from which all other search and filtering operations in Endeca Server will be performed. It is useful to use the `RecordKind` filter in cases when you want to issue subsequent queries only on a subset of records in the corpus, for example, only on those records that represent actual data records.

The following soapUI example illustrates how to use the `RecordKind` filter to return all CDRs:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <Language>en</Language>
         <State>
            <Name>RecKind</Name>
            <RecordKind>nondata</RecordKind>
            <SelectionFilter>
               <filterString>"mdex-collection_Key" is not null</filterString>
            </SelectionFilter>
         </State>
          <RecordListConfig Id="RecList">
            <StateName>RecKind</StateName>
            <RecordsPerPage>5</RecordsPerPage>
         </RecordListConfig>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that `RecordKind` results are not visible in the user interface — they do not appear in breadcrumbs, for example.

# Displaying records and attribute values with Studio

The **Results Table** component displays records in a tabular format. The **Results List** component displays records in a format similar to regular Web search results. The **Data Explorer** component displays each record as a set of key-value pairs.

For details on adding and configuring a **Results Table**, **Results List**, or **Data Explorer** component in your Studio application, see the *Oracle Endeca Information Discovery Studio User's Guide*.

# Displaying records and attribute values with the API

This section describes how to use the Conversation Web Service to request records and their attribute values from the data domain.



## Configuring a record list

You use the `RecordListConfig` complex type to configure settings for lists of records returned from a query.

For example, with `RecordListConfig`, you can configure how many records should be included per page in the list, how many pages of records should be returned, which attributes should be returned for each record included in the list, and the attribute by which to sort the record list. As a result of a query containing `RecordListConfig`, a `RecordList` is returned.

In the `RecordListConfig` complex type, you define what information should be returned in the record list. The format of the `RecordListConfig` type is:

```
<RecordListConfig Id="?" MaxPages="20">
   <StateName>?</StateName>
   <Column>?</Column>
   <RecordsPerPage>?</RecordsPerPage>
   <Page>?</Page>
   <Sort Key="?" Direction="?">
      <GeocodeReferencePoint latitude="?" longitude="?"/>
   </Sort>
</RecordListConfig>
```

The elements and attributes that are specified in the `RecordListConfig` complex type are the following:

| Element/Attribute | Description |
|---|---|
| `Id` | Required. An arbitrary identifier for this `RecordListConfig`. |
| `MaxPages` | Optional. Specifies an integer that is the maximum number of record pages to be returned. If this attribute is omitted, a default value of 20 is used for the query. |
| `StateName` | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the `StateName` element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the `StateName` element is used to reference that named state (as the state will be used in any event in the `RecordListConfig`).<br>• If the request has an unnamed state, then the `StateName` element cannot be used. |
| `Column` | Optional. Specifies an attribute that will be returned in the `RecordList` with the record. You can specify multiple instances of the `Column` element. Note that you do not have to specify the primary key, because it is automatically returned. If no `Column` elements are specified, then all the record's standard and managed attributes are returned. |
| `RecordsPerPage` | Optional. Specifies an integer that is the maximum number of records (`Record` elements) to be displayed in the `ContentElement` of the result. If this element is omitted, a default value of 10 is used. |
| `Page` | Optional. Specifies an integer that is the page to be displayed (that is, it provides an offset into the overall list of pages). The offset is a zero-based index, which means that 0 (zero) specifies the first page. This element allows users to page through a long result set, either directly or step by step. If an offset is greater than the total number of pages, then the record list returned will not include records. If this element is omitted, a default value of 0 is used. |
| `Sort` | Optional. Specifies a sort order for the record list. `Key` specifies the standard or managed attribute used for the sort. `Direction` is optional and specifies an `Ascending` (the default) or `Descending` sort order. `GeocodeReferencePoint` is used with geocode attributes to specify the geocode reference point. |

## Example of a RecordListConfig

This soapUI example searches for records with the Flavors attribute having an assignment of "peaches":

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
   xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types">
   <soapenv:Header/>
```

```
   <soapenv:Body>
      <ns:Request>
         <ns:Language>en</ns:Language>
         <ns:State>
            <ns:Name>MyRecSearch</ns:Name>
            <ns:TextSearchFilter Key="Flavors" Mode="AllPartial" EnableSnippeting="false"
                Language="en">peach</ns:TextSearchFilter>
         </ns:State>
         <ns:RecordListConfig Id="RecordList" MaxPages="20">
            <ns:StateName>MyRecSearch</ns:StateName>
            <ns:Column>Flavors</ns:Column>
            <ns:RecordsPerPage>5</ns:RecordsPerPage>
            <ns:Page>1</ns:Page>
            <ns:Sort Key="Flavors" Direction="Ascending"/>
         </ns:RecordListConfig>
      </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The `RecordListConfig` type configures how the returned record list should look.

## Understanding a RecordList result

The records returned from the query are contained in the `RecordList` element.

A list of records is returned with every query result received from the Oracle Endeca Server. The list of records is represented as a `RecordList` complex type that is returned in a `Results` response by the Conversation Web Service.

The `RecordList` type will typically have multiple `RecordListEntry` types. Each record is returned in a `Record` element within a `RecordListEntry`.

The following sample snippet shows a `RecordList` with one record, one pagination control, and one column:

```
<cs:Results
    xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <cs:State>
      ...
   </cs:State>
   <cs:RecordList Id="RecordList">
      <cs:NumRecords>2231</cs:NumRecords>
      <cs:TotalPages>224</cs:TotalPages>
      <cs:RecordRange First="1" Last="10"/>
      <cs:RecordListEntry>
        <cs:Record>
            <cs:attribute name="AFFINITY_CARD" type="mdex:boolean">false</cs:attribute>
            <cs:attribute name="AMOUNT_SOLD" type="mdex:double">550.370000</cs:attribute>
             ...
            <cs:attribute name="WEEK_ENDING_DAY" type="mdex:dateTime">2000-03-26T00:00:00.000Z
            </cs:attribute>
         </cs:Record>
         <cs:ComputedProperties/>
      </cs:RecordListEntry>
      ...
      <cs:DimensionHierarchy>
         <cs:DimensionValueWithPath>
            <cs:DimensionValue DimensionName="Channel" Spec="3">Direct Sales</cs:DimensionValue>
            <cs:DimensionValue DimensionName="Channel" Spec="Direct">Direct</cs:DimensionValue>
            <cs:DimensionValue DimensionName="Channel" Spec="/">Channel</cs:DimensionValue>
         </cs:DimensionValueWithPath>
      </cs:DimensionHierarchy>
      <cs:Column ColumnKey="AMOUNT_SOLD" DisplayName="Amount Sold" SpecColumn="false"/>
      ...
   </cs:RecordList>
```

```
</cs:Results>
```

The elements in the `RecordList` contain the following information:

- `NumRecords` specifies the total number of records (`Record` elements) that were returned from the query.
- `TotalPages` lists the total number of pages of records.
- `RecordRange` lists the starting and ending records for this page set.
- `DimensionHierarchy` lists paths of managed attributes whose values have assignments in the requested record list. Also contains `DimensionValueWithPath`.
- Each `RecordListEntry` contains a specific record in a `Record` element and a `ComputedProperties` element that has any computed attributes (such as geocode distance or snippets) for that record.

In addition, the attributes on the `Column` element contain the following information for a specific standard or managed attribute on a record:

- `ColumnKey` identifies the name (in an NCName format) of the attribute.
- `DisplayName` specifies the name of the attribute in an easy-to-understand format. (Once you define display names, they appear in the front-end application.)
- `SpecColumn` identifies whether the attribute is the primary key for the records. If set to `true`, identifies this property as the primary key attribute for the records. The `SpecColumn` allows you to select a record for viewing its record details.

A typical implementation will display a summarized list of matching records for the user's current navigation state. In the application front end, the record list is often displayed as a table, with each row corresponding to a specific record. Each row displays some identifying information about that specific record, such as a name, title, or identification number. Your application front end can iterate through this record list, extract the identifying information for each record, and display a table containing the results.

## Paging through a large record set

If many records are returned, you can use the `Page` attribute to specify the page to be displayed.

A query to the Oracle Endeca Server may return more records than can be displayed all at once. A common user interface mechanism for overcoming this is to create pages of results, where each page displays a subset of the entire result set.

The `RecordList` in the `Results` response includes these page-relevant fields:

- `NumRecords` specifies the total number of records that were returned from the query.
- `TotalPages` lists the total number of pages of records.
- `RecordRange` lists the starting and ending records for this page set.

The following is an abbreviated example of these fields - a `RecordList` with a total of seven record pages and three records per page:

```
<RecordList Id="RecordList">
   <NumRecords>2231</NumRecords>
   <TotalPages>224</TotalPages>
   <RecordRange First="1" Last="10"/>
   ...
</cs:RecordList>
```

The RecordList is the initial access point for providing the paging controls for the entire record set. By default, the query returns a maximum of ten records for display. To override this setting, use the RecordsPerPage
element in the RecordListConfig type, as in this example that sets five records per page for display:

```
<RecordListConfig Id="RecordList" MaxPages="20">
   <RecordsPerPage>5</RecordsPerPage>
   ...
</RecordListConfig>
```

The default page offset for a record set is zero, meaning that the first ten records are displayed. The default offset can be overridden with the Page attribute, as in this example that sets the offset to the third page of records:

```
<RecordListConfig Id="RecordList" MaxPages="20">
   ...
   <Page>3</Page>
</RecordListConfig>
```

If the number of total pages is 1:

```
<TotalPages>1</TotalPages>
```

then no paging controls are needed.

## Retrieving large numbers of records

To obtain a large number of records that can later be exported, you request them as part of the RecordListConfig type in the Conversation Web Service.

A query that requests a large number of records that could later be exported is the same as any valid navigation query requesting a list of records. This topic contains examples of Conversation Web Service request and response formats for such a query. No configuration is necessary to request a large number of records. Any record that is returned as part of the RecordListConfig request is available to be exported.

When creating the navigation query for a list of records that will be exported, you do not need to specify the number of records that should be returned. The Conversation Web Service returns records in the record list as it would for any other request for records. If you are using Studio, the setting that limits the number of records for export is configured in Studio. For information on configuring this setting, see the *Oracle Endeca Information Discovery Studio Administration and Customization Guide*.

### Example request

To request a record list with a Conversation Web Service request, use the RecordListConfig type. There is no requirement to specify any new parameters in the RecordListConfig. Simply set the RecordsPerPage to the number of records desired for export, and Page to 0.

In this abbreviated example, you can see the format for RecordListConfig:

```
<RecordListConfig Id="RecordList" MaxPages="40">
   <Column>WineType</Column>
   <Column>Price</Column>
      <RecordsPerPage>20</RecordsPerPage>
   <Page>0</Page>
   <Sort Key="Num" Direction="Ascending" />
</RecordListConfig>
```

### Example response

The following abbreviated example shows a returned list:

```
<cs:RecordList Id="RecordList">
   <cs:NumRecords>19</cs:NumRecords>
   <cs:TotalPages>40</cs:TotalPages>
   <cs:RecordRange First="1" Last="19"/>
   <cs:RecordListEntry>
      <cs:Record>
         ...
      </cs:Record>
      <cs:ComputedProperties/>
   </cs:RecordListEntry>
      ...
</cs:RecordList>
```

## Exporting large numbers of records

Endeca Server does not have a dedicated facility for the bulk export of records. Instead, you can pin a version of the index, and request a large number of records from this version, with paging through the results. The records are returned in an XML format and thus require parsing as a post-processing step.

The following procedure implies that you use the Endeca Server only, but do not use Studio. This procedure relies on direct methods within the Endeca Server. If you have installed Studio, you can use its facility to export large numbers of records.

To export records in bulk from the Endeca Server with the Conversation Web Service:

1. Pin a data version, using a request with `PinDataVersion` element, specifying an *`optional_pin_timeout`*. For details, see *Holding on to a data version on page 72*.

   For example:

   ```
   ...
   <ns:Request>
    <!--Optional:-->
     <ns:PinDataVersion>optional_pin_timeout</ns:PinDataVersion>
   <ns:State>
     ....
   </ns:State>
   ```

   The request should return the version number in its header: `X-Endeca-Served-Data-Version`. This is the version number that you pinned.

2. Issue a subsequent request that references this version number in the `DataVersionRequested` element. This request should also include a `RecordListConfig` type. For details, see *Retrieving large numbers of records on page 142*.

   If the request is successful, it returns the requested records.

It is up to you, as the front-end application developer, to determine what to do with the retrieved records. For example, you can display each record's attribute values. You can also write code to properly format the returned attribute values for export to an external file, such as a Microsoft Excel spreadsheet, or a CSV file. To help with performance of parsing received records, use a SAX parser. See, for example, *Parsing an XML file using SAX*.

## Displaying attribute values

You can send a request asking to display attribute values assigned on records.

Records are returned in `Record` elements. Each attribute value on a record is returned in a format like this example:

```
<attribute name="AMOUNT_SOLD" type="mdex:double">550.370000</attribute>
```

The `name` field lists the name of the standard or managed attribute, while the `type` field shows the data type of that attribute. Managed attributes also have the `displayName` that is used for UI purposes.

This example shows a record with four standard attribute values and one managed attribute value:

```
<cs:Record>
   <cs:attribute name="AFFINITY_CARD" type="mdex:boolean">false</cs:attribute>
   <cs:attribute name="AMOUNT_SOLD" type="mdex:double">550.370000</cs:attribute>
   <cs:attribute name="CALENDAR_MONTH_DESC" type="mdex:string">2000-03</cs:attribute>
   <cs:attribute name="Channel" type="mdex:string" displayName="Direct Sales">3</cs:attribute>
   <cs:attribute name="PROMO_FK" type="mdex:long">999</cs:attribute>
</cs:Record>
```

Your front-end application can iterate through the record, extract the attribute values for the record, and display a table containing the results.

## Displaying record details

The `RecordDetailsConfig` type defines the configuration for a record details query.

A record details query is a query for a single, specific record. Configuration information for this type of query is provided in a `RecordDetailsConfig` component. This component lets you configure aspects of the returned record, such as its primary key (record specifier) and which standard attributes and/or managed attributes should be returned.

### RecordDetailsConfig syntax

The format of the `RecordDetailsConfig` type is:

```
<RecordDetailsConfig Id="?">
   <StateName>?</StateName>
   <RecordSelector Name="?" Spec="?"/>
   <Column>?</Column>
</RecordDetailsConfig>
```

The meanings of the `RecordDetailsConfig` elements and attributes are as follows:

| Element/Attribute | Meaning |
|---|---|
| `Id` | Required. An arbitrary identifier for this `RecordDetailsConfig`. |
| `RecordSelector` `Name` `Spec` | Required. A `RecordSelector` selects a record based on a Key/Value pair that is unique to that record (typically, this is the record specifier). `Name` specifies the key (name) of the unique attribute while `Spec` is the value of the key. |

| Element/Attribute | Meaning |
|---|---|
| `Column` | Optional. Specifies a standard or managed attribute that should be returned with the record. You can specify multiple instances of the `Column` element. Note that you do not have to specify the primary key, because it is automatically returned. If no `Column` elements are specified, then all the record's assignments are returned. |
| `StateName` | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the `StateName` element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the `StateName` element is used to reference that named state (as the state will be used in any event in the `RecordDetailsConfig`).<br>• If the request has an unnamed state, then the `StateName` element cannot be used. |

## RecordDetailsConfig example

The following request is for details on the record with a primary key of 34750:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State/>
   <RecordDetailsConfig Id="RecDetails">
      <RecordSelector Name="WineID" Spec="34750"/>
      <Column>WineType</Column>
      <Column>Description</Column>
   </RecordDetailsConfig>
 </Request>
```

Two record assignment attributes (WineType and Description) will be returned in the results, as well as the primary key (WineID) of the record.

## RecordDetails result

The record details returned from the record query are contained in the `RecordDetails` element. The `RecordDetails` element is wrapped in a `Results` component, together with the original `State`.

The following example details of the requested record is returned in a `RecordDetails` component by the Conversation Web Service:

```
<cs:RecordDetails Id="RecDetails">
   <cs:Record>
      <cs:attribute name="Description" type="mdex:string">Dense and vegetal, with peach flavors.<
/cs:attribute>
      <cs:attribute name="WineID" type="mdex:int">34750</cs:attribute>
      <cs:attribute name="WineType" type="mdex:string" displayName="Red">Red</cs:attribute>
   </cs:Record>
   <cs:DimensionHierarchy>
      <cs:DimensionValueWithPath>
         <cs:DimensionValue DimensionName="WineType" Spec="Red">Red</cs:DimensionValue>
         <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
      </cs:DimensionValueWithPath>
   </cs:DimensionHierarchy>
   <cs:Column ColumnKey="Description" DisplayName="Description" SpecColumn="false"/>
   <cs:Column ColumnKey="WineID" DisplayName="WineID" SpecColumn="true"/>
```

```
   <cs:Column ColumnKey="WineType" DisplayName="WineType" SpecColumn="false"/>
   ...
</cs:RecordDetails>
```

The assigned attributes of the requested record are in the `Record` element.

In addition, the attributes on the `Column` elements contain the following additional property information:

- `ColumnKey` identifies the name (in an NCName format) of the standard or managed attribute.
- `DisplayName` specifies the name of the attribute in an easy-to-understand format.
- `SpecColumn` identifies whether the attribute is a primary key (i.e., whether it is a single-assign property with a unique key/value pair). Note that all unique, single-assign properties in the Dgraph are returned, even if they are not assigned to the record.

The application front end can iterate through this record details list, extract the identifying information for the record, and display a table containing the results.

## Displaying record counts

The `RecordCountConfig` type is used for counting the records in the state.

### RecordCountConfig syntax

The format of the `RecordCountConfig` type is:

```
<RecordCountConfig Id="?">
   <StateName>?</StateName>
</RecordCountConfig>
```

The meanings of the `RecordCountConfig` elements and attributes are as follows:

| Element/Attribute | Meaning |
|---|---|
| `Id` | Required. An arbitrary identifier for this `RecordCountConfig`. |
| `StateName` | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the `StateName` element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the `StateName` element is used to reference that named state (as the state will be used in any event in the `RecordCountConfig`).<br>• If the request has an unnamed state, then the `StateName` element cannot be used. |

### RecordCountConfig example

This example performs a record search and uses a `RecordCountConfig` to count the number of records that were returned:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
```

```
      <TextSearchFilter Key="Flavors" Mode="AllPartial" Language="en">oak</TextSearchFilter>
   </State>
   <RecordCountConfig Id="NumRecs"/>
</Request>
```

Note that the `StateName` was not used because the state is not named.

# Performance impact when working with records

There are two scenarios that can have a performance impact when requesting or displaying records.

## Performance impact of requesting large numbers of records

Requesting a large number of records at once can reduce memory usage in your front-end application if the response is handled by a streaming parser in the front-end application.

## Performance impact when displaying attribute values

Displaying too many attribute values can affect performance.

The main purpose of attribute values is to enable navigation through the records. Therefore, the default behavior of the Oracle Endeca Server is to return attribute values on records only when a record query request has been made (not for navigation-type query requests). You can change this behavior. However, requesting the Oracle Endeca Server to return too many attribute values can cause a performance hit.

Chapter 12

# Sorting Records

Sorting allows you to define the order of records returned with each navigation query. It can be configured globally or per-query.



## About record sorting

When making a basic navigation request, you may define a series of attributes and order (Ascending or Descending) of pairs.

The Oracle Endeca Server returns query results in a Descending order on the primary key for returned records. You cannot change the default record sort order for the system.

All of the records corresponding to a particular navigation state are considered for sorting, not just the records visible in the current request. For example, if a navigation state applies to 100 bicycles, all 100 bicycles are considered when sorting, even though only the first ten bicycle records may be returned with the current request.

Record sorting only affects the order of records. It does not affect the ordering of attributes or attribute values that are returned for query refinement.

Note that all attributes are automatically enabled for record sorting when they are created. Therefore, no attribute configuration is required for sorting.

## Global sort order of records

This topic discusses the global sort order of records.

Once the records have been added to the data domain, the Oracle Endeca Server maintains data files for the records in memory. The following rules apply to how the records are sorted in the results returned by the Oracle Endeca Server in response to queries:

- Records are sorted according to the sort order that you specified, if any.
- Even if you specified a sort order, it may not have uniquely determined the resulting order of records — this usually happens when some records only differ in attributes that were not included in the sort specification. In such cases, the Dgraph tie-breaks the sorting results at random.

- Subsequent requests with the same query will result in the same order (the tie-break is consistent) unless you have modified the records in any way between requests. For example, the order will change if you delete any of the records and add them to the data domain again, even if they are identical.

Note that when a sorted record result list is requested, string values will be sorted case-insensitively, with ties broken with a case-sensitive comparison (upper-cased letters will rank above lower-cased letters). For example, for the six records **A**, **B**, **C**, **a**, **b**, and **c**, the resulting sort order will be:

```
A
a
B
b
C
c
```

# Query-time sort ordering

On a per-query basis, you can specify a key on which to sort the records and a sort direction.

You can add a `Sort` type to a `RecordListConfig` configuration that lets you specify a key to sort on and the sort direction. The `Sort` format is:

```
<Sort Key="?" Direction="?">
   <GeocodeReferencePoint latitude="?" longitude="?"/>
</Sort>
```

where:

- `Key` is mandatory and specifies the name of a standard or managed attribute based on which sorting is performed.
- `Direction` is optional and is either `Ascending` (for an ascending order) or `Descending` (for a descending order). `Ascending` is used as the default if `Direction` is not specified.
- `GeocodeReferencePoint` is optional and is used for geospatial sorting. For details on geospatial sorting, see *Geospatial sorting on page 150*.

Both the attribute name and the sort direction are case sensitive.

The following example shows how to specify a sort order:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <Language>en</Language>
         <State>
            <Name>AllRecs</Name>
            <RecordKind>data</RecordKind>
         </State>
         <RecordListConfig Id="RecordList" MaxPages="20">
            <StateName>AllRecs</StateName>
            <RecordsPerPage>10</RecordsPerPage>
            <Page>1</Page>
            <Sort Key="ModelName" Direction="Ascending"/>
         </RecordListConfig>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The example specifies an ascending sort order based on the `ModelName` attribute.

### Sorting behavior for records without a sort-key value

If an Endeca record does not include a value for the specified sort key, that record is sorted to the bottom of the list, regardless of the sort order.

For example, the following record set is sorted by the P_Year attribute in an ascending order. Note that Record 4 has no P_Year assignment value.

```
Record 1 (P_Year 1998)
Record 2 (P_Year 2010)
Record 3 (P_Year 2013)
Record 4 (no P_Year property value)
```

If the sort order is reversed to P_Year descending, the new result set would appear in the following order:

```
Record 3 (P_Year 2013)
Record 2 (P_Year 2010)
Record 1 (P_Year 1998)
Record 4 (no P_Year property value)
```

Thus, Record 4 will always appear last because it has no P_Year attribute assignment.

## Geospatial sorting

You implement query-time geospatial sorting by using a geocode attribute as a sort key.

Geocode attributes represent latitude and longitude pairs to Endeca records. Result sets that have geocode attribute assignments can be sorted by the distance of the values of the geocode assignment values to a given geocode reference point.

For example, if the records of a particular data set represent individual books that a large vendor has for sale at a variety of locations, each book could be tagged with a geocode attribute (for example, named Location) that holds the store location information for that particular book. Users could then filter result sets to see only books that are located within a given distance, and then sort those books so that the closest books display first.

A geocode attribute can be configured as a multi-assign attribute, which means that an Endeca record may have more than one geocode location. In this case, the Dgraph compares the query's geocode reference point to all geocode values on the record and returns the record with the closest distance to the reference point.

**Note:** This feature is supported only if you use a Conversation Web Service request. It is not supported in Studio's user interface.

### Specifying the geocode reference point in RecordListConfig

The `Sort` type in a `RecordListConfig` configuration lets you specify a geocode reference point, with this format:

```
<Sort Key="?" Direction="?">
   <GeocodeReferencePoint latitude="?" longitude="?"/>
</Sort>
```

`Key` is the name of the geocode attribute on which to sort. `Direction` (which is optional) is either `Ascending` for an ascending order (which is the default) or `Descending` for a descending order.

In the `GeocodeReferencePoint` element, you specify the geocode coordinates as doubles:

- `latitude` is the latitude of the location in whole and fractional degrees. Valid values are from a minimum of -90.0 to a maximum of 90.0 (inclusive). Positive values indicate north latitudes and negative values indicate south latitudes.
- `longitude` is the longitude of the location in whole and fractional degrees. Valid values are from a minimum of -180.0 to a maximum of 180.0 (inclusive). Positive values indicate east longitudes and negative values indicate west longitudes.

## Geocode sort example

The following request uses the Location geocode attribute as the sort key.

```
<Request>
   <Language>en</Language>
   <State>
      <Name>GeoQuery</Name>
      <SelectionFilter Id="SelFlt">
         <filterString>WineID > 10</filterString>
      </SelectionFilter>
   </State>
   <RecordListConfig Id="Results" MaxPages="20">
      <StateName>GeoQuery</StateName>
      <RecordsPerPage>20</RecordsPerPage>
      <Sort Key="Location" Direction="Ascending">
         <GeocodeReferencePoint latitude="42.365615" longitude="-71.075647"/>
      </Sort>
   </RecordListConfig>
</Request>
```

The records are returned in a `RecordList` type.

## Computed distances in the response

In a geospatial sort, the Dgraph Engine creates a pair of computed dynamic properties for each record returned, as illustrated by this abbreviated example for one record:

```
<cs:RecordListEntry>
   <cs:Record>
      <cs:attribute name="Flavors" type="mdex:string">Cherry</cs:attribute>
      ...
      <cs:attribute name="WineType" type="mdex:string">Merlot</cs:attribute>
   </cs:Record>
   <cs:ComputedProperties>
      <cs:GeocodeDistance queryString="Location(42.3656,-71.0756)" units
="kilometers">296.17527882382</cs:GeocodeDistance>
      <cs:GeocodeDistance queryString="Location(42.3656,-71.0756)" units="miles">184.034729178036<
/cs:GeocodeDistance>
   </cs:ComputedProperties>
</cs:RecordListEntry>
```

The `GeocodeDistance` dynamic properties show the distance (in kilometers and miles, respectively) between the record's geocode address and the geocode reference point specified in the sort key. A record that does not have a geocode attribute assignment will have "NaN" (Not-a-Number) as the geocode distance.

These `GeocodeDistance` properties are intended for display purposes and are not persistent (that is, they are not added to the records).

# Troubleshooting sorting problems

This topic presents some approaches to solving sorting problems.

If the returned records do not seem to respect the sort key parameter, there are some potential problems:

- Was the attribute specified as numeric when it is actually alphanumeric? Or vice versa? In this case, Endeca Server returns a valid response, but the sorting may be incorrect.
- If a record has multiple attribute value assignments from a single attribute, Endeca Server sorts the records based on the first value associated with the key. If the application is displaying the last value, the records will not appear to be sorted correctly. In general, attributes that are used for sorting should only have one value assigned per record.
- If certain records in a data domain lack a sort-key value, they will always appear last in a result set. Therefore, if you reverse a sort order on a record set containing such records, the order of the entire record set will not be reversed — the records without a sort key value always sort at the end of the set.

Chapter 13

# Internationalized Data

This section describes how to include internationalized data in an Endeca data domain.



## Overview of using internationalized data

Oracle Endeca Server support for the Unicode Standard version 4.0 allows an Endeca data domain to process and serve data in many of the world's languages.

At either data ingest time (or later via a Configuration Web Service operation), you can specify that a given standard attribute will use internationalized data when it is provided in a native encoding. At query time, you can specify the language to be used for the record search or value search.

The section makes the following assumptions:

- If working with Chinese, you are familiar with the encoding and character sets (Traditional versus Simplified, Big5, GBK, and so on).
- If working with Chinese or Japanese, you know that these languages do not use white space to delimit words.
- If working with Japanese, you are familiar with the shift_jis variants and how the same character can be represented either the Yen symbol or the backslash character.

For more information about the Unicode Standard and character encoding, see *http://unicode.org*.

### Overview of supported language features

The following is a high-level list of which features are supported for international languages:

| Feature | Language support |
|---|---|
| Auto-correction spelling | Language-specific auto spelling correction is available for supported languages (i.e., spelling dictionaries are available for all supported languages). |

| Feature | Language support |
|---|---|
| Stemming and lemmatization | Language-specific stemming and lemmatization is available. Note that stemming is not available for segmented (non-whitespace) languages, such as Japanese, Chinese, and Thai. |
| Did You Mean (DYM) suggestions | Language-specific DYM is available for all supported languages. |
| Snippeting | Available for all supported languages. |
| Thesaurus | One language-agnostic thesaurus is available for use with queries in any of the supported languages (i.e., language-specific thesauruses are not supported). |
| Search characters | Available only for the `unknown` language identifier. |
| Stop words | Available only for the `unknown` language identifier. |
| Language auto-detection | Auto-detection of languages at ingest or query time is not supported. The user must explicitly specify the language for the PDR or the query. |
| Language collation | Language-specification collation (sorting) is not available for the supported languages. |

### Diacritic folding

Diacritic folding is the default behavior for all supported languages (including "unknown") during record searches. This feature is the automatic mapping of ISO-Latin1 international characters to ASCII equivalents in record search queries. It basically ignores character accents so that search queries containing international characters will match against Anglicized result text. For example, an English query for "café" will match "café" in records. Note that you cannot disable this diacritic folding behavior.

## Supported languages

You use a language code to identify a language.

Language codes must be specified as valid RFC-3066 language code identifiers. The supported languages and their language code identifiers are:

- Arabic — `ar`
- Catalan — `ca`
- Chinese, simplified — `zh_CN`
- Chinese, traditional — `zh_TW`
- Croatian — `hr`
- Czech — `cs`
- Danish — `da`

- Dutch — `nl`
- English, American — `en`
- English, British — `en_GB`
- Finnish — `fi`
- French — `fr`
- German — `de`
- Greek — `el`
- Hebrew — `he`
- Hungarian — `hu`
- Italian — `it`
- Japanese — `ja`
- Korean — `ko`
- Norwegian Bokmal — `nb`
- Norwegian Nynorsk — `nn`
- Persian — `fa`
- Polish — `pl`
- Portuguese — `pt`
- Portuguese, Brazilian — `pt_BR`
- Romanian — `ro`
- Russian — `ru`
- Serbian, Cyrillic — `sr_Cyrl`
- Serbian, Latin — `sr_Latn`
- Slovak — `sk`
- Slovenian — `sl`
- Spanish — `es`
- Swedish — `sv`
- Thai — `th`
- Turkish — `tr`
- unknown (i.e., none of the above languages) — `unknown`

The language codes are case insensitive.

Note that an error is returned if you specify an invalid language code.

With the language codes, you can specify the language of the text to the Dgraph during a record search or value search query, so that it can correctly perform language-specific operations.

### How country locale codes are treated

A country locale code is a combination of a language code (such as `es` for Spanish) and a country code (such as `MX` for Mexico or `AR` for Argentina). Thus, the `es_MX` country locale means Mexican Spanish while `es_AR` is Argentinean Spanish.

If you specify a country locale code for a `Language` element, the software ignores the country code but accepts the language code part. In other words, a country locale code is mapped to its language code and only that part is used for tokenizing queries or generating search indexes. For example, specifying `es_MX` is the same as specifying just `es`. The exceptions to this rule are the codes listed above (such as `pt_BR`).

Note, however, that if you create a standard attribute and specify a country locale code in the `mdex-property_Language` field, the attribute will be tagged with the country locale code, even though the country code will be ignored during indexing and querying.

### Language-specific dictionaries and indices

The Dgraph has two spelling correction engines. If the `mdex-property_Language` property in a PDR is set to `en`, then spelling correction will be handled through the English spelling engine (and its English spelling dictionary); if it is set to any other value, then spelling correction will use the non-English spelling engine (and its language-specific dictionaries). All dictionaries are generated from the data records in the Dgraph, and therefore require that the standard attribute PDRs be tagged with a language ID.

All dictionary files are stored in the data domain's index directory.

## Setting language identifiers

The following topics describe how to specify language identifiers for your application.

In an Oracle Endeca Server application, language can be specified in two places:

- The language of a standard or managed attribute can be specified in the PDR of that attribute.
- The language of a search query can be specified with search configuration options.

Keep in mind that the following language-identification scenarios are not supported:

- A global language identifier (for all of your data and queries) is not supported. However, you can set a global PDR language code that is used when PDRs are automatically created by the DIWS (Data Ingest Web Service) and Bulk Load interfaces.
- A per-record language identifier is not supported. Language codes can be set only on attributes, not on records.
- The use of multiple language identifiers for a single search query is not supported. That is, each query can have a maximum of one language identifier, which means that the language can vary on a per-query basis. A per-query language identifier should be used in your front-end application if the language varies on a per-query basis.

### Language effect on documents and searches

For every document, the language of the corresponding PDR determines:

- How it is tokenized
- How it is normalized

- In what language word forms are returned for its terms
- Which language's wordform expansion indexes do the returned forms contribute to
- Which language's spelling dictionary its terms contribute to

For every search, the language configured on the search determines:

- How it is tokenized
- How it is normalized
- In what language are word forms returned for its terms
- Which language's spelling dictionary to use for spelling-related re-queries

*Setting PDR language identifiers*

*Global PDR language code*

*Specifying a per-query language code*

## Setting PDR language identifiers

A language can be set for each standard attribute.

A Property Description Record (PDR) has an `mdex-property_Language` field that specifies the language of that standard attribute. This field takes one of the supported language codes listed in *Supported languages on page 154*.

This brief example creates a standard attribute named Beschreibung, whose language will be German (the `de` language code):

```
<mdex:record>
  <mdex-property_Key>Beschreibung</mdex-property_Key>
  <mdex-property_Type>mdex:string</mdex-property_Type>
  <mdex-property_Language>de</mdex-property_Language>
  ...
</mdex:record>
```

If it is not explicitly set, `mdex-property_Language` defaults to the `unknown` language identifier when the standard attribute is created by the system.

For example, your data may have an English standard attribute called Description with its language code set to `en`, and a Spanish attribute called Descripción with a language code of `es`. In this case, because an individual record can have both English and Spanish text, you can see that an attribute language code is more appropriate than a per-record language code.

### Changing the PDR language code

Typically, you set the `mdex-property_Language` property when creating the record schema for your data set. However, the language code can later be changed via the Configuration Web Service's `updateProperties` operation.

The following is an example of the `updateProperties` operation, in which the language of the Province standard attribute is changed to French:

```
<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
     xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
```

```
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:updateProperties>
            <ns1:record>
               <mdex-property_Key>Province</mdex-property_Key>
               <mdex-property_Language>fr</mdex-property_Language>
            </ns1:record>
         </ns:updateProperties>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

Keep in mind that changing the value of `mdex-property_Language` for an existing attribute will force a regeneration of the text search indexes, which is a potentially time-consuming operation.

## Global PDR language code

The global PDR language code determines which language code is used when a PDR is automatically created.

If a standard attribute is automatically created at ingest time (by the Data Ingest Web Service or the Bulk Load interface), or if you do not specify a language code when creating your attribute schema, then the value of the `mdex-property_Language` property for a PDR will be set to the global PDR language code. This is also the case if you partially define a PDR but do not set the `mdex-property_Language` property.

The value of the global PDR language code is `unknown` by default. However, you can use the Configuration Web Service's `setPropertyDefaultLanguage` operation to change it to a value of your choice. This example sets the language code to `de` (German):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
  xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:setPropertyDefaultLanguage>de</ns:setPropertyDefaultLanguage>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

The setting of this value is stored on disk in the index files for the specific Endeca data domain, so that it is available across restarts of that data domain. You can use the Configuration Web Service's `getPropertyDefaultLanguage` operation to retrieve the current setting of the global PDR language code.

## Specifying a per-query language code

Record search and value search queries can specify the language used for those queries.

The `TextSearchFilter` type has a `Language` attribute that you use to tell the Dgraph what language record (full-text) queries are in. Likewise, the `ValueSearchConfig` type has a similar `Language` attribute for value search queries. This per-query language code enables the Dgraph to select the appropriate dictionary for a given query.

If no per-query language ID is specified, the Dgraph uses the `unknown` language identifier.

The following code snippets show how to set English (using its language code of "en") as the language of any text portion of the query (such as search terms).

### Example of language setting for record search

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <State>
            <Name>MyRecSearch</Name>
            <TextSearchFilter Key="Description" Mode="AllPartial" EnableSnippeting="false"
                Language="en">mountain</TextSearchFilter>
         </State>
         <RecordListConfig Id="RecordList" MaxPages="20">
            <StateName>MyRecSearch</StateName>
            <Column>Description</Column>
            <RecordsPerPage>5</RecordsPerPage>
         </RecordListConfig>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

### Example of language setting for value search

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <State>
            <Name>MyProductSearch</Name>
            <RecordKind>data</RecordKind>
         </State>
         <ValueSearchConfig Id="ValueSearch" MaxPerProperty="5"
              RelevanceRankingStrategy="static (nbins,descending)" Mode="Any" Language="en">
            <StateName>MyProductSearch</StateName>
            <SearchTerm>aluminum</SearchTerm>
         </ValueSearchConfig>
    </Request>
  </soap:Body>
</soap:Envelope>
```

# Using custom dictionaries

You can optionally add a custom dictionary to supplement a default dictionary for any supported language.

The use of a custom dictionary may be necessary if searches for terms that you know exist in your data are not producing the expected results. The custom dictionary is a UTF-8 encoded file that is line oriented and tab delimited. Each line in the file represents an entry to supplement the primary dictionary.

The generic syntax for a line in the custom dictionary file is:

```
COMMAND value1 value 2 ...
```

The `COMMAND` should be set only to `STEM` (for dictionary terms) or `COMPOUND` (for decompounding). Each `value` is tab delimited and depends on the `COMMAND`.

### Dictionary terms

One use of a custom dictionary is to add new dictionary terms. (A dictionary term is also called a lemma.) Once a term is added to the dictionary, all morphological rules will apply to it. For example, adding a new noun will allow its plural form to stem to the lemma.

The generic syntax for a `STEM` line in the custom dictionary file is:

```
STEM new_term POS [,POS2 ...]
```

Each line beginning with `STEM` represents a lemma entry that includes:

- *new_term* is a tab-delimited simple text string that represents a lemma.
- *POS* is a valid part of speech listed below. At least one part of speech is required. Multiple parts of speech are delimited by a comma. Note that the parts of speech are case sensitive.

You can specify the part of speech attributes by their full name or abbreviation (in parentheses):

- `noun (N)` - a simple noun, like table, book, procedure
- `nounProper (propN)` - a proper name, for person, place, etc., typically capitalized, like Zachary, Supidito, Susquehanna
- `verb (V)` - any verb in its dictionary form, like deconstruct, upsell, skate
- `adjective (Adj)` - modifiers of nouns, typically can be compared (green, greener, greenest), like fast, trenchant, pendulous
- `adverb (Adv)` - any general modifier of a sentence that may modify an adjective or verb or may stand alone, like slowly, yet, perhaps
- `preposition (Prep)` - a word that forms a prepositional phrase with a noun, like off, beside, from. Used for postpositions too, in languages that have postpositions of similar function.
- `punct (Punct)` - any non-letter symbol that is treated as a unit by itself, like %, $, ]
- `pronoun (Pro)` - any pronominal form, including personal pronouns (I, they), demonstrative pronouns (those, this), relative pronouns (who, which, wherever)
- `interrog (Wh)` - an interrogative word, like who, why, when, where, how
- `determiner (Det)` - words that carry grammatical information about a noun group, for example definite/indefinite, like the, a, an
- `particle (Part)` - small, invariant words that convey grammatical information; also used for interjections.
- `conjunction (Conj)` - conjunctions that introduce a subordinate clause, e.g. although, because, while, and conjunctions that introduce a coordinate clause, e.g. and, or, yet
- `numCardinal (Card)` - cardinal numbers, like thirteen, 100, five
- `numOrdinal (Ord)` - ordinal numbers, like thirteenth, 100th, fifth

For example, this German custom dictionary shows three entries. Each entry is marked with the `N` attribute to indicate it is a noun:

```
STEM aalglatt N
STEM aalglatte N
STEM aalglatter N
```

## Decompounding

You can manually configure a custom dictionary to define components of compound words. This can be useful if existing language dictionaries do not align with the usage of the language in a region or market, or if existing libraries have not kept up with changes to the language. A record search query for any of the components in a compound word also returns the compound as a match.

For example, the German orthography reform of 1996 introduced a standard set of rules for compound words, but these rules are not always followed. For this and similar such cases, you may wish to explicitly configure dictionary entries that mark the divisions within compound words.

The generic syntax for a `COMPOUND` line in the custom dictionary file is similar to the `STEM` syntax, including the *POS* attributes.

For example, you may wish to decompound the German word "Binnenschiffahrt" (which refers to transport along inland rivers). You might wish to add two versions: one that adheres to the German orthography reform standards of 1996 and one that reflects the earlier spelling of the word:

```
COMPOUND Binnenschifffahrt Binnen|Schiff|Fahrt N
COMPOUND Binnenschiffahrt Binnen|Schiff|Fahrt N
```

Note that the component words of a compound word must each exist in the dictionary. For the example above, this means that the dictionary must include individual entries for "binnen", "schiff", and "fahrt".

*Creating a custom dictionary*

## Creating a custom dictionary

This topic provides the steps to create your custom dictionary file.

To create a custom dictionary:

1. Start a text editor that supports UTF-8 characters and enables you to edit the language you want to supplement.
2. Create a new UTF-8 encoded file.
3. Add words to the dictionary. Start each word on a separate line that begins with the command `STEM` or `COMPOUND`, followed by the word or character, any optional attributes, and then a carriage return.
4. Optionally, add comments to the file.

   Comments must begin with a pound sign (#). You can also have blank lines in this file.
5. Save the dictionary file with the filename `dictionary.<lang_code>.dict`, where `<lang_code>` is one of the supported language codes (such as `dictionary.de.dict` for a German custom dictionary).

   Note that the dictionary name does not need to include a region code unless you are using a language such as simplified Chinese (`zh_CN`) or Brazilian Portuguese (`pt_BR`). For example: `dictionary.zh_CN.dict`.
6. Place the file in the `$ENDECA_HOME/endeca-server/dgraph/olt` directory.
7. Re-index your data by re-ingesting it with your ETL tool.

# Viewing Dgraph logs

Log messages output by the Dgraph are in UTF-8 encoding.

Most common UNIX/Linux shells and terminal programs are not set up to display UTF-8 by default and will therefore display some valid characters as question marks (?). If you find unexpected question marks in the data, first validate that it is not simply a display issue. Try the `od` command on Linux, or use a UTF-8 compatible display.

# Part IV

## Attributes, Refinements, and Groups

Chapter 14

# Managed Attributes

Managed attributes are those attributes that, unlike standard attributes, carry additional information in them, such as hierarchy, a set of predefined values, and a set of synonyms or ranks. This section describes how to build the necessary parts of the records schema (PDRs and DDRs) that define managed attributes themselves.



## About managed attributes and their values

Managed attribute values represent specific value assignments on managed attributes in your data domain. Through managed attribute values, you can express hierarchical relationships and other useful information (such as synonyms and ranking), on record refinements in your data domain.

Typically, you use managed attribute values for these purposes:

- To represent hierarchy information on data records. For example, a managed attribute "location" can have values for countries, such as USA and Canada, states, such as MA and NY, and cities, such as Boston and New York City. All these managed attribute values are organized in an hierarchy — they have information about themselves via a unique spec that identifies them in the Endeca Server index, and their parent managed attribute value. In other words, you can create a parent managed attribute value "USA", a child managed attribute value "MA (Massachusetts)", and a grand-child managed attribute value "Boston".
- To hold a set of predefined values. For example, for a managed attribute "currency", you can define a limited set of values, "dollars" and "euros".
- To hold a set of synonyms for each value. For example, you can add a synonym "Beantown" to a managed attribute value "Boston".
- To add predefined static ranks to managed attribute values, to control how they are sorted when displayed to the end users in the front-end application, such as Studio.

You add managed attribute values before loading the source records into the data domain. Next, when you load source records that include these values (such as "MA", or "Boston"), Endeca Server recognizes them as managed attribute values, thus letting your end users navigate and search on them, utilizing their hierarchy, as well as static ranks and synonyms (if specified).

Inside the Endeca Server index for each data domain, each of the added managed attribute values, (such as "MA" and "Boston" in our example), are themselves represented as records. These records are described by the PDRs, DDRs, and Managed Attribute Value Description Records (MAVDRs) that are created in the index, when you add managed attributes and their values.

A request to add a managed attribute value record using the Configuration Web Service has this structure:

```
<ns:configTransaction>
   <ns:putManagedAttributeValues>
      <ns1:mav>
         <name>?</name>
         <spec>?</spec>
         <parent>?</parent>
         <managedAttribute>?</managedAttribute>
         <synonym>?</synonym>
         <synonym>?</synonym>
         <rank>?</rank>
      </ns1:mav>
   </ns:putManagedAttributeValues>
</ns:configTransaction>
```

In this structure, the `spec`, `parent`, and `managedAttribute` elements are required, all other elements are optional.

The following table describes each of the elements in the `mav` element of `putManagedAttributeValues`. For each of these elements, their value represents an assignment on some attribute. The corresponding attribute can belong to one of the description records (PDR, MAVDR) created in the index once you add this particular managed attribute value:

| Element | Type | Description |
|---|---|---|
| `name` | string | Optional. The display name of the managed attribute value record. You define this property when adding a managed attribute value. For example, you can define a managed attribute value with the name "Boston":<br>`<mav><name>Boston</name>...</mav>`<br>Inside the schema for your data records, when you create this managed attribute value, its MAVDR is created. The MAVDR gets an assignment on `mdex-dimension-value_Name`:<br>`<attribute name="mdex-dimension-value_Name" type ="mdex:string">Boston</attribute>` |

| Element | Type | Description |
|---|---|---|
| `spec` | string | Required. A unique identifier, or a spec, of the managed attribute value record. For example, you can define a managed attribute value with the spec "bos":<br>`<mav>`<br>`<spec>bos</spec>`<br>`...`<br>`</mav>`<br>Inside the schema for your data records, this value, in turn, is an assignment on the `mdex-dimension_`*`my_attr_key`*`_Spec` attribute that is created when you add a DDR for the associated managed attribute with the key *`my_attr_key`*. For example, for a managed attribute with the key "location", when you create its DDR, another record is automatically created, — namely, a PDR for `mdex-dimension_location_Spec`. It is always created as unique and single-assign, as values on it uniquely identify managed attribute values that you will be adding. Next, when you add a managed attribute value, such as "bos", the value "bos" becomes an assignment on `mdex-dimension_`*`location`*`_Spec`:<br>`<attribute name="mdex-dimension_location_Spec" type ="mdex:string">bos</attribute>`<br>  **Note:** If, when creating the DDR for the managed attribute with the key *`value`*, you do not specify your own value for its `mdex-dimension-value_Spec` attribute, the naming structure of the format `mdex-dimension_`*`value`*`_Spec` is used automatically by Endeca Server when it creates the PDR for the managed attribute value spec. |

| Element | Type | Description |
|---|---|---|
| `parent` | string | Required. The spec of the parent managed attribute value. Only one is allowed. To continue with the example, for a managed attribute value with the spec "bos", the parent managed attribute value spec is "MA" (it must already exist, in order to be specified):<br>`<mav>`<br>`<spec>bos</spec>`<br>`<parent>MA</parent>`<br>`...`<br>`</mav>`<br>Inside the schema for your data records, this value, in turn, is an assignment on the `mdex-dimension_`*`my_attr_key`*`_Parent` attribute that is created when you add a DDR for the associated managed attribute with the key *`my_attr_key`*. For example, for a managed attribute with the key "location", when you create its DDR, another record is automatically created, — namely, a PDR for `mdex-dimension_location_Parent`. It is always created as single-assign. Next, when you add a managed attribute value, such as "MA", the value "MA" becomes an assignment on `mdex-dimension_`*`location`*`_Parent`, for another managed attribute value, "bos":<br>`<attribute name="mdex-dimension_location_Parent" type ="mdex:string">MA</attribute>`<br>  **Note:** If, when creating the DDR for the managed attribute with the key *`value`*, you don't specify your own value for its `mdex-dimension-value_Parent` attribute, the naming structure of the format `mdex-dimension_`*`value`*`_Parent` is used automatically by the Endeca Server, when it creates the PDR for the parent managed attribute value. |
| `managedAttribute` | string | Required. The key (or spec) of the managed attribute to which this managed attribute value relates. Only one is allowed. You define it when adding a managed attribute value. For example, for the value Boston (whose spec is "bos"), the managed attribute key it is associated with is "location":<br>`<mav>`<br>`<spec>bos</spec>`<br>`<managedAttribute>location</managedAttribute>`<br>`...`<br>`</mav>`<br>Inside the schema for your data records, this value, "location", is a key of the managed attribute "location", in the DDR that defines this managed attribute:<br>`<mdex-dimension_Key>location</mdex-dimension_Key>`<br>In the MAVDR for a managed attribute value you are adding, it becomes an assignment to the managed attribute "location":<br>`<attribute name="mdex-dimension-value_Dimension" type ="mdex:string">location</attribute>` |

| Element | Type | Description |
|---|---|---|
| `rank` | integer | Optional. Static rank of the managed attribute value. Only one is allowed. This rank is an integer number according to which the Endeca Server ranks the managed attribute values within each managed attribute's hierarchy, when returning them to the end users. This rank affects the order in which values are displayed in the front-end application, such as Studio. You can optionally specify the rank, when adding a managed attribute value with the `putManagedAttributeValue` operation. `mdex-dimension-value_Rank`<br>`<ns1:mav>`<br>`<spec>bos</spec>`<br>`<rank>3</rank>`<br>`...`<br>`</mav>`<br>Inside the schema for your data records, the value of rank, such as "3", becomes an assignment on the `mdex-dimension-value_Rank` attribute of the MAVDR for the managed attribute you are adding:<br>`<attribute name="mdex-dimension-value_Rank" type="mdex:integer">3</attribute>` |
| `synonym` | string | Optional. List of one or more synonyms, for the managed attribute value. You can optionally specify one or more synonyms, when adding a managed attribute value. For example, for the value Boston, the synonyms can be "Beantown", and "the Hub":<br>`<ns1:mav>`<br>`<spec>bos</spec>`<br>`<synonym>Beantown</synonym>`<br>`<synonym>the Hub</synonym>`<br>`...`<br>`</mav>`<br>Inside the schema for your data records, the value of any synonym, such as "Beantown", becomes an assignment on the `mdex-dimension-value_Synonyms` attribute of the MAVDR for the managed attribute you are adding:<br>`<attribute name="mdex-dimension-value_Synonyms" type="mdex:string">Beantown</attribute>` |

To conclude, each managed attribute value record is identified by its own pair of spec and parent values that are unique to this managed attribute value. The following example illustrates how to add a managed attribute value, for a specific managed attribute.

### Example

Let's consider an example where you would like to create a managed attribute "location", with a set of managed attribute values associated with it. You start by creating a managed attribute itself. Since each managed attribute requires a PDR and a DDR, you first create the PDR with the spec "location", as in this abbreviated example:

```
<mdex-property_Key>location</mdex-property_Key>
```

Next, you create an associated DDR for this managed attribute, also with the spec "location":

```
<mdex-dimension_Key>location</mdex-dimension_Key>
```

Once you create this DDR, Endeca Server automatically creates two additional PDR records (if you do not specify your own values for them in the DDR):

```
<mdex-property_Key type="mdex:string" xmlns="">mdex-dimension_location_Parent</mdex-property_Key>
<mdex-property_Key type="mdex:string" xmlns="">mdex-dimension_location_Spec</mdex-property_Key>
```

Next, you can add three managed attribute values, using the `putManagedAttributeValues` operation of the Configuration Web Service (which will create a MAVDR for each added managed attribute value, internally):

```
<ns:configTransaction>
 <ns:putManagedAttributeValues>
  <ns1:mav>
     <name>USA</name>
     <spec>US</spec>
     <parent>/</parent>
     <managedAttribute>location</managedAttribute>
  </ns1:mav>
  <ns1:mav>
     <name>Massachusetts</name>
     <spec>MA</spec>
     <parent>US</parent>
     <managedAttribute>location</managedAttribute>
  </ns1:mav>
  <ns1:mav>
     <name>Boston</name>
     <spec>bos</spec>
     <parent>MA</parent>
     <managedAttribute>location</managedAttribute>
  </ns1:mav>
 </ns:putManagedAttributeValues>
</ns:configTransaction>
```

(For simplicity, namespaces are omitted in this example. They represent `xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"` and `xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09"`, respectively.)

When a managed attribute value is created, Endeca Server "attaches" it to a specific managed attribute. In this example, for a managed attribute "location", when you create a managed attribute value "Boston", the managed attribute's value spec, "bos", is assigned on the `mdex-dimension_location_Spec` that was created automatically when the "location" managed attribute was created, by adding its DDR. Similarly, the managed attribute's value parent ,"MA", is assigned on the `mdex-dimension_location_Parent`.

**Note:** Ranks and synonyms are omitted in this example. For information on how to add synonyms, see *Adding managed attribute values on page 170*. For information on how to add ranks, see *Adding and updating ranks on page 177*.

# Summary of operations

You can add and list managed attribute values. You can optionally assign their names, synonyms, or ranks at creation time. You can also update the display name and the rank of the managed attribute values without

stopping the data domain. You can delete managed attribute values by deleting the PDR that defines an associated managed attribute.

The following list summarizes operations you can perform:

- **Additions**. You can add a new managed attribute value using different methods:
  - Using the Data Ingest Web Service `addRecords` operation. You do this before loading data records, by creating the appropriate assignments on the new description record, (MAVDR). For information, see the *Oracle Endeca Server Data Loading Guide*.
  - Using the Configuration Web Service operation `putManagedAttributeValues`.
- **Updates**. You can update managed attribute value records only with changes to ranking and display name, using the `putManagedAttributeValues` operation, or an operation in the Data Ingest Web Service. You cannot change their hierarchy or other characteristics, once you add them.
- **Deletions**. The only way to delete an individual managed attribute value is to delete the PDR for an associated managed attribute. When you delete a PDR, Endeca Server deletes the managed attribute and all its values.
- Additionally, you can add ranks and synonyms for managed attribute values. See *About ranks and synonyms on page 169*.

# About ranks and synonyms

When you add managed attribute values, you can specify their synonyms and ranks.

Ranks and synonyms work as follows:

- **Rank**. You can assign a static rank to each managed attribute value. This lets you impose an order on values in your refinements, when returning refinement values to the end users. You can update the static ranking of managed attribute values on an active data domain. Rank is specified by assignments on `mdex-dimension-value_Rank` attribute. You can add rank when adding a managed attribute value with the Data Ingest Web Service `addRecords` sub-operation. An easier way to add rank is with `putManagedAttributeValues` of the Configuration Web Service (this automatically makes a rank assignment on `mdex-dimension-value_Rank`).
- **Synonyms**. You can specify one or more synonyms in a managed attribute value so that end users can search for other text strings and still obtain the same records as they would get while searching for the original managed attribute value name. To add synonyms when adding attribute values to the Endeca data domain, use the `putManagedAttributeValues` operation of the Configuration Web Service. Alternatively, you can add assignments on `mdex-dimension-value_Synonyms` attribute, when adding managed attribute values through Data Ingest Web Service. You can add synonyms only to child managed attribute values, not to top-most (root) attribute values (which are automatically created with the value "/").

  You can add synonyms only when you initially create managed attribute value records and before loading the data records that have assignments from managed attribute values This means that synonyms can only be added only to an empty data domain that contains only the schema for the source records (non-data records), but does not yet contain data records. Once added, synonyms cannot be updated. If you want to update synonyms, you must delete the managed attribute value (which, in turn, requires deleting the associated managed attribute).

# Adding managed attribute values

Typically, you add more than one managed attribute value. To add a set of them, you first define a managed attribute with which the values will be associated. Next, you add the managed attribute values, associating them with the managed attribute.

The procedure in this topic utilizes an example to illustrate how to add a set of managed attribute values that define a hierarchy. It builds an hierarchy of managed attribute values for the managed attribute "location", by adding parent managed attribute value "USA", followed by managed attribute values "Massachusetts", and "New York State", and their descendant managed attribute values "Boston", and "New York".

**Note:** You can add managed attribute values using two different web services: Configuration Web Service and Data Ingest Web Service. This topic discusses adding them with `putManagedAttributeValues` of the Configuration Web Service. Since, after they are added, managed attribute values are represented as records in the Endeca Server index for the data domain, you can also use the Data Ingest Web Service to add them as non-data records, before adding any data records. For information, see the *Oracle Endeca Server Data Loading Guide*.

The examples of Configuration Web Service requests in this topic are abbreviated and do not show namespaces. They use the following namespaces:
`xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"` and
`xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09"`.

Before you add one or more managed attribute values, note the following requirements:

- A managed attribute must exist in the index of the data domain. This means that its associated PDR and DDR must be created. Note that if a DDR does not exist, you can create it in the same request in which you are adding managed attribute values.
- When creating a DDR (as shown in the procedure in this topic), you can omit specifying the values for `mdex-dimension-value_Parent` and `mdex-dimension-value_Spec`, as these properties on the DDR are optional. If you do not explicitly provide values for them in the `putDimension` request that creates a DDR for the managed attribute, Endeca Server automatically creates them and assigns their names. For example, for a managed attribute "location", if parent and spec are not specified, `mdex-dimension_location_Parent`, and `mdex-dimension_location_Spec` are created. If you specify these properties, you can provide your own names. Whether or not you specify these properties, once you add a DDR, Endeca Server creates two PDR records in the index, for each of these properties. These two description records represent non-data records, and are needed to provide the internal structure that uniquely identifies each added managed attribute value record (via its value on the spec), and also defines the hierarchy (via its values on both the spec and the parent attributes).
- When creating the top-most managed attribute value for the managed attribute, specify the sign `"/"` for its parent managed attribute value. This is because, when a managed attribute is created, the spec for the root managed attribute value is also created automatically in the index, and its value is "/".
- When you add a managed attribute value with `putManagedAttributeValues`, its name, synonyms, and rank attributes are optional. All other parameters (spec, parent managed attribute value, and managed attribute spec) are required.
- You can optionally add its synonyms, which Endeca Server will use in searches. Since it is not possible to change or update synonyms on already added managed attribute values, plan which synonyms you are going to require. If you later need to add more synonyms, delete the existing managed attribute (this deletes all associated managed attribute values), and replace it with new values and synonyms.

- You can update a managed attribute value with a new name. In this case, when you use the `putManagedAttributeValues` for an existing value but specify a new name, the value is updated with this name.
- You can also add static ranks (and update them) to the managed attribute values. Ranks are not included in the examples in this topic. For information on adding them, see *Adding and updating ranks on page 177*.
- You can add more than one managed value attribute in one operation. However, if any of the values are specified incorrectly, causing Endeca Server to issue an error, the entire Configuration Web Service request fails and no values are added.

To add a set of managed attribute values:

1. In soapUI or another tool for issuing web service requests, access the Configuration Web Service, as in this example:

   ```
   http://localhost:7001/endeca-server/ws/config/data_domain?wsdl
   ```

   where `localhost` and `7001` are the Endeca Server host and port, and `data_domain` is the name of the data domain.

2. Start by creating a structure needed for the managed attribute.

   (a) Create a PDR "location":

   ```
   <ns:configTransaction>
     <ns:putProperties>
       <ns1:record>
         <mdex-property_DisplayName>location</mdex-property_DisplayName>
         <mdex-property_IsSingleAssign>false</mdex-property_IsSingleAssign>
         <mdex-property_IsTextSearchable>true</mdex-property_IsTextSearchable>
         <mdex-property_IsUnique>false</mdex-property_IsUnique>
         <mdex-property_IsPropertyValueSearchable>true<
   /mdex-property_IsPropertyValueSearchable>
         <mdex-property_Key>location</mdex-property_Key>
         <mdex-property_TextSearchAllowsWildcards>true<
   /mdex-property_TextSearchAllowsWildcards>
         <mdex-property_Type>mdex:string</mdex-property_Type>
       </ns1:record>
     </ns:putProperties>
   </ns:configTransaction>
   ```

   This creates the standard attribute "location", which is needed to create a managed attribute.

   Notice that in this example, the value for the `mdex-property_IsSingleAssign` property is `false`. The setting of `false` is required if you want to assign multiple values on the associated managed attribute.

   (b) Create a DDR "location":

   ```
   <ns:configTransaction>
     <ns:putDimensions>
       <ns1:record>
         <mdex-dimension_Key>location</mdex-dimension_Key>
         <mdex-dimension_EnableRefinements>true</mdex-dimension_EnableRefinements>
         <mdex-dimension_IsDimensionSearchHierarchical>true<
   /mdex-dimension_IsDimensionSearchHierarchical>
         <mdex-dimension_IsRecordSearchHierarchical>true<
   /mdex-dimension_IsRecordSearchHierarchical>
       </ns1:record>
     </ns:putDimensions>
   </ns:configTransaction>
   ```

This creates the managed attribute "location" to which you are adding a set of managed attribute values.

This request also creates two PDRs in the index for spec and parent, as evident if you run `listProperties`. Namely, this creates:

```
<mdex-property_Key type="mdex:string" xmlns="">mdex-dimension_location_Parent<
/mdex-property_Key>
 <mdex-property_Key type="mdex:string" xmlns="">mdex-dimension_location_Spec<
/mdex-property_Key>
```

- The `mdex-dimension_location_Spec` is the name of the attribute whose assignment is the spec of the managed attribute value you are adding, and it uniquely identifies the managed attribute value record.
- The `mdex-dimension_location_Parent` is the name of the attribute whose assignment is the spec of the parent managed attribute value, for this managed attribute value.

The assignments on spec and parent attributes are important — this is the mechanism by which the Endeca Server creates managed attribute values as records, inside its index for the data domain.

If you do not include these two attributes in the request for creating a DDR (as shown in the example in step 2.b), the names of the spec and parent PDRs are assigned automatically, by appending the name of the current managed attribute ("location"). However, if you explicitly include them in the request, you can provide your own names, as in the following alternative example. It shows how to add a managed attribute "location" with explicitly specified `loc_parent` and `loc_spec` values for these two attributes:

```
<ns:configTransaction>
  <ns:putDimensions>
    <ns1:record>
      <mdex-dimension_Key>location</mdex-dimension_Key>
      <mdex-dimension-value_Parent>loc_parent</mdex-dimension-value_Parent>
      <mdex-dimension-value_Spec>loc_spec</mdex-dimension-value_Spec>
      <mdex-dimension_EnableRefinements>true</mdex-dimension_EnableRefinements>
      <mdex-dimension_IsDimensionSearchHierarchical>true<
/mdex-dimension_IsDimensionSearchHierarchical>
      <mdex-dimension_IsRecordSearchHierarchical>true<
/mdex-dimension_IsRecordSearchHierarchical>
    </ns1:record>
  </ns:putDimensions>
</ns:configTransaction>
```

Note that `loc_parent` must be of type string, be single-assign and have no assignments from data records. The `loc_spec` attribute should be of type spring, unique, single-assign and have no assignments from data records.

This example is not used in the rest of this procedure.

Now that you have added a managed attribute "location", you can start adding managed attribute values to it, organized in a hierarchy.

3. Add the managed attribute value record, "USA":

```
<ns:configTransaction>
  <ns:putManagedAttributeValues>
    <ns1:mav>
      <name>USA</name>
      <spec>US</spec>
      <parent>/</parent>
      <managedAttribute>location</managedAttribute>
    </ns1:mav>
```

```
  </ns:putManagedAttributeValues>
</ns:configTransaction>
```

Notice that because this is going to be the top-most value in the hierarchy, it includes `"/"` as its parent managed attribute value. This sign, `"/"`, must be specified for the root managed attribute value. Additionally, the associated managed attribute for this value is "location". It must exist before you create a managed attribute value.

4. In a similar way, add the "Massachusetts" and "New York State" managed attribute values, under the parent value "USA", for the same managed attribute "location":

```
<ns:configTransaction>
  <ns:putManagedAttributeValues>
    <ns1:mav>
      <name>Massachusetts</name>
      <spec>MA</spec>
      <parent>US</parent>
      <managedAttribute>location</managedAttribute>
    </ns1:mav>
    <ns1:mav>
      <name>New York State</name>
      <spec>NY</spec>
      <parent>US</parent>
      <managedAttribute>location</managedAttribute>
    </ns1:mav>
  </ns:putManagedAttributeValues>
</ns:configTransaction>
```

5. Add the managed attribute values "Boston" and "New York City", under their respective parent managed attribute values "Massachusetts" and "New York State", for the same managed attribute "location":

```
<ns:configTransaction>
  <ns:putManagedAttributeValues>
    <ns1:mav>
      <name>Boston</name>
      <spec>bos</spec>
      <parent>MA</parent>
      <managedAttribute>location</managedAttribute>
      <synonym>BeanTown</synonym>
      <synonym>The Hub</synonym>
    </ns1:mav>
    <ns1:mav>
      <name>New York city</name>
      <spec>nyc</spec>
      <parent>NY</parent>
      <managedAttribute>location</managedAttribute>
      <synonym>NYC</synonym>
      <synonym>the Big Apple</synonym>
    </ns1:mav>
  </ns:putManagedAttributeValues>
</ns:configTransaction>
```

Alternatively, you could also add all managed attribute values (for country, states and cities) in a single request.

Notice that each of the managed attribute values has two synonyms. For information on synonyms, see *Summary of operations on page 168*.

Now that you have created these managed attribute values, you can list them with `listManagedAttributeValues` operation of the Configuration Web Service.

# Listing managed attribute values

You can list either all managed attribute values, or list only values added for a specific managed attribute with `listManagedAttributeValues` operation in the Configuration Web Service.

To list which managed attribute values exist in the system:

1. Issue a request with `listManagedAttributeValues`, using one of the two options:
   - To list all managed attribute values, regardless for which managed attribute they are added, use:

     ```
     <ns:configTransaction>
       <ns:listManagedAttributeValues>
       </ns:listManagedAttributeValues>
     </ns:configTransaction>
     ```

   - To list managed attribute values for a specific managed attribute, use:

     ```
     <ns:configTransaction>
       <ns:listManagedAttributeValues>
         <mdex-dimension_Key>managed_attribute_name</mdex-dimension_Key>
       </ns:listManagedAttributeValues>
     </ns:configTransaction>
     ```

     where `mdex-dimension_Key` is the name of the DDR for the specific managed attribute.

**Example**

For example, this request:

```
<ns:configTransaction>
  <ns:listManagedAttributeValues>
    <mdex-dimension_Key>location</mdex-dimension_Key>
  </ns:listManagedAttributeValues>
</ns:configTransaction>
```

Returns the following result, listing all managed attribute values for the managed attribute "location":

```
<config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3/0">
 <managedAttributeValues xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09">
  <mav>
   <name xmlns="">Boston</name><spec xmlns="">bos</spec><parent xmlns="">MA</parent>
   <managedAttribute xmlns="">location</managedAttribute>
  </mav>
  <mav>
   <name xmlns="">Massachusetts</name><spec xmlns="">MA</spec><parent xmlns="">US</parent>
   <managedAttribute xmlns="">location</managedAttribute>
  </mav>
  <mav>
   <name xmlns="">New York city</name><spec xmlns="">nyc</spec><parent xmlns="">NY</parent>
   <managedAttribute xmlns="">location</managedAttribute>
  </mav>
  <mav>
   <name xmlns="">USA</name><spec xmlns="">US</spec>
   <parent xmlns="">/</parent><managedAttribute xmlns="">location</managedAttribute>
  </mav>
  <mav>
   <name xmlns="">New York state</name><spec xmlns="">NY</spec>
   <parent xmlns="">US</parent><managedAttribute xmlns="">location</managedAttribute>
  </mav>
 </managedAttributeValues>
</config-types:results>
```

# Deleting managed attribute values

You can only delete all managed attribute values for a specific managed attribute, by deleting its PDR (this deletes the associated DDR and all assignments on it). You cannot delete a single managed attribute value. Since managed attribute values are represented in the index as records, to delete them, you use the Data Ingest Web Service operations for deleting records.

To delete a set of managed attribute values:

1. Use the `DeleteRecords` operation of the `IngestChanges` element in the Data Ingest Web Service request, to delete a PDR for the associated managed attribute.

   For example, assume a managed attribute named "location" (both its PDR key and DDR key are named "location"). The following request of the Data Ingest Web Service deletes the PDR and its associated DDR:

   ```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http:/
   /www.endeca.com/MDEX/ingest/3/0" xmlns:ns1="http://www.endeca.com/MDEX/XQuery/2009/09">
      <soapenv:Header/>
      <soapenv:Body>
         <ns:ingestChanges>
            <ns:Language>en</ns:Language>
            <ns:deleteRecords>
               <ns:recordSpecifier>"mdex-property_Key" = location</ns:recordSpecifier>
            </ns:deleteRecords>
         </ns:ingestChanges>
      </soapenv:Body>
   </soapenv:Envelope>
   ```

   As a result of this step, the PDR for the managed attribute, its associated DDR, and all the managed attribute values are deleted.

   Note that this request does not delete the parent PDR (in this example, let it be named "loc_parent"), and spec PDR (named "loc_spec").

2. (Optional). You can also delete the two PDRs created for the `mdex-dimension-value_Spec` and `mdex-dimension-value_Parent` properties of the DDR. These two properties, and their associated PDRs are created by the Endeca Server when you add a DDR. They are not deleted automatically when you delete a DDR though. If you are not using these non-data PDR records for any other assignments in your data domain, you can also delete them, when deleting managed attribute values. Since they are records, you also delete them with the Data Ingest Web Service, using the request similar to the one in this topic, and specifying:

   For the spec PDR named "loc_spec":

   ```
   ...
   <ns:recordSpecifier>"mdex-property_Key" = loc_spec</ns:recordSpecifier>
   ...
   ```

   For the parent PDR named "loc_parent":

   ```
   ...
   <ns:recordSpecifier>"mdex-property_Key" = loc_parent</ns:recordSpecifier>
   ...
   ```

   For more information on deleting records, see the *Oracle Endeca Server Data Loading Guide*.

# About static ranking

When you add managed attribute values with `PutManagedAttributeValues`, you can add (and later update), the static ranking of the particular value, within this managed attribute. Static ranking affects how Endeca Server displays the values when returning refinements to the end users.

You specify static ranking as `rank`, in this structure of the Configuration Web Service:

```
<ns:putManagedAttributeValues>
  <ns1:mav>
    <name>?</name>
    <spec>?</spec>
    <parent>?</parent>
    <managedAttribute>?</managedAttribute>
    <synonym>?</synonym>
    <rank>?</rank>
  </ns1:mav>
</ns:putManagedAttributeValues>
```

**Note:** You can only control ranking if you use the `PutManagedAttributeValues` of the Configuration Web Service. This interface operation lets you add ranks and also resolves ties and prevents duplicate ranks. However, if you use the Data Ingest Web Service for adding managed attribute values (as records, via `ingestChanges:addRecords` request, while you can still add ranks, this operation does not resolve ranking ties.

The following statements describe how the Endeca Server uses rank values:

- Managed attribute values within a specific managed attribute are ordered according to their ranks (if they are specified), in the ascending order. If you did not specify a rank for any of the values, Endeca Server considers their rank as zero (0), and ranks the values accordingly.
- Ranking occurs within all values that are added for the specific managed attribute. Ranking is independent of the managed attribute's hierarchy. For example, for a managed attribute "location", two managed attribute values, "MA", and "NY State" can be ranked 1, and 2, accordingly. Then, the child values of MA, "Boston" and "Springfield" can be ranked 3, and 4.
- You can specify a rank when initially adding or updating a managed attribute value, or by using `putManagedAttributeValues` to update an existing value. In the case of an update, a new rank is used and any affected values are reordered.
- If the rank of any managed attribute value (when using the `putManagedAttributeValues` request) is tied with the rank of any another value of the same managed attribute, the ranks of the pre-existing values change according to the following algorithm. For each value A that is being added, if A is assigned a rank (A) and there is already a value B of rank (B) that is equal to rank (A), then B is assigned a rank (B)+1 instead. If the new rank(B)+1 is tied with the rank for an existing value C, ties are broken again as if B is a new value. (That is, C's rank become rank©)+1). This process repeats until there are no ties. The ranks of all other values specified in the request remain as specified unless changed by a subsequent request.
- (For `putManagedAttributeValues` requests only). No two managed attributes specified in the same request may have the same rank (the Configuration Web Service request is rejected). The entire request fails if any of the values is specified incorrectly. (Note that if you use the Data Ingest Web Service request, then this checking does not occur.)

- Overall, you can order refinements (which comprise both standard and managed attribute values), either globally or per query.
  - For the global order, you specify either the `record-count`, or `lexical` values on the `system-navigation_Sorting` property in the associated PDR for the attribute (both standard and managed).
  - For the per-query order, you specify `OrderByRecordCount="false"` in the `RefinementConfig` of the specific Conversation Web Service request, for that query. The query-time control overrides the global order.

  Depending on which of these methods is used, Endeca Server uses the following logic for breaking ties with static ranking specified for managed attribute values:
  - If `record-count` is enabled, Endeca Server first uses `record-count`, then it breaks ties with static ranks on managed attribute values, and returns the results.
  - If `lexical` is enabled, Endeca Server uses static rank numbers first, and then orders refinements based on their lexical order. In other words, if any managed attribute values have rank numbers associated with them, Endeca Server takes into account these rank numbers when returning and sorting these refinements. Note that if no rank is specified, a rank of 0 is assumed. This means that in practice, lexically-ordered refinements may be returned first, followed by those that have a static rank on their managed attribute values. As a workaround, in such cases, you can add negative ranks, which will result in ranked managed attribute values sorted earlier than unranked values.

*Adding and updating ranks*

## Adding and updating ranks

To add or update a static rank for the managed attribute value, use `putManagedAttributeValues` operation. A static rank is optional, and is used by Endeca Server when ranking values when they are returned as refinements, to the end users.

For information on how static ranking behaves and how Endeca Server assigns ranks and break ties with existing ranks, see *About static ranking on page 176*.

To add or update a static rank:

1. Use a request similar to the following:

```
<ns:configTransaction>
  <ns:putManagedAttributeValues>
    <ns1:mav>
      <name>Boston</name>
      <spec>bos</spec>
      <parent>MA</parent>
      <managedAttribute>location</managedAttribute>
      <rank>2</rank>
    </ns1:mav>
  </ns:putManagedAttributeValues>
</ns:configTransaction>
```

   This request specifies the rank 2 for the managed attribute value "Boston". Endeca Server uses this rank value to return the managed attribute value Boston.

Chapter 15

# Working with Attributes and Refinements

This section discusses attributes and Guided Navigation. It also describes how to retrieve and display refinements in your front-end Web application powered by Endeca Server.



## About Guided Navigation

Guided Navigation is a key feature of Endeca Server.

Endeca Server is designed to facilitate an interactive query model that allows end users to discover information in their data without knowing exactly what they are looking for from the start.

It is designed around the idea of a "conversation" where the users make repeated queries, and with each one receive a set of matching data and also hints about how to further narrow down their results. This process is known as ***Guided Navigation***.

## About refinements

Refinements allow end users to explore their data records using Guided Navigation.

To utilize Guided Navigation, users first select a refinement and expand and collapse the available values. For a refinement that has a hierarchy, users can expand the refinement more than once.

To use refinements to narrow their results, end users select a specific refinement value. For example, to narrow results from all refrigerators to only white ones, users might choose the value "white" for the color refinement.

As they explore, users need to know what values are currently available for refinement. At any given step in the exploration process, Endeca Server returns the list of available values for each refinement, and displays to the users only those refinements (attribute values) for which records exist in the resulting record set.

In other words, after a user creates a query using record and value search, only valid remaining refinement values are provided to the user to further refine that query. This allows the user to reduce the number of matching records without creating an invalid query.

To conclude, ***refinements*** are values presented to the users during Guided Navigation. They contain attribute values for the records loaded into the data domain.

## How refinements relate to attributes

Once the source records are loaded into the data domain, Endeca Server uses attributes on the Endeca records for refinements. In other words, the attribute values for refinements are derived from attributes defined on records. Attributes, in turn, are typically generated from attributes on the source records. Attributes consist of key-value pairs.

In addition, managed attributes can have a multi-level hierarchy, can be enumerated, and can have specs. They can also be searched and displayed using their display names. These characteristics of managed attributes affect the ways in which you control refinements.

## Types of refinements in the user interface

In the user interface, refinements fall into two categories, depending on how they are displayed in the user interface:

- **Suggested refinements** are those that can be used for further navigation. When you select a specific refinement, you change (or refine), the resulting set of records. Suggested refinements are always returned by the Endeca Server, if they are available in the current navigation state.
- **Applied refinements** show your current location in the guided navigation process. These refinements are already selected. Applied refinements can be of two sub-types:
  - ***Explicitly-selected refinements*** are those that the end users have chosen during their navigation process, for example, by clicking them in the navigation menu.
  - ***Implicit refinements*** are those that were implicitly applied by the end user's explicit selections of other refinements. For example, in a data set containing both city and state attributes, selecting "Cambridge" implicitly selects "Massachusetts" (if all records tagged with "Cambridge" are also tagged with "Massachusetts"). In other words, a refinement is considered implicit if selecting it does not narrow the resulting record set.

  By default, Endeca Server does not return applied refinements, and only retrieves those refinements that are still available for navigation (suggested). However, in some instances, it is useful to request the full list of refinements, both suggested and applied (which includes implicits).

It is common to build separate user interfaces to display suggested refinements and applied refinements separately.

## Collapsing and expanding refinements

When Endeca Server computes refinements, it takes into account whether you want to expand or collapse the suggested refinements in the user interface.

The computation of available refinement values can be expensive, especially if there are many attributes defined on records in your data domain. In cases when the number of attributes is large, it is often not useful for a user interface to display values for every available attribute at once, because there would be too many options for users to consider.

To address this, Endeca Server supports the notion of "expanding" and "collapsing" suggested refinements displayed in the user interface:

- If a returned attribute is configured as collapsed, Endeca Server computes whether it is a valid refinement, but does not compute all available values.
- If a returned attribute is configured as expanded, in addition to determining whether it is a valid refinement, Endeca Server also computes and returns all available values.

# Working with refinements in Studio and other front-end applications

Refinements are handled slightly differently in Studio than in other front-end applications powered by Endeca Server.

Studio uses the **Available Refinements** component, which utilizes the Conversation Web Service, while other applications should rely directly on the Conversation Web Service:

- **Studio**. In Studio, each component affected by the **Available Refinements** component automatically uses suggested refinements and information received from refinements computation, such as the order of refinements or a number of refinements for a given attribute.
- **Other front-end applications**. For other front-end applications powered by Endeca Server, including custom-built applications, you can use the Conversation Web Service requests to work with refinements and build the user interface around them, utilizing the principles of Guided Navigation. For example, you can:
  - Request all available refinements from Endeca Server, so that you can display them as suggested refinements in the user interface.
  - Request a list of applied refinements from Endeca Server, so that the current navigation state is reflected in the user interface, showing the users their relative location as they navigate the data.
  - Configure the behavior of refinements, such as number of refinements to be displayed, or the order in which to display them, or whether to return records tagged with implicit refinements.

  In the Conversation Web Service, suggested refinements are included in the navigation menu configuration, and the current navigation state configuration lets you include applied refinements, both implicit and explicitly-selected.

  For information on how to use the Conversation Web Service requests for refinements, see *Working with attributes and refinements using the API on page 185*.

# Schema configuration for enabling refinements

The schema configuration for your records specifies whether attributes are enabled to be used as refinements.

If you want to use values from standard attributes as refinements, no configuration is needed. If a standard attribute is present on some records, the corresponding refinement values are automatically identified by the Endeca Server to let you create a new query or refine an existing query. Further, there are no Dgraph configuration flags necessary to enable the basic displaying of refinements.

For managed attributes, you can control in the schema whether they should become available as refinements. In the managed attribute's DDR (Dimension Description Record), the `mdex-`

`dimension_EnableRefinements` attribute is set to `true` by default. This setting is typically not changed, as it allows refinements to be displayed. If you set the configuration attribute to `false`, refinements will not be displayed, that is, the values from the managed attribute will be hidden.

Although this setting is typically not changed, as with any schema configuration, you can change the value of the `mdex-dimension_EnableRefinements` attribute using the Configuration Web Service (before the records are loaded) — this will suppress the display of refinements. For information, see *Configuration Web Service Interface on page 52*.

# Configuring the order of suggested refinements

Refinements can be displayed using either the default order for the attributes, or the order that you specify per query.

The `system-navigation_Sorting` attribute in the attribute's PDR controls the default order of the available refinement values. `system-navigation_Sorting` can be set to these values:

- `record-count` sorts refinement values in descending order, by the number of records available for each refinement value. This is the default.
- `lexical` sorts refinement values in alphabetical or numeric order. For example, if the end user in the front-end application chooses the values Red (15 records), Green (25 records), and Blue (5 records), then if the sorting is lexical, the values are displayed in this order: Blue (5 records), Green (25 records) Red (15 records).

You configure the default order for the values of suggested refinements by sending the configuration request to the Oracle Endeca Server with the Configuration Web Service, or by using Integrator ETL.

For information on how to change the value for the `system-navigation_Sorting` in the PDR, see *Configuration Web Service Interface on page 52*, or in the *Oracle Endeca Server API References*, see the section about the Configuration Web Service. For information on using Integrator ETL for the same purpose, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

You can also control the order of suggested refinements for each query. The query-time control overrides the order that is set globally for refinements in the PDR. For information, see *About query-time control of refinement ordering on page 204*.

# Configuring whether to display refinement counts

***Refinement counts*** represent the number of records (in the current navigation state) available beneath a given refinement value. These counts are computed dynamically at run-time by Endeca Server and can be displayed in the user interface.

By showing the user the number of records that will be returned for each refinement, refinement counts can enhance the front-end application's navigation controls by providing more context at each point in the user's Guided Navigation process.

By default, attributes are enabled to display refinement counts, and no further configuration is needed to display them. So long as there are attribute values returned for a given request, refinement value counts are also returned as an attribute attached to each attribute value.

You configure whether to show record counts for an attribute by changing the value in the `system-navigation_ShowRecordCounts` attribute in the PDR, using either the Configuration Web Service or Integrator ETL.

The valid settings for `system-navigation_ShowRecordCounts` are:

- `true` means that record counts are enabled and will display. This is the default.
- `false` means that record counts are disabled and will not be displayed.

For information on how to change the value for the `system-navigation_ShowRecordCounts` in the PDR, see *Configuration Web Service Interface on page 52* and the *Oracle Endeca Server API References*, the section about the Configuration Web Service. For information on using Integrator ETL for the same purpose, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

For information on how to retrieve the record counts for a refinement, see *How refinement counts are returned on page 204*.

You can also control the number of available refinements to display, in addition to the number of records under each refinement. For information, see *Increasing the number of refinements to be displayed on page 203*.

# Displaying refinements on multi-select attributes

This section describes refinements for attributes that can have multiple values. It discusses how to configure attributes for multi-select refinements and also how displaying such refinements affects the user interface.

*About multi-select attributes*

*Configuring attributes for multi-select refinement*

*Multi-select refinements and the user interface*

*Avoiding dead-end query results*

*Refinement counts for multi-or refinements*

## About multi-select attributes

If an attribute on your records can have more than one value, it is known as multi-select. For example, an attribute "Flavor" on wine records can have values "Apple" and "Apricot".

The schema for your records controls whether a standard attribute can have a single value, or multiple values. Attributes with multiple values can be of two types:

- **multi-select AND** attributes (also referred to as `multi-and`)
- **multi-select OR** attributes (also referred to as `multi-or`)

Respectively, refinements on such attributes are also known as multi-select refinements.

## Configuring attributes for multi-select refinement

You configure whether an attribute is multi-select by changing the value in the `system-navigation_Select` attribute of a PDR for a particular attribute defined on records in your data domain, using the Configuration Web Service or Integrator ETL.

The `system-navigation_Select` attribute of a PDR can have the following settings:

- `multi-and` configures the attribute as a multi-select AND attribute.

- `multi-or` configures the attribute as a multi-select OR attribute.
- `single` configures the attribute as a single-select attribute. This is the default.

You can perform this configuration using Configuration Web Service requests (before records are loaded).

The multi-select setting is only supported for non-hierarchical, or standard, attributes.

## Multi-select refinements and the user interface

If an attribute is configured as multi-and or multi-or, this affects the way Endeca Server calculates the refinements for such attributes, and therefore, has implications for the display of such refinements in the user interface.

The default Guided Navigation behavior of Endeca Server is to allow users to add only a single value from an attribute to the navigation state. This means that when users select a refinement from an attribute (by clicking it in the user interface, in the list of suggested refinements), that attribute is removed from the list of suggested refinements available for future refinement in the query results. For example, after selecting "Apple" from the Flavors attribute, the Flavors attribute is removed from the user interface's navigation controls. However, sometimes it is useful at navigation time to allow the user to select more than one value from an attribute. For example, the user interface can provide a user with the ability to show wines that have both "Apple" and "Apricot" values from the "Flavor" attribute.

To summarize, even though the fact that an attribute is tagged as multi-select is transparent to the front-end application's developer, the behavior of multi-select attributes may require changes in the user interface. Once an attribute is tagged as multi-select, the semantics of how Endeca Server interprets navigation queries and returns available refinements changes. After tagging an attribute as multi-select, Endeca Server allows multiple attribute values from the same attribute to be added to the navigation state. Endeca Server behaves differently for the two types of multi-select attributes:

- A refinement for a **multi-and** attribute. Endeca Server treats the list of attribute values selected from a `multi-and` attribute as a Boolean AND operation. That is, Endeca Server will return all records that satisfy the Boolean AND of all the attribute values selected from an attribute that is `multi-and`. For example, it will return all records that have been tagged with "Apple" AND "Apricot". If the user continues the refinement process by clicking one of the suggested refinements, Endeca Server will also continue to return refinements for a `multi-and` attribute. The list of available refinements will be the set of attribute values that have not been chosen, and are still valid refinements for the results.
- **multi-or** refinement. If the navigation state contains multiple values from a `multi-or` refinement, then all of the records in that state contain at least one of the selected values. A `multi-or` attribute is analogous to a `multi-and` attribute, except that a Boolean OR operation is performed instead (that is, all records that have been tagged with "Apple" OR "Apricot" are returned). Endeca Server will always return all attribute values for a `multi-or` attribute that have not already been selected – this means that the set of suggested refinements in the user interface does not correlate to the set of remaining records. Also note that as more `multi-or` attribute values are added to the navigation state, the set of record results gets larger instead of smaller, because adding more terms to an OR expands the set of results that satisfy the query.
- **single** configures the attribute as a single-select attribute. This means that only one value can be selected for an attribute at a time. This is the default.

## Avoiding dead-end query results

Be careful when rendering the selected attribute values of `multi-or` attributes. It is possible to create an interface that might result in dead ends when removing selected attribute values.

Consider this example: an attribute Alpha has been flagged as `multi-or`, and contains values 1 and 2. Attribute Beta contains value 3.

Assume the user's current query contains all three values. The user's current navigation state would represent the query:

```
"Return all records tagged with (1 or 2) and 3"
```

If the user then removes one of the values from Attribute Alpha, a dead end could be reached. For example, if the user removes value 1, the new query becomes:

```
"Return all records tagged with 2 and 3"
```

This could result in a dead end if no records are tagged with both value 2 and 3.

Due to this behavior, it is recommended that the user interface be designed so that the user must be forced to remove all values from a `multi-or` attribute when making changes to the list of selected attribute values.

## Refinement counts for multi-or refinements

Refinement counts on an attribute that is `multi-or` indicate how many records in the result set will be tagged with the refinement if you select it. When no selections are made yet, the refinement count equals the total number of records in the result set if that refinement were selected. However, for subsequent refinements, the refinement count may differ from the total results set.

Consider the following example that illustrates this use case. A `cuisine` refinement is configured as `multi-or`. In the data set, there are 2 records that have assignments only to a `Chinese` attribute, and 3 records that have assignments only to a `Japanese` attribute. There is also 1 record that has assignments on both of these attributes.

When the user requests `Chinese` or `Japanese` as refinements during navigation, the resulting record list includes all 6 records, out of which 2 have only `Chinese` attribute, 3 have only `Japanese` attribute, and 1 has both:

| **Records** | **Assignment on a `Chinese` attribute** | **Assignment on a `Japanese` attribute** |
|---|---|---|
| 1 | x | |
| 2 | x | |
| 3 | x | x |
| 4 | | x |
| 5 | | x |
| 6 | | x |

If the user first selects only `Chinese`, the navigation state shows that there is one remaining follow-on refinement (Japanese) with the refinement count of 4 records (3 with only `Japanese` assignment on a attribute and 1 that has both `Chinese` and `Japanese` attribute assignments on them). When the user navigates on that refinement, the resulting record list includes all 6 records. This illustrates that a record count for a `Japanese` refinement shows the number of records (4) tagged with that refinement, within the entire record set (6).

# Working with attributes and refinements using the API

This section provides examples of Conversation Web Service requests and responses that work with attributes and refinements.



## NavigationMenuConfig

`NavigationMenuConfig` is the element in the Conversation Web Service request where instructions are specified for how to return refinements and how those refinements should behave. This gives you a lot of control in which information about refinements you would like to retrieve, to build the Guided Navigation experience for the end users. For example, the request can specify whether to compute and return values for all refinements, whether to return a full path of the hierarchical refinements, what should be the maximum number of refinements returned, and whether to retrieve already applied refinements, such as those that were applied implicitly.

The `NavigationMenuConfig` has the following syntax:

```
<NavigationMenuConfig
      Id="?"
      ExposeAllRefinements="?"
      ReturnFullPath="false"
      MaximumRefinementCount="?"
      IncludeAllExplicitSelections="false"
      IncludeAllImplicitSelections="false">
   <StateName>?</StateName>
   <RefinementGroupConfig ...>
      ...
```

```
   </RefinementGroupConfig>
   <RefinementConfig .../>
</NavigationMenuConfig>
```

`NavigationMenuConfig` contains the following parameters:

| Attribute | Description |
|---|---|
| `Id` | Required. An identifier for this query configuration. |
| `ExposeAllRefinements` | Optional. Indicates whether to expose refinements (that is, whether to compute refinement values) for all attributes in the group (the default is `false`). Setting this to `true` is equivalent to sending a `RefinementConfig` with `Expose="true"` for each standard attribute and managed attribute. |
| `ReturnFullPath` | Optional. If set to `true`, each set of refinements will be accompanied with the path to the root for that set of refinements. The default is `false`. |
| `MaximumRefinementCount` | Optional. An integer that specifies a maximum limit on the number of refinements returned per standard or managed attribute. The default is 10. |
| `IncludeAllExplicitSelections` | Optional. Specifies whether to return all explicitly-selected refinements, in addition to returning suggested refinements, (which are always returned, if available). The default is `false`. |
| `IncludeAllImplicitSelections` | Optional. Specifies whether to return all implicitly-selected refinements, in addition to returning suggested refinements. The default is `false`. |
| `StateName` | Specifies a named state, using these rules:<br>• If the request has multiple named states, then the `StateName` element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the `StateName` element is used to reference that named state (as the state will be used in any event in the `NavigationMenuConfig`).<br>• If the request has an unnamed state, then the `StateName` element cannot be used. |

| Attribute | Description |
|---|---|
| `RefinementGroupConfig` | Returns a list of refinements from attributes included in attribute groups.<br>The options under this element let you enable the expand/collapse options in the UI for refinements in a group, compute values behind all refinements in a group, and control the behavior of individual attributes in a group.<br>This element contains the `RefinementConfig` element. To perform operations on individual attributes in a group, instead of performing operations on all of them (for example, to override settings for an individual attribute), you can use multiple `RefinementConfig` elements in a `RefinementGroupConfig`.<br>Note that you can also include `RefinementConfig` element into `NavigationMenuConfig` on its own, without enclosing it in `RefinementGroupConfig`. This is useful if you have attributes that are not included in groups. |
| `RefinementConfig` | Controls the behavior of an individual attribute in a `NavigationMenuConfig`. It specifies which attributes, out of all valid refinements returned with a navigation query, should return actual refinement values, and includes specifications about the order and number of refinement values.<br>You can omit `RefinementConfig` in your request if you do not need any of its optional items. |

## RefinementGroupConfig

The `RefinementGroupConfig` element returns a list of refinements from those attributes that are included in attribute groups.

Use the `RefinementGroupConfig` and `RefinementConfig` elements of the Conversation Web service request to expose refinement values for attributes. These elements allow you to perform operations on many attributes at once without enumerating all of them.

By default, attributes in groups and refinements are collapsed. If you would like to expose attributes that have refinements, either use `RefinementConfig` on each attribute whose refinements you want to expose, or use `ExposeAllPropertyRefinements` and expose refinements for all attributes at once.

The `RefinementGroupConfig` element contains the `RefinementConfig` element. The `RefinementGroupConfig` syntax is:

```
<RefinementGroupConfig Name="?" Expose="?" ExposeAllPropertyRefinements="?">
   <RefinementConfig .../>
</RefinementGroupConfig>
```

RefinementGroupConfig contains the following parameters:

| Attribute | Description |
|---|---|
| Name | Required. The name of the attribute group. |
| Expose | Required. A boolean that indicates whether to evaluate attributes in the group as potential refinements at all. Using the Expose attribute is important when you have groups of attributes and would like to be able to expand and collapse them in the user interface of the front-end application. For example, if you wanted country name refinements, then in order to obtain them, you use Expose in the initial request which enables the expand/collapse options in the UI. Next, you request these refinements in a separate Conversation Web Service request. |
| ExposeAllPropertyRefinements | Optional. Indicates whether to expose refinements (that is, whether to compute refinement values) for all attributes in the group. |
| RefinementConfig | Controls the behavior of an individual attribute in a NavigationMenuConfig. It specifies which attributes, out of all valid refinements returned with a navigation query, should return actual refinement values, and includes specifications about the order and number of refinement values. Also includes specifications on whether to return a full list of refinements (both suggested and applied, including those that are implicit). |

The RefinementConfig element is used in a NavigationMenuConfig element, as in this example. Notice two Expose attributes, used for two different levels of exposure — first, the attributes in the group are exposed and next, the refinements under those attributes are exposed. In other words, this request exposes the attributes in the group, and then the refinement values for the OrderQuantity attribute that is part of the group Sales-Transaction:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State/>
   <NavigationMenuConfig Id="NavigationMenu">
     <RefinementGroupConfig Name="Sales-Transaction" Expose="true">
       <RefinementConfig Name="OrderQuantity" Expose="true" MaximumCount="100" />
     </RefinementGroupConfig>
   </NavigationMenuConfig>
</Request>
```

To perform operations on individual attributes in a group, instead of performing operations on all of them (for example, to override settings for an individual attribute), you can use multiple RefinementConfig elements in a RefinementGroupConfig.

## RefinementConfig

The RefinementConfig element controls the behavior of an individual attribute in a NavigationMenuConfig.

It specifies which attributes, out of all valid refinements returned with a navigation query, should return actual refinement values, and includes specifications about the order and number of refinement values. Also includes

specifications on whether to return a full list of refinements (both suggested and applied, including those that are implicit). You can omit `RefinementConfig` in your request if you do not need any of its optional items.

**Note:** For hierarchical refinements that are derived from managed attributes, you can additionally retrieve their full path. For information, see *Retrieving the full path of hierarchical refinements on page 206*.

The `RefinementConfig` syntax is:

```
<RefinementConfig
   Name="?"
   Spec="?"
   Expose="false"
   OrderByRecordCount="false"
   MaximumCount="?"
   IncludeExplicitSelections="false"
   IncludeImplicitSelections="false"/>
```

For a given attribute, you can use only one `RefinementConfig` element.

The descriptions of the attributes are:

| Attribute | Description |
|---|---|
| `Name` | Required. The name of the attribute. |
| `Spec` | Optional. Used for walking down a hierarchy, without following the refinements as you navigate down the refinement tree. The `Spec` identifies the refinement parent for the query.<br><br>For example, in an empty state, a `NavMenu` for the initial navigation state shows managed attribute `ProductCategory` to be available. The user expands `ProductCategory`; this puts a `RefinementConfig Name="ProductCategory"` into the `RefinementConfig` and sends it in to the Endeca Server. The `NavMenu` shows `ProductCategory` refinements, including `Electronics`, which is expandable. The user expands `Electronics`; this puts a `RefinementConfig Name="ProductCategory" Spec="Electronics"` into the `RefinementConfig` and sends it to the Endeca Server.<br><br>Next, the response received from the Endeca Server contains `NavMenu` with the `ProductCategory` refinement starting at `Electronics`.<br><br>For a refinement with hierarchy (which is based on values from a managed attribute), refinement values will be returned for any child managed attribute values of this spec. For example, `Spec="/"` refers to a root managed value (such as for `WineType`), while `Spec="Merlot"` refers to a child managed value. |
| `Expose` | Optional. Specify `false` (the default) to compute whether this attribute is a refinement at all, or `true` to compute whether it is a refinement and also to retrieve and expose individual refinements (if any are present). |

| Attribute | Description |
|---|---|
| `OrderByRecordCount` | Optional. Specify `true` to order by record count based on the individual query (dynamic ordering) or `false` (the default) to use the default order from the Dgraph, specified by the `system-navigation_Sorting` attribute in the PDR. For details, see *Enabling the refinement order at query time on page 205*. |
| `MaximumCount` | Optional. An integer that specifies a maximum limit on the number of refinements returned per standard or managed attribute. If this setting is not specified, the number of refinements returned per attribute in the Conversation Web Service response is dictated by the value of the `MaximumRefinementCount` attribute in the `NavigationMenuConfig` element in the Conversation Web Service request. If that value is not specified, the default is 10. |
| `IncludeExplicitSelections` | Optional. Controls whether explicitly-selected refinements are returned (along with suggested refinements, which are always returned, if available), for this specific attribute. The default is `false`. This setting controls retrieval of explicitly-selected refinements per individual attribute — it overrides the global setting for retrieving explicitly-selected refinements in `NavigationMenuConfig`. |
| `IncludeImplicitSelections` | Optional. Controls whether implicitly-selected refinements are returned, for this specific attribute. The default is `false`. This setting controls retrieval of implicitly-selected refinements per individual attribute — it overrides the global setting for retrieving implicit refinements in `NavigationMenuConfig`. |

## Notes on RefinementConfig

**Note:** Keep in mind that the `RefinementConfig` element is an optional query parameter. However, attributes for which `RefinementConfig` is not included will not return refinements (unless `ExposeAllPropertyRefinements` is used in the group). In other words, by default, attributes in groups and refinements are collapsed. If you would like to expose attributes that have refinements, either use `RefinementConfig` on each attribute whose refinements you want to expose, or use `ExposeAllPropertyRefinements` and expose refinements for all attributes at once. The following examples illustrate these use cases.

The `Expose` attribute is also optional and defaults to `false`. `Expose="false"` helps improve performance.

For example, in a simple data set with three attributes in a user-defined group "Sales-Characteristics", the query:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <State/>
  <NavigationMenuConfig Id="NavigationMenu">
   <RefinementGroupConfig Name="Sales-Characteristics" Expose="false"/>
  </NavigationMenuConfig>
 </Request>
```

will return information in `HasRefineableProperties`, but will not return refinements themselves. This is faster for the Oracle Endeca Server to compute. (To retrieve all attributes in the group, `Expose` should be set to `true`).

However, this query for the `ProductType` managed attribute:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <State/>
  <NavigationMenuConfig Id="NavigationMenu">
     <RefinementGroupConfig Name="Sales-Characteristics" Expose="true">
       <RefinementConfig Name="ProductType" Expose="true" />
     </RefinementGroupConfig>
  </NavigationMenuConfig>
</Request>
```

will return all three managed attributes (since they are included in the Sales-Characteristics group), as well as the top-level refinement attribute values for the `ProductType` managed attribute. This is slightly more expensive for the Dgraph to compute, and returns the three root managed attribute values as well as the top-level managed attribute values for `ProductType`, but is necessary for selecting a valid refinement.

A more advanced query option returns all the top-level managed attribute value refinements for all attributes requested (instead of a single attribute). This option involves setting the `ExposeAllRefinements` attribute to `true`. If an application sets this attribute to `true`, the query:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <State/>
  <NavigationMenuConfig Id="NavigationMenu" ExposeAllRefinements="true">
    <RefinementGroupConfig Name="Sales-Characteristics" Expose="true"/>
  </NavigationMenuConfig>
</Request>
```

will return three managed attributes, as well as all valid top-level managed attribute values for each of these attributes.

This is the equivalent of the query:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State/>
   <NavigationMenuConfig Id="NavigationMenu">
      <RefinementGroupConfig Name="Sales-Characteristics" Expose="true">
       <RefinementConfig Name="UnitPrice" Expose="true" />
       <RefinementConfig Name="OrderQuantity" Expose="true" />
       <RefinementConfig Name="CustomerPONumber" Expose="true" />
      </RefinementGroupConfig>
   </NavigationMenuConfig>
</Request>
```

This is the most expensive type of query for the Oracle Endeca Server to compute, and returns three root managed attribute values as well as all the top-level managed attribute values, creating a larger network and page size strain. This method, however, is effective for creating custom navigation solutions that require all possible refinement values to be displayed at all times.

## PropertyListConfig

In a query, the `PropertyListConfig` type returns a list of all available attributes for the data domain.

It contains the `Property` element, which includes pertinent information about an attribute including its key, display name, and other options. The PDR (and DDR, if present) is included for those front-end clients of the Conversation Web Service that prefer to read descriptor records directly.

The `PropertyListConfig` syntax is:

```
<PropertyListConfig Id="?">
   <StateName>?</StateName>
</PropertyListConfig>
```

The meanings of the `PropertyListConfig` elements and attributes are as follows:

| Element/Attribute | Description |
|---|---|
| `Id` | Required. An arbitrary identifier for this `PropertyListConfig`. |
| `StateName` | Optional. Specifies an existing named state in the request. Note that specifying a state has no effect on the results (even in a request with multiple states). |

## SelectedRefinementFilter

The `SelectedRefinementFilter` type allows you to create a refinement navigation query from a specific refinement.

The `SelectedRefinementFilter` type (which is used in the `State`) has this syntax:

```
<SelectedRefinementFilter Name="?" Spec="?" Id="?">
   <AppliedFilterRule>
      <Source FilterId="?">
         <StateName>?</StateName>
      </Source>
      <TargetPropertyKey>?</TargetPropertyKey>
   </AppliedFilterRule>
</SelectedRefinementFilter>
```

The meanings of attributes for the `SelectedRefinementFilter` type are as follows:

| Attribute | Description |
|---|---|
| `Name` | Required. The name of the standard attribute or managed attribute. |
| `Spec` | Required. The format depends on the type of the `Name` attribute:<br>• For a managed attribute, `Spec` is the specifier of a managed attribute value. For example, `Spec="/"` refers to a root managed value (such as for the **WineType** managed attribute), while `Spec="Merlot"` refers to a child managed value of `WineType`.<br>• For a standard attribute, `Spec` is a specific value of an assignment. For example, if `Flavors` is a multi-assign standard attribute, then `Spec="Almond"` refers to a value of "Almond" for a record assignment from the `Flavors` attribute. |
| `Id` | Optional. An identifier for this filter configuration. The identifier is needed only when using filter rules. The identifier must be unique among other filter identifiers in the state. |
| `AppliedFilterRule` | Optional. Used for collections and filter rules. If present, indicates that the filter was implicitly generated and supplies relevant information. |

| Attribute | Description |
|---|---|
| `Source` | Optional. Information about the source of the implicit filter. If the implicit filter has been derived via a filter rule, the `Source` names the Id of the filter and the State where it came from. |
| `FilterId` | Optional. The identifier of an implicit filter. |
| `StateName` | Optional. Specifies the name of a named state from which the implicit filter came from. |
| `TargetPropertyKey` | Optional. The key that is derived from the filter rule. This is the target attribute that was used when the filter rule was applied. |

### AppliedFilterRule elements

The `AppliedFilterRule` elements are used to display information about the source of an applied filter. If a filter is implicitly created via a filter rule, the `StateName` and `FilterId` indicate its origin If the `TargetPropertyKey` element is present, it indicates the name of the target attribute used in the generated filter.

The use of the `AppliedFilterRule` elements is the same as in the `SelectionFilter`. For more information, see *Use of the AppliedFilterRule element on page 120*.

## Obtaining a list of available attributes

When configuring attributes for refinements, it is useful to first obtain a list of available attributes. You can do this, utilizing `PropertyListConfig` request of the Conversation Web Service, or `listProperties` and `RecordKind` requests from the Configuration Web Service.

The following soapUI sample request illustrates how to obtain a list of attributes with `PropertyListConfig` in the Conversation Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
         <State/>
         <PropertyListConfig Id="MyAttrs"/>
      </Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Note that this sample request uses an empty, unnamed state.

The following abbreviated example shows the characteristics of the ProdType managed attribute. Using this output, you can view the values of PDRs and DDRs for the attribute, indicating whether the attribute is searchable, and specifying other characteristics contained in the PDR and DDR for this managed attribute:

```
<cs:Property Key="ProdType" Type="mdex:string" Dimension="true" DisplayName="Product Type" Refinable
="true">
   <cs:PropertyRecord>
      <cs:attribute name="mdex-property_DisplayName" type="mdex:string">Wine Type</cs:attribute>
      <cs:attribute name="mdex-property_IsPropertyValueSearchable" type="mdex:boolean">true<
/cs:attribute>
      <cs:attribute name="mdex-property_IsSingleAssign" type="mdex:boolean">false</cs:attribute>
      <cs:attribute name="mdex-property_IsTextSearchable" type="mdex:boolean">true</cs:attribute>
```

```
      <cs:attribute name="mdex-property_IsUnique" type="mdex:boolean">false</cs:attribute>
      <cs:attribute name="mdex-property_Key" type="mdex:string">ProdType</cs:attribute>
      <cs:attribute name="mdex-property_Language" type="mdex:string">unknown</cs:attribute>
      <cs:attribute name="mdex-property_TextSearchAllowsWildcards" type="mdex:boolean">true<
/cs:attribute>
      <cs:attribute name="mdex-property_Type" type="mdex:string">mdex:string</cs:attribute>
      <cs:attribute name="system-navigation_Select" type="mdex:string">single</cs:attribute>
      <cs:attribute name="system-navigation_ShowRecordCounts" type="mdex:boolean">true</cs:attribute>
      <cs:attribute name="system-navigation_Sorting" type="mdex:string">record-count</cs:attribute>
   </cs:PropertyRecord>
   <cs:DimensionRecord>
      <cs:attribute name="mdex-dimension-value_Parent" type
="mdex:string">mdex-dimension_ProdType_Parent</cs:attribute>
      <cs:attribute name="mdex-dimension-value_Spec" type="mdex:string">mdex-dimension_ProdType_Spec<
/cs:attribute>
      <cs:attribute name="mdex-dimension_EnableRefinements" type="mdex:boolean">true</cs:attribute>
      <cs:attribute name="mdex-dimension_IsDimensionSearchHierarchical" type="mdex:boolean">true<
/cs:attribute>
      <cs:attribute name="mdex-dimension_IsRecordSearchHierarchical" type="mdex:boolean">true<
/cs:attribute>
      <cs:attribute name="mdex-dimension_Key" type="mdex:string">WineType</cs:attribute>
   </cs:DimensionRecord>
</cs:Property>
```

### Other list attribute methods

Besides the `PropertyListConfig` type, you can use two other methods to get a list of the attributes in a data domain.

The `listProperties` operation of the Configuration Web Service returns all the attributes in the data domain, as in this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:listProperties/>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

Another method is using the `RecordKind` filter with the `nondata` value. For usage information, see *Filtering data and non-data records on page 136*.

## Retrieving refinements with the API: high-level overview

Displaying attribute refinement values is the core concept behind Guided Navigation. This topic provides a high-level overview of the procedure for retrieving and then applying refinements.

If your refinements are derived from attributes that are configured in groups, then before retrieving them you need to expose groups in the user interface. If the group is collapsed, Endeca Server does not compute refinements for the attributes within the group. If the group is exposed, Endeca Server computes refinements, but it may or may not expose them, depending on your request. If you issue a request that asks to expose already computed refinements, they are computed.

To display refinements in the front-end application so that they can be navigated upon, create two related Conversation Web Service requests:

1. In the first request, you accomplish two goals:
    1. Identify which attributes have valid refinements. See *Step 1: Obtaining and exposing attributes that have refinements on page 195*.
    2. Retrieve (or expose) the list of suggested refinements from the Endeca Server, using `Expose="true"` in `RefinementGroupConfig` (for groups of attributes), or `RefinementConfig` (for individual attributes). See *Step 2: Applying refinements by creating a new query on page 197*.
2. In a subsequent request, select and apply one of the retrieved selected refinements, in response to a user gesture in your user interface. Use the `SelectedRefinementFilter` type to select the refinement.

## Step 1: Obtaining and exposing attributes that have refinements

The first step in displaying refinements is to retrieve those attributes that potentially have refinements.

You can retrieve refinements in two ways, depending on whether their attributes are included in groups:

- If you are using attribute groups, you retrieve refinements on groups with `RefinementGroupConfig`. For its format, see *RefinementGroupConfig on page 187*.
- If you are not using attribute groups, you retrieve individual refinements with `RefinementConfig`. For its format, see *RefinementConfig on page 188*.

  As an alternative to using `RefinementConfig`, to retrieve refinements that are not explicitly included in any user-configured attribute groups, you can request a group `system-navigation_InternalGroup`. This group exists in Endeca Server and includes all refinements that are not members of any other groups.

In the request, either you are using `RefinementGroupConfig`, or `RefinementConfig`, you include them in a `NavigationMenuConfig`. For its format, see *NavigationMenuConfig on page 185*.

Refinements are returned in a `NavigationMenu` content element. If your attributes belong to groups, this element contains a `NavigationMenuItemGroup` element with `NavigationMenuItem` elements for each managed attribute with refinements.

### Retrieving suggested refinements for attribute groups

Consider this request in which the `WineType` refinement is requested and exposed:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
 <State/>
 <NavigationMenuConfig Id="NavigationMenu" IncludeAllExplicitSelections="false"
IncludeAllImplicitSelections="false">
   <RefinementGroupConfig Name="WineCharacteristics" Expose="true">
     <RefinementConfig Name="WineType" Expose="true"/>
   </RefinementGroupConfig>
  </NavigationMenuConfig>
</Request>
```

Notice that `IncludeAllExplicitSelections` and `IncludeAllImplicitSelections` are set to `false`. This is the default — the request asks to retrieve only those refinements that are still available for navigation (known as suggested refinements), and does not return refinements that have already been applied (which include explicitly-selected and implicit). For information on how to retrieve a full list of refinements, see *Retrieving the full list of refinements (applied and suggested) on page 197*.

The request returns the following query results. Notice that the query results show the `WineType` refinement and the refinement values on it — `Red`, `White`, and `Sparkling`.

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <State xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <cs:NavigationMenu" Id="NavigationMenu">
    <cs:NavigationMenuItemGroup Name="WineCharacteristics" HasRefinablePRoperties="true">
      <cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="None" HasMore
="false">
        <cs:ExposureControl Exposed="true"/>
        <cs:Refinement Name="WineType" Spec="Red" Label="Red" Count="31021"/>
        <cs:Refinement Name="WineType" Spec="White" Label="White" Count="23031"/>
        <cs:Refinement Name="WineType" Spec="Sparkling" Label="Sparkling" Count="3020"/>
        <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
        <cs:FullPath>
          <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
        </cs:FullPath>
      </cs:NavigationMenuItem>
    </cs:NavigationMenuItemGroup>
  </cs:NavigationMenu>
</cs:Results>
```

In this example, the `NavigationMenuItem` element is used for the managed attribute included in a group:

```
<cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="None" HasMore="false">
   <cs:ExposureControl Exposed="true"/>
```

Notice the `ExposureControl` type:

```
<cs:ExposureControl Exposed="true"/>
```

The `<cs:ExposureControl Exposed=""true">` statement indicates the current exposure status of a top-level refinement included in `NavigationMenuItem`.

Further examining this example, each refinement in this group is returned in a `Refinement` element, as shown in this example for the Red managed attribute value:

```
<cs:Refinement Name="WineType" Spec="Red" Label="Red" Count="31021"/>
```

The `Count` element indicates that 31,021 records would be in the result set if you were to refine on the Red refinement.

## Retrieving suggested refinements for attributes not included in groups

Consider the following request for a `Region` refinement:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <State/>
  <NavigationMenuConfig Id="NavigationMenu" IncludeAllExplicitSelections="false"
IncludeAllImplicitSelections="false">
    <RefinementConfig Name="Region" Expose="true" IncludeExplicitSelections="false"
  IncludeImplicitSelections="false"/>
  </NavigationMenuConfig>
</Request>
```

This request will return individual suggested refinements from a record set, listing all records for which values exist in the `Region` attribute. In addition to suggested refinements, you can also optionally return applied refinements for this attribute. For information, see *Retrieving applied refinements per attribute on page 200*.

**Note:** If you have precedence rules configured, they will suppress attributes that have valid refinements until a trigger for the precedence rule is met. For information on precedence rules, see *Precedence Rules on page 224*.

### Step 2: Applying refinements by creating a new query

Once refinement values have been retrieved/exposed, these values typically are used to create additional refinement navigation queries.

Based on the result in Step 1, the user has requested a list of red wines, which are defined by the `Red` managed attribute value. A follow-on request uses the `SelectedRefinementFilter` type to retrieve the records:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <Name>RefState</Name>
      <SelectedRefinementFilter Name="WineType" Spec="Red" Id="MyRef">
      </SelectedRefinementFilter>
   </State>
   <RecordListConfig Id="RecList" MaxPages="10">
      <StateName>RefState</StateName>
      <Column>WineType</Column>
      <Column>Wine</Column>
      <RecordsPerPage>5</RecordsPerPage>
   </RecordListConfig>
</Request>
```

The resulting `RecordList` entries are then displayed to the user by the front-end code.

## Retrieving the full list of refinements (applied and suggested)

By default, once end users make selections from a list of suggested refinements, Endeca Server narrows the result set. It does not return those refinements that have already been selected and returns remaining available, or suggested, refinements. However, in some cases, it may be useful to ask Endeca Server to return both suggested and already-applied refinements, (including both types of applied refinements: explicitly-selected and implicit).

Retrieving the full list of refinements — both applied and suggested — is useful only in specific implementations, and requires a special treatment in the user interface of the front-end application powered by the Endeca Server, to distinguish which of the refinements have been applied.

**Note:** When applied refinements are returned together with suggested refinements, a decision needs to be made in the user interface for how to handle them, to indicate to the end users which refinements have already been selected and which are still available for navigation. For example, one possibility is to display applied refinements along with suggested refinements, but make applied refinements un-selectable (with some indication that they are already selected implicitly).

The Conversation Web Service has controls that allow you to decide whether to retrieve the full list of refinements (or any combination of suggested, implicitly- and explicitly-selected refinements), globally for any attribute with a `RefinementConfig` in the containing `NavigationMenuConfig`, or for individually-specified attributes. Specifying these settings per attribute overrides the global settings. Further, these settings only apply when the `RefinementConfig` contains `Expose=true`.

For information, see:

- *Retrieving applied refinements for all attributes on page 198*
- *Retrieving applied refinements per attribute on page 200*

## Retrieving applied refinements for all attributes

By default, Endeca Server does not return applied (explicitly-selected or implicit) refinements once the end-user makes a query and selects some refinements as part of the Guided Navigation experience. In other words, by default, all refinements that are returned are guaranteed to narrow the result set in the future, if end users select any of them. However, you can issue a Conversation Web Service request for the navigation menu that should retrieve not only suggested refinements, but also applied (explicitly-selected and implicit) refinements, for all attributes used as refinements.

The following two settings in the `NavigationMenuConfig` element of the Conversation Web Service control whether to return explicitly-selected and implicit refinements for all attributes:

- `IncludeAllExplicitSelections` specifies whether Endeca Server should retrieve explicitly-selected refinements. The default is `false`.
- `IncludeAllImplicitSelections` specifies whether Endeca Server should retrieve implicit refinements. The default is `false`.

You can use any combination of `true` and `false` values.

Once the refinements are retrieved, `ExposeAllRefinements="true"` is the setting that controls whether they are also returned in the web service response.

A `NavigationMenuConfig` has the following structure:

```
<ns:NavigationMenuConfig Id="?"
ExposeAllRefinements="?"
ReturnFullPath="false"
MaximumRefinementCount="?"
IncludeAllExplicitSelections="false"
IncludeAllImplicitSelections="false">
  </ns:StateName>
   <ns:RefinementConfig Name="?"
    Spec="?" Expose="false" OrderByRecordCount="false" MaximumCount="?"
    IncludeExplicitSelections="false" IncludeImplicitSelections="false"/>
</ns:NavigationMenuConfig>
```

### Example: retrieving a full list of applied refinements for all attributes

In these examples, the namespaces are omitted and they are:

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"
xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types"
```

In the following example request, a query is made in a particular navigation state (assuming that the users have navigated to this state). It includes a refinement filter inside the state, and thus narrows the results to a specific Winery, named "A.R.Lenoble". The request retrieves the list of suggested refinements that are still available, and also retrieves, for all refinements, those refinements that have been already applied (explicitly-selected and implicit):

```
<ns:Request>
 <ns:State>
  <ns:Name>ref_state</ns:Name>
  <ns:SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"/>
 </ns:State>
 <ns:NavigationMenuConfig Id="NavigationMenu" ExposeAllRefinements="true"
 ReturnFullPath="true"
 IncludeAllExplicitSelections="true"
 IncludeAllImplicitSelections="true">
   <ns:StateName>ref_state</ns:StateName>
```

```
   <ns:RefinementConfig Name="WineType" Expose="true"/>
   <ns:RefinementConfig Name="Region" Expose="true"/>
   <ns:RefinementConfig Name="Winery" Expose="true"/>
  </ns:NavigationMenuConfig>
</ns:Request>
```

This request specifies `true` for both types of applied refinements—implicit and explicitly-selected—at the navigation level. This means, that if the current navigation state for the Winery ""A.R.Lenoble" includes any such refinement, the response will include retrieve it, along with the list of suggested refinements. Note that the full list of applied refinements is returned in the request only if `ExposeAllRefinements="true"`. In addition, this request also asks to expose all refinement values for three attributes: "WineType", "Region", and "Winery".

The response to this request is (namespaces are omitted):

```
<cs:Result>
 <State>
   <Name>ref_state</Name>
   <SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"/>
 </State>
 <cs:NavigationMenu Id="NavigationMenu">
    <cs:NavigationMenuItem Name="Region" DisplayName="Region" MultiSelect="None"
    HasMore="false">
      <cs:ExposureControl Exposed="true"/>
      <cs:RootDimensionValue DimensionName="Region" Spec="/"/>
      <cs:ImplicitRefinement Name="Region" Spec="Champagne" Label="Champagne"/>
    </cs:NavigationMenuItem>
    <cs:NavigationMenuItem Name="Winery" DisplayName="Winery" MultiSelect="None"
    HasMore="false">
      <cs:ExposureControl Exposed="true"/>
      <cs:RootDimensionValue DimensionName="Winery" Spec="/"/>
      <cs:SelectedRefinement Name="Winery" Spec="A.R. Lenoble" Label="A.R. Lenoble" Count="3"/>
    </cs:NavigationMenuItem>
    <cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="None"
    HasMore="false">
      <cs:ExposureControl Exposed="true"/>
      <cs:Refinement Name="WineType" Spec="Brut Rose" Label="Brut Rose" Count="1"/>
      <cs:Refinement Name="WineType" Spec="Brut Blanc de Blancs" Label="Brut Blanc de Blancs" Count
="1"/>
      <cs:Refinement Name="WineType" Spec="Brut" Label="Brut" Count="1"/>
      <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
      <cs:FullPath>
        <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
        <cs:DimensionValue DimensionName="WineType" Spec="Sparkling">Sparkling</cs:DimensionValue>
      </cs:FullPath>
     <cs:ImplicitRefinement Name="WineType" Spec="Sparkling" Label="Sparkling"/>
    </cs:NavigationMenuItem>
 </cs:NavigationMenu>
</cs:Results>
```

In this response, the `State` is returned first, reflecting the selected filter for the winery "L.A.Lenoble". It is followed by a `NavigationMenu`, which includes three `NavigationMenuItem` elements, for each of the attributes—"Region", "Winery", and "WineType".

For the attribute "Region", the response includes the implicitly-selected refinement value "Champagne":

```
<cs:NavigationMenuItem Name="Region" DisplayName="Region" MultiSelect="None"
    HasMore="false">
    <cs:ExposureControl Exposed="true"/>
    <cs:RootDimensionValue DimensionName="Region" Spec="/"/>
    <cs:ImplicitRefinement Name="Region" Spec="Champagne" Label="Champagne"/>
  </cs:NavigationMenuItem>
```

For the attribute "WineType", the response includes another implicit refinement, "Sparkling":

```
<cs:NavigationMenuItem Name="WineType" ...>
```

```
    ...
    <cs:ImplicitRefinement Name="WineType" Spec="Sparkling" Label="Sparkling"/>
  </cs:NavigationMenuItem>
```

This is because, in this data set, there are only three wines from the "A.R. Lenoble" winery which is located in the Champagne region and produces only Champagnes. So, by specifying winery as "A.R. Lenoble" in the state's filter: `SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"`, users also implicitly select two refinements: Region with value "Champagne" and WineType with value "Sparkling".

Notice also that for the attribute "Wine", the refinement "A.R. Lenoble" is returned as a `SelectedRefinement` for Winery, since it is selected in the query state of the request:

```
<cs:NavigationMenuItem Name="Winery" ...>
    ...
    <cs:SelectedRefinement Name="Winery" Spec="A.R. Lenoble" Label="A.R. Lenoble" Count="3"/>
  </cs:NavigationMenuItem>
```

The user interface can use this information to display these already-applied refinements as un-selectable (because they have already been selected). It can also optionally indicate to the users which of these refinements have been selected implicitly.

Additionally, this response includes a list of suggested refinements that are still available in this navigation state. These are present only for the refinement "WineType":

```
<cs:NavigationMenuItem Name="WineType" ...>
    ...
    <cs:Refinement Name="WineType" Spec="Brut Rose" Label="Brut Rose" Count="1"/>
    <cs:Refinement Name="WineType" Spec="Brut Blanc de Blancs" Label="Brut Blanc de Blancs" Count="1"
/>
    <cs:Refinement Name="WineType" Spec="Brut" Label="Brut" Count="1"/>
```

Finally, since all three requested refinements in the request are on managed attributes, (which have hierarchy), additional information about hierarchy is returned for each refinement, inside `RootDimensionValue`, and `FullPath` elements.

To summarize, this method lets you control, for all your attributes, whether applied refinements are retrieved. In addition to this global control, you can specify per each attribute, whether you would like to retrieve explicitly-selected refinements, implicit refinements, or both of them. Retrieving a full list of applied refinements per each attribute overrides the retrieving them for all attributes. For information see *Retrieving applied refinements per attribute on page 200*.

## Retrieving applied refinements per attribute

You can issue a Conversation Web Service request that retrieves, for a refinement of your choice, not only suggested refinements, but also applied refinements (both explicitly-selected and implicit).

By default, Endeca Server does not return explicitly-selected or implicit refinements once end-users make a query and select some refinements as part of the Guided Navigation experience. In other words, by default, all refinements that are returned are guaranteed to narrow the result set in the future, if end users select any of them. However, you can specify in the Conversation Web Service requests to retrieve the full set of refinements (suggested and applied). You can do so globally for all refinements, or for a specific refinement.

The following two attributes in the `RefinementConfig` element, (this is a sub-element of the `NavigationMenuConfig`), control whether to retrieve explicitly-selected and implicit refinements, per each specified attribute:

- `IncludeExplicitSelections` specifies whether Endeca Server should retrieve explicitly-selected refinements, for this attribute. The default is `false`.

- `IncludeImplicitSelections` specifies whether Endeca Server should retrieve implicit refinements, for this attribute. The default is `false`.

You can use any combination of `true` and `false` values.

Once the refinements are retrieved, `Expose="true"` is the setting that controls whether they are also returned in the web service response.

The settings made per attribute (in the `RefinementConfig` for each attribute) override the analogous settings made for all attributes (in the `NavigationMenuConfig`).

**Example: retrieving a full list of applied refinements for a specific attribute**

In these examples, the namespaces are omitted and they are:
`xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`
`xmlns:ns="http://www.endeca.com/MDEX/conversation/3/0"`
`xmlns:typ="http://www.endeca.com/MDEX/eql_parser/types"`

The following example of a request assumes that a query is made in a particular navigation state. In this state, end users have already made a selection for the winery "L.A.Lenoble". This request performs two tasks — it retrieves the next list of suggested refinements still available to users, and also, for each of the three specified attributes, asks to retrieve those refinements that have been already applied (explicitly-selected and implicit):

```
<ns:Request>
 <ns:State><ns:Name>ref_state</ns:Name>
  <ns:SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"/>
 </ns:State>
 <ns:NavigationMenuConfig Id="NavigationMenu" ExposeAllRefinements="true" ReturnFullPath="true"
  IncludeAllExplicitSelections="false" IncludeAllImplicitSelections="false">
 <ns:StateName>ref_state</ns:StateName>
  <ns:RefinementConfig Name="WineType" Expose="true" IncludeExplicitSelections
="true" IncludeImplicitSelections="true"/>
  <ns:RefinementConfig Name="Region" Expose="true" IncludeExplicitSelections
="true" IncludeImplicitSelections="true"/>
  <ns:RefinementConfig Name="Winery" Expose="true" IncludeExplicitSelections
="true" IncludeImplicitSelections="true"/>
 </ns:NavigationMenuConfig>
</ns:Request>
```

In this request:

- In `NavigationMenuConfig`, the global settings for returning explicitly-selected and implicit refinements are set to `"false"`:

  ```
  IncludeAllExplicitSelections="false" IncludeAllImplicitSelections="false"
  ```

- In each `RefinementConfig`, the request specifies `true` for both implicit and explicitly-selected refinements for these attributes: "WineType", "Region" and "Winery". This means, that if the navigation state for this refinement includes any such refinements, the response will include them:

  ```
  <ns:RefinementConfig Name="Region" Expose="true"
  IncludeExplicitSelections="true"
  IncludeImplicitSelections="true"/>
  ```

  The settings on `RefinementConfig` override those on `NavigationMenuConfig`, which means that if applied refinements are present for these attributes, they will be retrieved. Note also the setting `Expose="true"`. This setting controls whether the retrieved refinements are actually returned to you in the web service response.

Here is an abbreviated response to this request (with namespaces omitted):

```
<cs:Results>
```

```
 <Name>ref_state</Name>
 <SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"/>
 </State>
 <cs:NavigationMenu Id="NavigationMenu">
  <cs:NavigationMenuItem Name="Region" DisplayName="Region" MultiSelect="None" HasMore="false">
   <cs:ExposureControl Exposed="true"/>
   <cs:RootDimensionValue DimensionName="Region" Spec="/"/>
   <cs:ImplicitRefinement Name="Region" Spec="Champagne" Label="Champagne"/>
  </cs:NavigationMenuItem>
  <cs:NavigationMenuItem Name="Winery" DisplayName="Winery" MultiSelect="None" HasMore="false">
   <cs:ExposureControl Exposed="true"/>
   <cs:RootDimensionValue DimensionName="Winery" Spec="/"/>
   <cs:SelectedRefinement Name="Winery" Spec="A.R. Lenoble" Label="A.R. Lenoble" Count="3"/>
  </cs:NavigationMenuItem>
  <cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="None" HasMore="false">
   <cs:ExposureControl Exposed="true"/>
   <cs:Refinement Name="WineType" Spec="Brut Rose" Label="Brut Rose" Count="1"/>
   <cs:Refinement Name="WineType" Spec="Brut Blanc de Blancs" Label="Brut Blanc de Blancs" Count="1"
/>
   <cs:Refinement Name="WineType" Spec="Brut" Label="Brut" Count="1"/>
   <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
   <cs:FullPath>
    <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
    <cs:DimensionValue DimensionName="WineType" Spec="Sparkling">Sparkling</cs:DimensionValue>
   </cs:FullPath>
   <cs:ImplicitRefinement Name="WineType" Spec="Sparkling" Label="Sparkling"/>
  </cs:NavigationMenuItem>
 </cs:NavigationMenu>
</cs:Results>
```

Let's review this response.

For the attribute "Region", no suggested refinements are available, but there is one implicit refinement "Champagne":

```
<cs:NavigationMenuItem Name="Region" ... >
 ...
 <cs:RootDimensionValue DimensionName="Region" Spec="/"/>
 <cs:ImplicitRefinement Name="Region" Spec="Champagne" Label="Champagne"/>
</cs:NavigationMenuItem>
```

This is because, in this data set, there are only three wines from the "A.R. Lenoble" winery which is located in the Champagne region and produces only Champagnes. So, by specifying winery as "L.A.Lenoble" in the state's filter: `SelectedRefinementFilter Name="Winery" Spec="A.R. Lenoble"`, users also implicitly select two refinements: Region with value "Champagne", and WineType with value "Sparkling".

For the attribute "WineType", a list of suggested refinements is returned, followed by one implicit refinement "Sparkling":

```
<cs:NavigationMenuItem Name="WineType" ... >
...
 <cs:Refinement Name="WineType" Spec="Brut Rose" Label="Brut Rose" Count="1"/>
 <cs:Refinement Name="WineType" Spec="Brut Blanc de Blancs" Label="Brut Blanc de Blancs" Count="1"/>
 <cs:Refinement Name="WineType" Spec="Brut" Label="Brut" Count="1"/>
 ...
 <cs:ImplicitRefinement Name="WineType" Spec="Sparkling" Label="Sparkling"/>
</cs:NavigationMenuItem>
```

Finally, for the attribute "Winery", while there are no remaining suggested refinements, there is one selected refinement, "A.R. Lenoble":

```
<cs:NavigationMenuItem Name="Winery" ... >
...
 <cs:SelectedRefinement Name="Winery" Spec="A.R. Lenoble" Label="A.R. Lenoble" Count="3"/>
</cs:NavigationMenuItem>
```

To conclude, the user interface can use this information to display these refinements as un-selectable (because they have already been selected). It can also optionally indicate to the users that some of these refinements have been selected implicitly.

For each of these attributes, the request also includes information in the `RootDimensionValue`, and `FullPath` elements, because these are managed attributes with hierarchy.

## Increasing the number of refinements to be displayed

The number of refinements that are displayed defaults to 10. If this number is not sufficient, you can increase it using `NavigationMenuConfig` in the Conversation Web Service request.

Generally, when the request from the Conversation Web Service asks for attributes to return in response to a query, it asks for all of them that were requested with a `RefinementGroupConfig` element.

To provide a meaningful navigation experience, Endeca Server returns only those attributes that actually have refinements on them and that are not filtered by precedence rules. In other words, the attributes are returned based on the navigation state.

For the request to return refinements if they are present in the data set, the `Expose` attribute should be set to `true` in the `RefinementConfig`. Its default value is `false`.

The Conversation Web Service uses the following logic to identify the number of refinements to be displayed:

- The number of refinements that are displayed defaults to 10. If this number is not sufficient, you can increase it using `NavigationMenuConfig` in the Conversation Web Service request. You can override it in a global setting or per each refinement value, which are both optional and are not specified by default:
    - The global configuration setting controls this number for all attributes in a particular navigation menu. You specify it in the `MaximumRefinementCount` attribute of the `RefinementConfig` element. The setting is per content element, not per query.
    - Further, the setting per each refinement value controls the number returned for each attribute. You specify it in the attribute `MaximumCount` in the `RefinementConfig` element.

For example, in this configuration for the navigation menu, `MaximumRefinementCount` is set to 15. In addition, for the `WineType` refinement value, `MaximumCount` is set to 40. `MaximumCount` is not set in each of the other refinement values.

```
<NavigationMenuConfig Id="NavigationMenu" MaximumRefinementCount="15">
   <RefinementGroupConfig Name="WineCharacteristics" Expose="true">
     <RefinementConfig Name="WineType" Spec="/" MaximumCount="40"/>
     <RefinementConfig Name="Year"/>
     <RefinementConfig Name="Score"/>
   </RefinementGroupConfig>
</NavigationMenuConfig>
```

This request returns up to 40 refinement values for `WineType`. It returns up to 15 refinement values for each of the other two refinement values (`Year` and `Score`).

The attribute `HasMore` (with possible Boolean values `true` or `false`) in the response specifies whether the total refinement count exceeds the value returned with the `MaximumRefinementCount`.

The following example shows a response where the `HasMore` attribute is set to `true` in the `NavigationMenuItem` type of the Conversation Web Service response:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <State xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <cs:NavigationMenu" Id="NavigationMenu">
```

```
    <cs:NavigationMenuItemGroup Name="WineCharacteristics" HasRefinablePRoperties="true">
      <cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="Or" HasMore="true">
         <cs:ExposureControl Exposed="true"/>
         <cs:Refinement Name="WineType" Spec="Red" Label="Red" Count="31021"/>
         <cs:Refinement Name="WineType" Spec="White" Label="White" Count="23031"/>
         <cs:Refinement Name="WineType" Spec="Sparkling" Label="Sparkling" Count="3020"/>
         <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
         <cs:FullPath>
           <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
         </cs:FullPath>
      </cs:NavigationMenuItem>
    </cs:NavigationMenuItemGroup>
  </cs:NavigationMenu>
</cs:Results>
```

## How refinement counts are returned

The application user interface can display the number of records returned for each refinement. These record counts are returned in a `Count` attribute.

Each refinement is returned in a `Refinement` element, as shown in this example:

```
<cs:Refinement Name="WineType" Spec="Red" Label="Red" Count="31021"/>
```

In the example, a record count of 31021 is returned for the Red managed attribute value.

## Retrieving the order of refinements

A core capability of Endeca Server is the ability to dynamically order and present the most popular refinement values to the user.

There are two ways in which you can configure the display order of refinements in the Conversation Web Service:

- By specifying the value for `system-navigation_Sorting` in the PDR, for a standard or managed attribute.
- By using query-time control of the display order specified in the `OrderByRecordCount` attribute in the `RefinementConfig` element of the Conversation Web Service request. Note that by using this method, you can override the `system-navigation_Sorting` settings for a given attribute. For detailed information, see *About query-time control of refinement ordering on page 204* and *Enabling the refinement order at query time on page 205*.

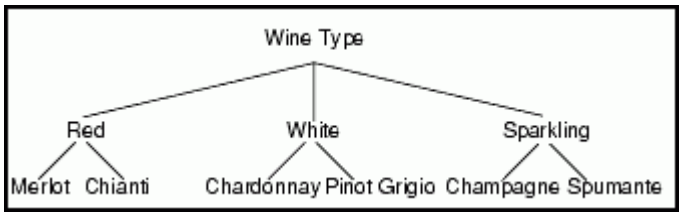
### About query-time control of refinement ordering

The Oracle Endeca Server allows you to switch refinement ordering on and off on a per-query basis.

A use case for this refinement ordering would be an application that renders refinements as a tag cloud. Such an application may adjust the size of the tag cloud at query time, depending on user preferences or the page from which the query originates.

You set the refinement ordering at the refinement value level that you want to control. For managed attributes, ordering is applied to that managed attribute value and all its children. For example, assume that you have a

managed attribute named WineType that has three child attribute values (named Red, White, and Sparkling), which in turn have two child attribute values each. The attribute's hierarchy would look like this:




You would set the ordering depending on which level of the hierarchy you want to order and present, for example:

- If you set the ordering on the root attribute value (which has the same name and ID as the managed attribute itself), the refinements in the Red, White, and Sparkling attribute values will be returned.
- If there are multiple child attribute values, you can set an order on only one sibling. In this case, the refinements from the other siblings will not be exposed. For example, if you set an order on the Red attribute value, only the refinements of the Merlot and Chianti attribute values will be returned. The refinements from the White and Sparkling attribute values will be not be shown, even if you explicitly set orders for them.

The settings of the per query ordering of refinements are not persistent. That is, each query must have its own configuration, because it is not carried over from the previous query.

## Enabling the refinement order at query time

The `OrderByRecordCount` attribute sets the refinement order at query time.

Setting the `OrderByRecordCount` attribute to **true** in the `RefinementConfig` element sets the order in which refinements will be displayed, at query time, as in this example:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State/>
   <NavigationMenuConfig Id="NavigationMenu"
      <RefinementGroupConfig Name="WineCharacteristics" Expose="true">
        <RefinementConfig
          Name="WineType"
          Expose="true"
          OrderByRecordCount="true"
          MaximumCount="100" />
      </RefinementGroupConfig>
   </NavigationMenuConfig>
</Request>
```

This setting overrides the setting (either `lexical` or `record-count`) for refinement order that you can specify in the `system-navigation_Sorting` in the PDR for a refinement.

## Refinement order for managed attribute values

When you create a managed attribute value, you can optionally specify a static rank, which ranks the managed attribute values within each managed attribute's hierarchy. The two tables below show how the rank value (if it exists) affects the resulting refinement order.

For **unranked** managed attribute values, the refinement order is as follows:

| OrderByRecordCount setting | system-navigation_Sorting setting | Resulting sort order used |
|---|---|---|
| `true` | `lexical` | `record-count` |
| `true` | `record-count` | `record-count` |
| `false` | `lexical` | `lexical` |
| `false` | `record-count` | `lexical` |
| not used in `RefinementConfig` | `lexical` | `lexical` |
| not used in `RefinementConfig` | `record-count` | `record-count` |

For **ranked** managed attribute values, the refinement order is as follows:

| OrderByRecordCount setting | system-navigation_Sorting setting | Resulting sort order used |
|---|---|---|
| `true` | `lexical` | `record-count` |
| `true` | `record-count` | `record-count` |
| `false` | `lexical` | rank value |
| `false` | `record-count` | rank value |
| not used in `RefinementConfig` | `lexical` | rank value |
| not used in `RefinementConfig` | `record-count` | `record-count` |

## Retrieving the full path of hierarchical refinements

For managed attribute groups (which are groups of those attributes that contain hierarchy), you can request hierarchy information about a refinement with the `ReturnFullPath` specified in `NavigationMenuConfig`. In addition, for managed attribute values, hierarchy information is returned with `DimensionHierarchy` and `DimensionValueWithPath` types in any record list request.

Hierarchy information represents refinements behind a particular managed attribute. For example, if a `ProductCategory` managed attribute contains one level of hierarchy (`CAT_COMPONENTS`) and the current query is at the category components level, the full path of hierarchical refinements can be represented by the following list:

```
ProductCategory > CAT_COMPONENTS > Brakes
```

Refinement values, in this case specific components, may still exist for the `Brakes` refinement to refine the query even further.

## About navigation on attributes with hierarchy

Managed attributes represent a hierarchical relationship where records assigned to a particular value are implicitly assigned to all of the ancestors of that value. In the wine records example, the classification hierarchy includes the path Wine Type : Red > Merlot. This means that any record tagged to "Merlot" is implicitly tagged to "Red."

Refining to, or grouping by, "Red" will display all records mapped to "Merlot" (as well as any records mapped directly to "Red"). With hierarchical attributes, refinements continue to be generated for follow-on navigation. For example, the user may be able to click "Merlot" to see just the Merlots, excluding items tagged directly to "Red", or items tagged to "Cabernet" or another sibling of "Merlot."

To summarize, the expected behavior of managed attributes with hierarchy is that at any point in the navigation, Endeca Server not only returns attributes tagged with the user-selected value, but also those attributes that are implicitly tagged with the value that is above it in hierarchy. In other words, it is not possible to retrieve attribute values at single levels. This consideration is important when deciding which types of attributes — standard or managed, you should create for the records in your data domain. If, for example, the users of your front-end application would like to retrieve all records tagged with a particular value, then this value should belong to a standard (and not managed) attribute where hierarchy is not utilized.

## Retrieving hierarchy information for attribute groups

To request the full path of hierarchical refinements for an attribute group, use the `ReturnFullPath` attribute on `NavigationMenuConfig`. The `ReturnFullPath` has the following values:

| Attribute | Description |
|---|---|
| `ReturnFullPath` | Specifies whether to return the full path of hierarchical refinements with the response. This setting is relevant in navigation queries for refinements and breadcrumbs.<br>If set to `true`, the returned refinement contains the full path to its parent refinement values, as in `ProductCategory` > `CAT_COMPONENTS` > `Brakes`.<br>If set to `false`, returns only the refinement, without the path to its ancestors. The default is `false`. |

The format of the `NavigationMenuConfig` is shown in this example. It uses the `ReturnFullPath` attribute set to `true` for the existing attribute group "ProductCategories":

```
<NavigationMenuConfig Id="NavigationMenu" ReturnFullPath="true">
   <RefinementGroupConfig Name="ProductCategories" Expose="true">
     <RefinementConfig Name="CAT_COMPONENTS" Expose="true" MaximumCount="3"/>
   </RefinementGroupConfig>
</NavigationMenuConfig>
```

For a flat managed attribute with no hierarchy, the refinement parent will always be the attribute root, because there would be no further refinements if a value had already been selected for the attribute.

Refinements for a given managed attribute can only be returned on the same level within the attribute. For example, Endeca Server could never return a list of refinement choices that included a mix of countries, states, and regions. In all cases where hierarchy is explicitly defined for an attribute, only refinements on an equal level of hierarchy will be returned for a given query.

### Retrieving hierarchy information for managed attribute values

To retrieve hierarchy information on managed attribute values, you can use a query that requests a record list, with the `RecordListConfig` type.

In the response to a `RecordListConfig` query, the following two types include hierarchy and path information for managed attributes:

- The `DimensionHierarchy` complex type returns a collection of paths from specified managed attributes.
- The `DimensionValueWithPath` complex type specifies a path to a refinement attribute value from the root of that managed attribute.

For example, consider this abbreviated example of a record list query:

```
<ns:RecordListConfig Id="RecordList" MaxPages="60">
   <ns:Column>ProductCategory</ns:Column>
   <ns:RecordsPerPage>200</ns:RecordsPerPage>
   <ns:Page>2</ns:Page>
   <ns:Sort Key="Description" Direction="Ascending"/>
</ns:RecordListConfig>
```

This request returns hierarchy information and hierarchy paths for the managed attribute `ProductCategory`. In the following abbreviated example of the response, you can see the returned hierarchy information:

```
<cs:DimensionHierarchy>
   <cs:DimensionValueWithPath>
      <cs:DimensionValue DimensionName="ProductCategory"
       Spec="4">Handlebars</cs:DimensionValue>
      <cs:DimensionValue DimensionName="ProductCategory"
        Spec="CAT_COMPONENTS">Components</cs:DimensionValue>
      <cs:DimensionValue DimensionName="ProductCategory"
        Spec="/">ProductCategory</cs:DimensionValue>
   </cs:DimensionValueWithPath>
   <cs:DimensionValueWithPath>
      <cs:DimensionValue DimensionName="ProductCategory"
        Spec="6">Brakes</cs:DimensionValue>
      <cs:DimensionValue DimensionName="ProductCategory"
        Spec="CAT_COMPONENTS">Components</cs:DimensionValue>
      <cs:DimensionValue DimensionName="ProductCategory"
        Spec="/">ProductCategory</cs:DimensionValue>
   </cs:DimensionValueWithPath>
</cs:DimensionHierarchy>
```

# Performance impact of returning and displaying refinements

This topic summarizes performance impact for returning and displaying refinements, refinement ordering, refinement counts, and using multi-select managed attributes.

### Performance impact of returning and displaying refinements

Run-time performance of the Dgraph is directly related to the number of refinement values being computed for display. Only request refinement values if you are planning to display them in the front-end application. If any refinement values are being computed by the Dgraph process, but not being displayed by the application, this negatively affects performance. Attributes containing large numbers of refinements also affect performance.

Additionally, even exposing a large number of attributes (not their individual values/refinements) within each attribute group can have performance implications. This is because, for a query that returns a large number of attributes, the Dgraph needs to compute whether any valid refinements exist for each of the attributes. While this computation has a lower performance cost than listing the actual refinements, it can still have

performance impact, because, even if an attribute does not have any valid refinements, the Dgraph checks all the assignments on records to determine this.

Finally, retrieving a full list of refinements (both suggested refinements and applied refinements, which includes explicitly-selected and implicit), has performance implications.

If you must return a large number of attributes, to offset performance costs, consider increasing the system cache: Determine the amount of free RAM on the system, while the Dgraph is under load. If you are seeing a fair amount of free memory, consider increasing the cache size by that amount.

## Performance impact of refinement ordering

You can use the data domain configuration flag, `--refinement-sampling-min`, to specify the minimum number of records to sample during refinement computation (for managed attributes only). This option is useful because sampling the entire navigation state during the refinement computation can be one of the more performance intensive operations for the Dgraph.

For most applications, larger values for the data domain configuration flag, `--refinement-sampling-min`, reduce performance without improving the quality of refinement ordering. For some applications with extremely large, non-hierarchical attributes (if they cannot be avoided), larger values can meaningfully improve refinement ordering quality with minor performance cost. You specify this flag as:

```
endeca-cmd --put-dd-profile <profile-name> --refinement-sampling-min
```

## Performance impact of refinement counts

Dynamic statistics on records are expensive computations. You should only enable a managed attribute for dynamic statistics if you intend to use the statistics. Because the Dgraph does additional computation for additional statistics, there is a performance cost for those refinement counts that you are not using.

## Performance impact of multi-select managed attributes

Tagging an attribute as multi-select has an impact on performance. Users will take longer to refine the list of results, because each selection from a multi-select attribute still allows for further refinements from that attribute. Also, refinements for `multi-or` attributes are more expensive.

Chapter 16

# Attribute Groups

This section discusses how to implement attribute groups.



## About attribute groups

***Attribute groups*** are ordered lists of attributes. They are stored in the Oracle Endeca Server as records.

Attribute groups are useful for organizing a large number of attributes in the user interface of your front-end application, such as Studio. You can define a set of attribute groups to be displayed, assign attributes to each group, and determine the display order of the groups and attributes.

Because you define the attribute groups, you can group the attributes in any way that makes sense for your data. Do not confuse attribute groups with entity attribute groups (also called view attribute groups). Entity attribute groups are created with the Entity and Collection Configuration Web Service and are described in the chapter *Working with Entities on page 231*.

You can assign an attribute to more than one of your attribute groups. There is also a default `Other` attribute group containing all of the attributes that you have not assigned to a group.

There is no impact on Endeca Server performance from using attribute groups — the Endeca Server evaluates attribute groups at run-time.

## Configuring and using attribute groups in Studio

In Studio, lists of attributes can be displayed in attribute groups.

This includes:

- For application administrators, when configuring Studio components
- For end users, when viewing components such as the **Available Refinements** component

From the **Attribute Groups** page, power users can:

- Create and delete attribute groups
- Add and remove the attributes in each attribute group
- Set the default display order of the attributes within each group. You cannot change the group display order.

For information on using and configuring attribute groups in Studio, see the *Oracle Endeca Information Discovery Studio User's Guide*.

**Note:** In addition to creating attribute groups in Studio, you can also create them by sending Configuration Web Service requests to the Oracle Endeca Server.

# Working with attribute groups using the API

This section provides examples for using use Configuration Web Service requests to create and manage groups.

The Configuration Web Service has the following operations for attribute groups:

- `putGroups`
- `listGroups`
- `getGroups`
- `exportGroups`
- `importGroups`
- `updateGroupConfigs`
- `deleteGroups`

Each of these operations requires specifying `configTransaction` as the top-level element, followed by one of the group operations. For a list of operation descriptions, see *Configuration Web Service operations on page 54*.

For additional information about the Conversation Web Service WSDL and the Configuration Web Service WSDL, see the *Oracle Endeca Server API References*.



## Creating attribute groups

You can use the Configuration Web Service to create attribute groups.

The `putGroups` operation creates one or more attribute groups. For example, this request illustrates how to create an attribute group named Ratings that has three standard attributes (PriceRange, ReviewScore, and Designation):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
 <soapenv:Header/>
  <soapenv:Body>
   <ns:configTransaction>
    <ns:putGroups>
      <ns1:group key="Ratings" displayName="Product Ratings">
       <mdex-property_Key>PriceRange</mdex-property_Key>
       <mdex-property_Key>ReviewScore</mdex-property_Key>
```

```
       <mdex-property_Key>Designation</mdex-property_Key>
      </ns1:group>
     </ns:putGroups>
    </ns:configTransaction>
  </soapenv:Body>
</soapenv:Envelope>
```

Keep in mind that the group name (the `key` attribute) must use an NCName format.

To create an attribute group in the Endeca data domain:

1. Make sure that the Oracle Endeca Server and the data domain are running. Access the Configuration Web Service for the data domain: `http://localhost:<port>/ws/config/dataDomain?wsdl`.
2. Make a SOAP request to the Configuration Web Service as shown above, indicating the key of the new group, and its display name. (Omit specifying other optional attributes because they are not used by the Endeca Server).

   If the request is successful, the response will look like this abbreviated example:

   ```
   <soapenv:Body>
    <config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3
   /0"/>
   </soapenv:Body>
   ```

3. Issue a request for listing groups, to verify that this group is included, as in the following example:

   ```
   <ns:configTransaction>
      xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
      xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
             <ns:listGroups/>
   </ns:configTransaction>
   ```

   The response should include a `Ratings` group.

## Retrieving lists of groups

To retrieve a list of attribute groups, use a request with the `AttributeGroupListConfig` type.

The `AttributeGroupListConfig` syntax is:

```
<AttributeGroupListConfig Id="?">
    <StateName>?</StateName>
</AttributeGroupListConfig>
```

where:

- `Id` is an optional attribute that provides an arbitrary identifier for this configuration.
- `StateName` is an optional attribute that specifies the name of a state in the request. Note that specifying a state has no effect on the results (even in a request with multiple states).

To retrieve a list of groups:

1. Use a request similar to the following example:

   ```
   <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
      <State/>
      <AttributeGroupListConfig Id="AttrGroups"/>
   </Request>
   ```

The Conversation Web Service request contains a list of groups that are currently defined, specifying each group's display name and the number of attributes in each group. Information about each group is returned inside the `GroupSummary` element of the `AttributeGroupList` response, as shown in this example:

```
<cs:Results ...>
   <ns:State .../>
   <cs:AttributeGroupList Id="AttrGroups">
      <cs:GroupSummary Key="WineGroup" Cardinality="4">
         <cs:Record>
            <cs:attribute name="system-group_DisplayName" type="mdex:string">Wine Refs</cs:attribute>
            <cs:attribute name="system-group_Key" type="mdex:string">WineGroup</cs:attribute>
         </cs:Record>
         <cs:GroupMembers>
            <cs:attribute name="mdex-property_Key">WineType</cs:attribute>
            <cs:attribute name="mdex-property_Key">Wine</cs:attribute>
            <cs:attribute name="mdex-property_Key">Region</cs:attribute>
            <cs:attribute name="mdex-property_Key">Flavors</cs:attribute>
         </cs:GroupMembers>
      </cs:GroupSummary>
   </cs:AttributeGroupList>
</cs:Results>
```

In this example, one group (named WineGroup) is returned. The `Cardinality` attribute specifies the number of attributes in the group. The attributes for each group member are also listed.

**Note:** In a response, you may also notice a group `system-navigation_InternalGroup` (not shown in this example), which contains all of the attributes that are not members of any other user-created groups. This group is used by the Oracle Endeca Server and Studio and is not intended to be used in your application.

## Retrieving groups

Any request that asks for refinements is also requesting groups, if the attributes to be returned are configured as part of groups.

In other words, the Conversation Web Service returns groups for those types of queries that return refinements. Any attributes returned from the Conversation Web Service as refinements are returned as part of their respective groups.

The request for groups is implemented with the `RefinementGroupConfig` element of the Conversation Web service request. This element contains one or more `RefinementConfig` elements that list which attributes, out of all valid properties returned with a navigation query, should return actual refinement values. Note that only the top-level refinement values are returned.

The complex type `RefinementGroupConfig` has the following format:

```
<complexType name="RefinementGroupConfig">
   <sequence>
     <element maxOccurs="unbounded" minOccurs="0"
       name="RefinementConfig" type="cs_v3_0:RefinementConfig"/>
   </sequence>
   <attribute name="Name" type="cs_v3_0:NonEmptyString" use="required"/>
   <attribute name="Expose" type="boolean" use="required"/>
   <attribute name="ExposeAllPropertyRefinements" type="boolean"/>
 </complexType>
```

**Note:** The type `"cs_v2_0:RefinementConfig"` indicates the version of the web service. In this example, the version is 2.0. It may or may not correspond to the version of the Conversation Web Service that you are using and that is currently supported.

The meanings of the attributes are:

| Attribute | Description |
|---|---|
| `Name` | Required. The name of the group. |
| `Expose` | Required. Specify `true` to expose all top-level attributes in the group, or `false` (the default) to just show the root of the group.<br>  **Note:** If an attribute is a managed attribute, it contains a hierarchy of attributes under its root. Whether these nested attributes are exposed is controlled by the `Expose` attribute on the `RefinementConfig` element for each attribute within a managed attribute. The default for `Expose` is `false`. |
| `ExposeAllPropertyRefinements` | Optional. If set to `true`, specifies whether to expose all attribute refinements underneath each managed attribute that has them. The default is `false`.<br>This setting supersedes the `Expose` attribute on the `RefinementConfig` element for each attribute refinement. |

Groups are returned in a `NavigationMenuItemGroup` element that contains one or more `NavigationMenu` elements, each of which returns refinements in the `NavigationMenuItem`. Here is the format for the `NavigationMenuItemGroup`:

```
<complexType name="NavigationMenuItemGroup">
  <sequence>
    <element maxOccurs="unbounded" minOccurs="0" name="NavigationMenuItem"
      type="cs_v3_0:NavigationMenuItem"/>
  </sequence>
  <attribute name="HasRefineableProperties" type="boolean"/>
  <attribute name="Name" type="string" use="required"/>
</complexType>
```

The required attribute `HasRefineableProperties` specifies whether a group has attributes that could be refined further.

 **Note:** From the perspective of controlling the groups behavior in the front-end application, another attribute may be useful. It is the `ExposureControl` attribute of type Boolean, on the `NavigationMenuItem`. If set to `false` (the default), it does not expose refinements contained within `NavigationMenuItem`. If set to `true`, it exposes the collection of refinements.

To request groups:

1. In the Conversation Web Service request, for each group, specify its name and whether to expose all top-level attributes in the group by specifying the value of `Expose` attribute on the `RefinementGroupConfig` element. Optionally, you can also use this attribute on the refinements within the group.

   In this example, two groups are requested, `WineGroup` and `ProvenanceGroup`, but exposing top-level properties is requested for `WineGroup` only:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
  <State/>
  <NavigationMenuConfig MaximumRefinementCount="10" ReturnFullPath="true"
    ExposeAllRefinements="false" Id="Navigation">
    <RefinementGroupConfig Name="WineGroup" Expose="true">
       <RefinementConfig Name="Flavors" Expose="true" MaximumCount="2"/>
    </RefinementGroupConfig>
    <RefinementGroupConfig Name="ProvenanceGroup" Expose="false"/>
  </NavigationMenuConfig>
</Request>
```

The Conversation Web Service result includes results for one group, `WineGroup`, for which refinements were requested to be exposed.

**Note:** When a group is retrieved with the Conversation Web Service, the attribute ordering is determined by the order in which attributes were listed when the group was initially defined (either in Studio, or using the Configuration Web Service).

```
<cs:NavigationMenu Id="Navigation">
   <cs:NavigationMenuItemGroup Name="WineGroup" HasRefineableProperties="true">
      <cs:NavigationMenuItem Name="WineType" DisplayName="Wine Type" MultiSelect="None" HasMore
="true">
         <cs:ExposureControl Exposed="false"/>
         <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
         <cs:FullPath>
            <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
         </cs:FullPath>
      </cs:NavigationMenuItem>
      <cs:NavigationMenuItem Name="Wine" DisplayName="Wine Name" MultiSelect="None" HasMore="true">
         <cs:ExposureControl Exposed="false"/>
      </cs:NavigationMenuItem>
      <cs:NavigationMenuItem Name="Region" DisplayName="Region Grown" MultiSelect="None" HasMore
="true">
         <cs:ExposureControl Exposed="false"/>
      </cs:NavigationMenuItem>
      <cs:NavigationMenuItem Name="Flavors" DisplayName="Flavors" MultiSelect="None" HasMore="true">
         <cs:ExposureControl Exposed="true"/>
         <cs:Refinement Name="Flavors" Spec="Almond" Label="Almond" Count="1312"/>
         <cs:Refinement Name="Flavors" Spec="Anise" Label="Anise" Count="1626"/>
      </cs:NavigationMenuItem>
   </cs:NavigationMenuItemGroup>
</cs:NavigationMenu>
```

## Examples of other operations on groups

Using operations of the Configuration Web Service, you can obtain a group summary with the group's key, name and the number of standard attributes. You can also obtain a number of attributes in a specific group, as well as import, export and update groups (although these operations are used primarily through Studio).

### Getting group summary information

The `listGroup` operation returns summary information about the attribute groups that are currently defined in the Endeca data domain. The group summary information consists of:

- The name of the group (the `key` attribute), which uses the NCName format.
- The display name of the group (the `displayName` attribute), which uses a non-NCName-format. This name is typically used in front-end displays where a user-friendly format is needed.

- The number of standard attributes that are members of the group (the `cardinality` attribute). Note that the names of the member standard attributes are not listed.

The `listGroup` operation takes no arguments. For example:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:listGroups/>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

The response should look similar to this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3
/0">
         <groupSummaries xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09">
            <groupSummary key="Channels" cardinality="6" displayName="Channels"/>
            <groupSummary key="Countries" cardinality="10" displayName="Countries"/>
            <groupSummary key="Customers" cardinality="30" displayName="Customers"/>
            <groupSummary key="Products" cardinality="22" displayName="Products"/>
            <groupSummary key="Promotions" cardinality="11" displayName="Promotions"/>
            <groupSummary key="Sales" cardinality="16" displayName="Sales"/>
            <groupSummary key="Times" cardinality="38" displayName="Times"/>
            <groupSummary key="system-navigation_InternalGroup" cardinality="54"/>
         </groupSummaries>
      </config-types:results>
   </soapenv:Body>
</soapenv:Envelope>
```

This group summary example shows that there are seven user-created attribute groups (such as Channels and Products). The group named `system-navigation_InternalGroup` is a collection of system primordial attributes (such the attributes on PDRs).

## Getting the member attributes of a specific group

The `getGroups` operation can retrieve information about one or more specific attributes, using this structure in the request:

```
<ns:getGroups>
   <ns1:groupSummary key="?" displayName="?" cardinality="?"/>
</ns:getGroups>
```

where `key` is the only required attribute indicating the name of the specific group. Note that the name is case sensitive.

For example, this request:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:getGroups>
            <ns1:groupSummary key="Channels"/>
         </ns:getGroups>
```

```
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

returns this information about the Channels attribute group:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3
/0">
         <groups xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09">
            <group key="Channels" displayName="Channels">
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_CLASS</mdex-property_Key>
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_CLASS_ID</mdex-property_Key>
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_DESC</mdex-property_Key>
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_ID</mdex-property_Key>
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_TOTAL</mdex-property_Key>
               <mdex-property_Key type="mdex:string" xmlns="">CHANNEL_TOTAL_ID</mdex-property_Key>
            </group>
         </groups>
      </config-types:results>
   </soapenv:Body>
</soapenv:Envelope>
```

## Getting detailed information about groups

The `exportGroups` operation is basically a combination of the `listGroup` and `getGroups` operations. That is, it returns a list of your attribute groups with detailed information about each group in the `getGroups` format. Note that while the `system-navigation_InternalGroup` is listed, no details are provided on its member attributes.

The `exportGroups` syntax is:

```
<ns:configTransaction>
  <ns:exportGroups/>
</ns:configTransaction>
```

## Importing groups

To utilize `importGroups`, use this structure in the request:

```
<ns:importGroups>
   <ns1:group key="?" displayName="?">
      <mdex-property_Key>?</mdex-property_Key>
   </ns1:group>
</ns:importGroups>
```

where `mdex-property_Key` is the primary key of the group.

This request replaces the specified group with another group of the same name, but with the new list of attributes. For example, if an existing group contained three attributes, you can use `importGroups` to replace this group with a group that will contain only two of them. The keys for the attributes you want to include must be specified in the `importGroups` request.

## Updating the group configuration

The `updateGroupConfigs` operation replaces the assignment on the group description record with a new assignment. The operation's syntax is:

```
<ns:updateGroupConfigs>
   <ns1:record>
```

```
      <system-group_DisplayName>?</system-group_DisplayName>
      <system-group_Key>?</system-group_Key>
   </ns1:record>
</ns:updateGroupConfigs>
```

You specify a `system-group_Key` indicating which group to update, and zero or more assignments in the group description record, such as an assignment on the display name, if an existing group does not have one.

For example, the group `system-navigation_InternalGroup` is a group that contains all attributes that do not belong to any user-specified groups. This group is created automatically and does not have a display name initially. To provide a display name "Other attributes" for this group, send the following request to the Configuration Web Service running on the particular data domain:

```
<config-service:updateGroupConfigs>
   <mdex:record>
     <system-group_DisplayName>Other Attributes</system-group_DisplayName>
     <system-group_Key>system-navigation_InternalGroup</system-group_Key>
   </mdex:record>
</config-service:updateGroupConfigs>
```

# Part V

## Breadcrumbs, Precedence Rules, and Entities

Chapter 17

# Breadcrumbs

The section discusses how to implement breadcrumbs.



## About breadcrumbs

Breadcrumbs let you summarize any Guided Navigation selections, keyword searches, or range filters specified by the end user.

Breadcrumbs represent the following information that was passed to the navigation state by the Conversation Web Service response:

- Selected refinement values that were used to query for the current record set.
- Keyword searches that were used to query for the current record set.
- Range filters that have been selected for the query.

Any standard or managed attribute value available in the index of the particular data domain can be selected as a breadcrumb.

Breadcrumbs honor EQL record filters (such as security filters), but do not display them.

Breadcrumbs can reflect spelling correction and DYM (Did You Mean) information returned by the Dgraph process of the Oracle Endeca Server in response to keyword search queries.

In Studio, the **Selected Refinements** component lets you display breadcrumbs made with navigation queries (when users select refinement values or range filters for navigation), and keyword search queries.

For example, here is how user selections made in the **Available Refinements** component are reflected in the **Selected Refinements** component:

- When the user selects a refinement in the **Available Refinements** component, it is reflected as a breadcrumb in the **Selected Refinements** component.
- The user can select an additional refinement in the **Available Refinements** component, thereby narrowing down the scope of the record set for the query.
- Alternatively, the user can remove a refinement value from the **Selected Refinements** component, which increases the scope of the record set for the query.

# Implementing breadcrumbs with the API

This section describes how to issue queries requesting breadcrumbs using the Conversation Web Service.

The Conversation Web Service returns breadcrumb results for these types of queries:

- Navigation
- Search
- Range filters

For more information on the Conversation Web Service interface, see the *Oracle Endeca Server API References*. This reference contains documentation generated from the interface WSDL document.



## BreadcrumbConfig

The request for breadcrumbs is implemented with the `BreadcrumbConfig` type.

The `BreadcrumbConfig` type has this syntax:

```
<BreadcrumbConfig Id="?" ReturnFullPath="true">
   <StateName>?</StateName>
</BreadcrumbConfig>
```

The attributes are:

| Attribute | Description |
|---|---|
| `Id` | Required. An arbitrary identifier for this `BreadcrumbConfig`. |
| `ReturnFullPath` | Optional. Specifies whether to return the full path of all suggested hierarchical refinements with the response:<br>• If set to `true`, the returned breadcrumb contains the full path to its parent refinement values, as in `Wine` > `Red` > `Merlot`.<br>• If set to `false`, returns only the refinement, without the path to its ancestors. If not specified, the default is `false`.<br>This setting is relevant only in navigation queries that request breadcrumbs; it is ignored in search or range filter queries requesting breadcrumbs. |

| Attribute | Description |
|---|---|
| StateName | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the StateName element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the StateName element is used to reference that named state (as the state will be used in any event in the BreadcrumbConfig).<br>• If the request has an unnamed state, then the StateName element cannot be used. |

### Adding supplemental information

If spelling is enabled in the data domain configuration, and in addition to breadcrumbs, you may want the Conversation Web Service response to contain supplemental information about spelling suggestions and DYM (Did You Mean), a second SearchAdjustmentConfig type is required. If this type is included, spelling correction or DYM suggestions are returned with the breadcrumbs in the response.

If spelling is enabled, spelling correction occurs for breadcrumb results even if the SearchAdjustmentConfig is not included. However, while spelling correction takes place, the spelling correction and DYM suggestions are not returned in the response.

In the response, breadcrumbs are returned in the order in which they were added (requested).

## Retrieving breadcrumbs in a navigation query

An initial Conversation Web Service request that is made in response to a user-initiated navigation query (in which no selections have been made in the navigation state) does not yet return breadcrumbs. However, a subsequent request (in which the user made selections within the available attribute values) returns breadcrumbs, which represent explicitly-selected refinements.

In the Conversation Web Service request, make sure you specify the selection for a specific refinement, as well as the BreadcrumbConfig type.

In this example, the navigation state includes a selection of the WineType refinement, as well as the BreadcrumbConfig type:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <Name>Breadstate</Name>
      <SelectedRefinementFilter Name="WineType" Spec="/" Id="MyWines">
      </SelectedRefinementFilter>
   </State>
   <BreadcrumbConfig Id="BreadcrumbInfo" ReturnFullPath="true">
      <StateName>Breadstate</StateName>
   </BreadcrumbConfig>
</Request>
```

The Conversation Web Service result includes the original State request, followed by the Breadcrumbs type that lists attribute values identified as breadcrumbs, based on the user-selected navigation state, as in this sample result:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
```

```
            xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <State xmlns="http://www.endeca.com/MDEX/conversation/3/0"
          xmlns="http://www.endeca.com/MDEX/eql_parser/types">
      <Name>Breadstate</Name>
      <SelectedRefinementFilter Name="WineType" Spec="/" Id="MyWines"/>
   </State>
   <cs:Breadcrumbs Id="BreadcrumbInfo">
      <cs:RefinementBreadcrumb Name="WineType" DisplayName="Wine Type" Spec="/">
         <cs:DimensionValue DimensionName="WineType" Spec="/">WineType</cs:DimensionValue>
      </cs:RefinementBreadcrumb>
   </cs:Breadcrumbs>
</cs:Results>
```

## Example of breadcrumbs with spelling correction

Breadcrumbs returned by the Conversation Web Service in response to a record or value search query can reflect spelling correction.

There are two forms of spelling correction:

- Automatic spelling correction for record search and value search.
- Explicit spelling suggestions for record search (that is, Did You Mean)

The following requirements must be met to implement breadcrumbs that also return spelling correction information in response to a query:

- The spelling must be enabled in the data domain. To enable spelling, after you install the Oracle Endeca Server and create a data domain, run the `updateSpellingDictionaries` operation of the Data Ingest Web Service.
- The request must include the `BreadcrumbConfig` type. This ensures that breadcrumbs are returned:

  ```
  <BreadcrumbConfig Id="BreadcrumbInfo" ReturnFullPath="false">
     <StateName>Breadstate</StateName>
  </BreadcrumbConfig>
  ```

- If you would like to return DYM and spelling correction results with breadcrumbs, the request must include the `SearchAdjustmentConfig` type:

  ```
  <SearchAdjustmentConfig Id="SearchAdjust">
     <StateName>Breadstate</StateName>
  </SearchAdjustmentConfig>
  ```

The request in this example specifies a navigation state that includes a search for a user-entered word "pech" (a misspelling for "peach"). It illustrates a search request with a breadcrumb that needs to be corrected for spelling:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <Name>RecState</Name>
      <TextSearchFilter Key="Flavors" RelevanceRankingStrategy="numfields"
        Mode="AllPartial" EnableSnippeting="false" Language="en">pech</TextSearchFilter>
   </State>
   <RecordCountConfig Id="NumRecs">
      <StateName>RecState</StateName>
   </RecordCountConfig>
   <BreadcrumbConfig Id="Crumbs" ReturnFullPath="true">
      <StateName>RecState</StateName>
   </BreadcrumbConfig>
   <SearchAdjustmentConfig Id="CorrectSpell">
      <StateName>RecState</StateName>
   </SearchAdjustmentConfig>
```

```
</Request>
```

The response from the Conversation Web Service contains the original `State` from the request with search filter applied, as well as the original (not yet spelling-corrected) term of "pech". The `SearchAdjustments` response includes the automatically-corrected term "peach" in the `AdjustedTerms` element for `AppliedAdjustment`:

```
...
   <cs:SearchAdjustments Id="CorrectSpell">
        <cs:AppliedAdjustment>
             <cs:TextSearchFilter Key="Flavors" Mode="AllPartial"
                  RelevanceRankingStrategy="numfields">pech</cs:TextSearchFilter>
             <cs:AdjustedTerms>peach</cs:AdjustedTerms>
        </cs:AppliedAdjustment>
   </cs:SearchAdjustments>
</cs:Results>
```

For more information on the `SearchAdjustments` responses, see *Retrieving spelling corrections and DYM in query results on page 299*.

Chapter 18

# Precedence Rules

This section describes how to configure and use precedence rules.



## About precedence rules

Precedence rules provide a way to delay the display of attributes until they offer a useful refinement of the navigation state.

Precedence rules are defined in terms of a trigger attribute and a target attribute, where a user's selection of the trigger reveals the previously unavailable target attribute to the user. That is, precedence rules are triggered by implicit or explicit selections of either managed attribute values or standard attribute values. These triggers cause either managed attributes or standard attributes to be included as available refinements.

Precedence rule **triggers** can be expressed as:

- Managed attribute value (mval): triggered when a particular mval is selected. This can be configured to control whether the mval itself must be selected, or whether any child of the mval will trigger the rule. Using a root mval for a managed attribute effectively causes any selection within that managed attribute to trigger the rule.
- Standard attribute value (sval): triggered when a particular sval is selected.
- Standard attribute: triggered when any value in a particular standard attribute is selected

The precedence rule **target** can be a managed attribute or a standard attribute. If it is a managed attribute, the `mdex-dimension_EnableRefinements` property on its DDR should be set to `true` so that refinements can be displayed.

Note that either attribute type can trigger the other type. That is, a managed attribute value configured as a trigger can display a standard attribute, while a standard attribute (or standard attribute value) can be a trigger for a managed attribute target.

To illustrate the concept of precedence rules, assume that one might not want both the Country and State managed attributes to appear simultaneously in a geographical data set. A precedence rule could be defined so that the State managed attribute would appear only after a managed attribute value from the Country managed attribute is selected. This simplifies the user's navigation choices and avoids information overload by hiding the State managed attribute until it is relevant to the navigation state.

### Treatment of target attributes associated with multiple precedence rules

A target managed or standard attribute associated with more than one precedence rule is exposed when at least one associated trigger is selected.

For example, assume we have three managed attributes: Author, Region, and Language. We have two precedence rules:

```
Region > Author
Language > Author
```

In this case, the Author managed attribute is displayed after a managed attribute value from either the Region or Author managed attribute is selected.

### Precedence rules with non-existent sources

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

### Precedence rules versus hierarchical managed attributes

The creation of managed attributes can be facilitated with precedence rules. Consider the task of creating a Geography managed attribute as a hierarchy of country, state, and city. The hierarchy would need to be created manually, with Country as the root managed value. Each country managed value would have its corresponding states as children and each state its corresponding cities. In this scenario, the onus is on the knowledge worker to create and maintain this potentially enormous hierarchy.

Precedence rules offer a much simpler solution. The knowledge worker can produce the same results by creating three individual managed (or standard) attributes (Country, State, and City) and configuring precedence rules such that the State attribute is not presented until a country has been chosen and the City attribute is not presented until a state has been chosen. Because each attribute is flat, this solution involves much less initial and maintenance effort. Clearly, creating a managed attribute hierarchy by hand is a much more difficult task than creating the three flat attributes, configuring precedence rules, and letting contraction do the work to give the application the desired behavior (that is, to mimic the hierarchy).

## Managed attribute trigger types

During configuration, you can specify a rule type for managed attribute triggers.

Managed value triggers are either leaf or non-leaf, while standard attribute triggers are not typed. Non-leaf precedence rules display the target attribute if the trigger managed value or its descendants are in the navigation state. Leaf precedence rules display the target attribute only after descendants of the trigger managed value have been selected.

The two types differ in how the trigger value of the managed attribute is interpreted:

- For the non-leaf type, if the navigation state contains the trigger managed value or any of its descendants, then the target attribute is displayed.
- For the leaf type, only leaf managed values (managed values with no children) that are descendants of the specified trigger managed value cause the target attribute to be displayed. The presence of the specified trigger managed value in the navigation state does not cause the target attribute to appear. Hence, a leaf precedence rule requires that the trigger managed value have children.

When managed value triggers are created, the `isLeafTrigger` attribute sets the type.

### Non-leaf rule example

In this non-leaf rule example, we have a **Color** managed attribute with a child managed value named **blue**. We can construct a non-leaf precedence rule with **blue** as the trigger managed value and the managed attribute **ShadesOfBlue** as the target.

When the user drills into **Color** and selects **blue**, the target managed attribute **ShadesOfBlue** is displayed in the user interface.

### Leaf rule example

For leaf type rules, we will use a hierarchical managed attribute named **Country** and a second managed attribute named **State**. The **Country** attribute hierarchy looks like this:

```
Country
   - North America
       - Canada
       - Mexico
       - United States
   - Europe
       - England
       - Spain
       - Italy
```

Logically, a user should choose a country before choosing a state. We can use a leaf precedence rule to suppress the display of the **State** attribute until a leaf value in the **Country** managed attribute (an actual country as opposed to a continent) has been selected. To achieve this, a leaf precedence rule is constructed with the **Country** root managed value as the trigger and the **State** managed attribute as the target.

If the user drills into **Country** and selects an intermediate child managed value (North America or Europe), the target **State** attribute is not displayed. However, once the user has selected a leaf value from the **Country** managed attribute (United States, Canada, Mexico, England, Spain, or Italy) the **State** managed attribute appears.

## Precedence rule create operations

The Configuration Web Service has two operations to create precedence rules.

The two create operations are:

- The `putPrecedenceRules` operation creates each of the given precedence rules or updates them if they already exist. Existing rules that are not specified in the operation are not affected.
- The `importPrecedenceRules` operation first removes all existing precedence rules, and then adds the given ones. Use this operation when you want a new set of precedence rules.

The precedence rules take effect as soon as they are loaded into the Dgraph. The precedence rule is stored by the Dgraph process as a record in its data files, so that the precedence rules are automatically reloaded each time the Dgraph process is re-started.

Both operations use the same schema syntax for the `precedenceRule` element:

```
<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. |
| `isLeafTrigger` | If the trigger is a managed attribute, `isLeafTrigger` specifies a Boolean value that denotes the type of the trigger attribute value:<br>• 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).<br>• 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).<br>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. |

## putPrecedenceRules example

The following is an example of a `putPrecedenceRules` operation that creates a precedence rule named CityRule:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
```

```
         <ns:putPrecedenceRules>
             <ns1:precedenceRule
                key="CityRule"
                triggerAttributeKey="DimGeography_StateProvinceName"
                triggerAttributeValue="Victoria"
                targetAttributeKey="DimGeography_City"
                isLeafTrigger="true"/>
         </ns:putPrecedenceRules>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

# Creating precedence rules with Integrator ETL

You can use Integrator ETL to load precedence rules into the Endeca data domain.

Using Integrator ETL to create precedence rules is an alternate method to explicitly using the `putPrecedenceRules` and `importPrecedenceRules` operations of the Configuration Web Service.

To create precedence rules in Integrator ETL:

1. Create an input source file (such as a text file or a CSV file) that defines your precedence rules.
2. In Integrator ETL, create a graph that will read the input file, create the precedence rules, and send them to the Oracle Endeca Server.

Note that the precedence rules take effect as soon as they are loaded into the Dgraph.

The *Oracle Endeca Information Discovery Integrator ETL User's Guide* provides details on creating the precedence rules and loading them into the Endeca data domain.

# Precedence rule list and delete operations

The Configuration Web Service has operations for listing and deleting precedence rules.

## Listing precedence rules

The `listPrecedenceRules` and `exportPrecedenceRules` operations return information about your current set of precedence rules. The following is an example of the `listPrecedenceRules` operation:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:listPrecedenceRules/>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

If there are no precedence rules defined, the response would be:

```
<config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3/0">
   <precedenceRules xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09"/>
</config-types:results>
```

If there are defined precedence rules, the response will include one or more `precedenceRule` elements, as shown in this example:

```
<config-types:results xmlns:config-types="http://www.endeca.com/MDEX/config/services/types/3/0">
   <precedenceRules xmlns="http://www.endeca.com/MDEX/config/XQuery/2009/09">
      <precedenceRule key="AUS_Rule" triggerAttributeKey="DimGeography_CountryRegionName"
        triggerAttributeValue="Australia" targetAttributeKey="DimGeography_StateProvinceName"
        isLeafTrigger="false"/>
      <precedenceRule key="City_Rule" triggerAttributeKey="DimGeography_StateProvinceName"
        triggerAttributeValue="Victoria" targetAttributeKey="DimGeography_City"
        isLeafTrigger="false"/>
   </precedenceRules>
</config-types:results>
```

### Deleting precedence rules

The `deletePrecedenceRules` operation deletes one or more specified precedence rules. The only attribute that you need to specify is the name of the precedence rule in the `key` attribute, as in this example:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/config/services/types/3/0"
   xmlns:ns1="http://www.endeca.com/MDEX/config/XQuery/2009/09">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:configTransaction>
         <ns:deletePrecedenceRules>
            <ns1:precedenceRule key="City_Rule"/>
         </ns:deletePrecedenceRules>
      </ns:configTransaction>
   </soapenv:Body>
</soapenv:Envelope>
```

If the delete operation is successful, the response will be:

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

If the delete operation fails because the specified precedence rule does not exist, the response will be similar to this example:

```
<soapenv:Fault>
   <faultcode>soapenv:Client</faultcode>
   <faultstring>endeca-err:MDEX0001 : Invalid input : No record has an assignment of
     value 'Citty_Rule' for property 'mdex-precedenceRule_Key'</faultstring>
</soapenv:Fault>
```

In the example, the operation failed because the name of the precedence rule was misspelled.

# Precedence rules and implicit attribute value selection

When all records in the navigation state are assigned a given attribute value, that attribute value is an implicit selection.

In addition to being selected explicitly by the application, attribute values (either standard attribute values or managed attribute values) can be selected implicitly. For example, if all Champagnes are from France, then the explicit selection of **WineType>Champagne** causes the implicit selection of **Region>France**. Implicit selection is a function of the set of records in the navigation state, regardless of what combination of search, navigation, and record filters was used to obtain them.

Implicitly-selected attribute values trigger precedence rules in exactly the same way as explicitly-selected attribute values. This behavior helps ensure a consistent user experience, by providing the same attributes for

refinement of a given result set, regardless of whether that result set was obtained through search, navigation, or a combination of the two.

For this reason, two navigation paths leading to the same set of records will always have exactly the same set of navigation selections (differing only in whether the selections are implicit or explicit). Because of this equivalence, the set of precedence rules fired in both states will be identical.

## When precedence rules are overridden

Implicit selection of a precedence rule's trigger attribute value fires the rule. Under some circumstances, implicit selection of the rule's target managed value also fires it. Specifically, when a precedence rule's target managed value is implicit in the navigation state, and when refinements are available underneath that target managed value, the precedence rule fires and the target attribute is displayed. This occurs even when none of the rule's trigger values have been implicitly or explicitly selected. The Oracle Endeca Server treats any precedence rules targeting the parent managed attributes of these managed values as having fired, even though the rules' trigger values have not been selected.

For this reason, precedence rule target attributes may appear when no precedence rule trigger has been selected.

Chapter 19

# Working with Entities

This section describes how to create and manage entities.



## About entities

Entities provide you with intuitive, conceptual views on top of various categories in your data. They reflect the relational complexity between categories of data that is present in Endeca Server's flat data model, but that is not immediately visible.

An ***entity*** (or view, in Studio) represents a logical set of records that are derived from the physical records by aliasing, filtering, and grouping. An entity has its own metadata, which include names, types, and display names of the attributes, and the names and definitions of metrics.

**Note:** In Studio, entities are known as **views**. The Entity and Collection Configuration Web Service interface is used by Studio to create and manage views. For information on creating and managing views in Studio, see the *Oracle Endeca Information Discovery Studio User's Guide*.

When you create entities on top of various data categories, you map business concepts to complex data structures, based on how you would like to analyze data. Entities reestablish the relationship between categories of data once all data is loaded into Endeca Server.

You define entities by specifying metadata on them, such as their metrics. This allows you, as the data architect, to inform the business analyst about the relationship between different categories in your data, and to suggest metrics that can be requested on the entities. For example, metrics can provide information on which entities are useful to be grouped by, or to be aggregated upon.

Once you create entities, they serve as (aggregated) logical views of your data, allowing business analysts to run analytic queries on them.

You create entities using the `putEntity` or `putEntities` operations of the Entity and Collection Configuration Web Service. You can only create entities if the underlying attributes are already defined in your schema and exist in your data domain.

# Entity operations

The following table lists the entity-related operations in the Entity and Collection Configuration Web Service.

| Operation | Description |
|---|---|
| `listEntities` | List the entities that exist in the data domain. One use for this operation is to export the existing entities, for example during an upgrade procedure, in order to later import them to the Endeca data domain with the `putEntities` operation. |
| `validateEntity` | Validate an entity (either active or inactive) with the specified key and definition. |
| `validateEntities` | Validate multiple entities (either active or inactive) with specified definitions. |
| `putEntity` | Add an entity with the specified key and definition to the data domain.<br>The key must be valid according to the NCName format. The NCName format is defined in the W3C document Namespaces in XML 1.0 (Second Edition), located at this URL: *http://www.w3.org/TR/REC-xml-names/#NT-NCName*<br>For an entity to be created, its building blocks—the physical records and attributes—must already exist in the data domain.<br>If an entity with the specified key already exists in the corpus, it is replaced by the new entity with the same key. Note that the EQL statements defining the entity must be valid if the entity is active.<br>If an entity does not exist, the entity is created. Note that the entity is not created if its `isActive` flag is set to true and its EQL definition is not valid. |
| `putEntities` | Add multiple entities with the specified keys and definitions to the data domain. The keys must be valid according to the NCName format. |
| `deleteEntities` | Delete multiple entities for which keys are specified. |
| `deleteAllEntities` | Delete all entities that exist in the corpus without specifying any of their keys. |

# semanticEntity general syntax

The `semanticEntity` complex type defines an entity and all its attributes.

The `semanticEntity` syntax is:

```
<semanticEntity key="?" displayName="?" isActive="?">
   <definition>?</definition>
   <description>?</description>
   <attributes>
      <semanticAttribute name="?" displayName="?" datatype="?"
         isDimension="?" isKeyColumn="?" description="?">
         <property key="?">?</property>
      </semanticAttribute>
   </attributes>
   <metrics>
      <metric name="?" displayName="?" datatype="?" description="?">
```

```
         <definition>?</definition>
         <property key="?">?</ns:property>
      </metric>
   </metrics>
   <groups>
      <group key="?" displayName="?">
         <semanticAttributeKey name="?"/>
         <property key="?">?</property>
      </group>
   </groups>
   <property key="?">?</property>
</semanticEntity>
```

The meanings of its elements and attributes is as follows:

| Name of element or attribute | Description |
|---|---|
| `key` | Required. A unique identifier for the entity, which you provide when creating an entity. For example, you may create an entity with the key `Sales`. The key name must be in the NCName format. |
| `displayName` | Optional. Defines the display name which may be used by the front-end application such as Studio. The display name can use a non-NCName format. |
| `isActive` | Required. A boolean value that specifies whether this entity is active (`true`) or inactive (`false`). An Inactive entity's `definition` is not evaluated as part of a put operation or as an EQL query, and that definition is not concatenated onto queries. An inactive status thus this allows you to save entities with invalid or incomplete EQL definitions (which you will later correct) or to save the entity for later use (at which time you will activate it). An entity must be active if other EQL queries refer to it. When saving an entity, `isActive` must be explicitly set. |
| `definition` | Required. An EQL statement defining the entity. This EQL statement must create (or filter out) a virtual collection of records, based on the EQL expressions included in it. The EQL definition of an entity consists of one or more DEFINE statements separated by semicolons. The definition can refer to a named state.<br>The definition must include a DEFINE statement that matches the name (`key` attribute) of the entity. For example, the DEFINE statement for the Sales entity might be:<br>`<definition>`<br>`DEFINE Sales AS SELECT FactSales_SalesAmount AS SalesAmount,`<br>`DimReseller_ProductLine AS ProductLine,`<br>`DimSalesTerritory_SalesTerritoryCountry AS SalesTerritoryCountry,`<br>`DimDate_FiscalYear AS FiscalYear,`<br>`FactSales_SalesOrderNumber AS SaleOrderNumber`<br>`</definition>` |
| `description` | Optional. Provides descriptive text about the entity. |
| `attributes` | Optional. Represents a list of attributes in an entity. The `attributes` element may contain zero or more `semanticAttribute` elements. See below for details. |

| Name of element or attribute | Description |
|---|---|
| `metrics` | Optional. Provides a list of one or more suggested metric elements. The `metrics` element may contain zero or more `metric` elements. See below for details. |
| `groups` | Optional. Lets you create one or more entity attribute groups. See below for details. |
| `property` | Optional. Note that this is the `property` element for the entire `semanticEntity`. Lets you specify a string metadata global property for the entire entity. The key name must be in the NCName format. |

## attributes element

The `attributes` element may contain one or more `semanticAttribute` elements. Each attribute in an entity must correspond to an attribute specified in the EQL statement included in `definition`. Each `semanticAttribute` element defines a member attribute of the entity.

The syntax of the `semanticAttribute` element is:

```
<semanticAttribute name="?" displayName="?" datatype="?"
   isDimension="?" isKeyColumn="?" description="?">
   <property key="?">?</property>
</semanticAttribute>
```

For each `semanticAttribute` element, specify the following:

- `name` specifies a unique identifier for the attribute. The identifier must follow the NCName format.
- `displayName` is the name of the entity attribute in an easy-to-understand format. The display name can use a non-NCName format.
- `datatype` specifies a valid data type, such as `mdex:string`. Valid types are listed in the `mdex.xsd`.
- `isDimension` is set to `true` on attributes on which it is useful to do a GROUP BY. For example, attributes such as Size, Region, or Category should have `isDimension="true"`, indicating that they are managed attributes containing a hierarchy, and are candidates for GROUP BY statements in EQL.
- `isKeyColumn` is set to `true` if this attribute is part of the entity's composite key. The composite key on an entity is the set of entity attributes with `isKeyColumn` set to `true`. The default is `false`.
- `description` provides a brief description of the attribute.
- `property` sets the name (`key`) and value of the string metadata for this attribute. The key name must be in the NCName format.

This abbreviated example shows one of the several attributes based on which a `Sales` entity is created:

```
<attributes>
  <semanticAttribute name="SalesAmount" displayName="Sales Amount" datatype="mdex:double"
     isDimension="false" isKeyColumn="true" description="sales info">
     <property key="locale">EN</property>
  </semanticAttribute>
  ...
</attributes>
```

## metrics element

The `metrics` element can contain one or more `metric` elements. The syntax of the `metric` element is:

```
<metrics>
  <metric name="?" displayName="?" datatype="?" description="?">
     <definition>?</definition>
     <property key="?">?</property>
  </metric>
</metrics>
```

Specify these attributes for the `metric` element:

- `name` specifies a unique identifier for the metric. The identifier must follow the NCName format.
- `displayName` is the name of the metric in an easy-to-understand format. The display name can use a non-NCName format.
- `datatype` specifies a valid data type, such as `mdex:double`.
- `description` provides a brief description of the metric.
- `definition` is an EQL statement defining the metric. The `definition` element must contain an arithmetic formula in EQL used for aggregation when querying against the entity's attributes.
- `property` sets the name (`key`) and value of the string metadata for this metric. The key name must be in the NCName format.

Each metric must contain at least one aggregation function, such as `SUM(X)`, or `AVG(Y)`, where `X` and `Y` are attributes defined for the entity.

For example, an entity may include the attribute `SalesAmount`, and a metric `TotalSales`, defined as the sum of the values of the `SalesAmount` attribute:

```
<metrics>
  <metric name="TotalSales" displayName="Total Sale" datatype="mdex:double">
    <definition>sum(SalesAmount)</definition>
    <property key="currency">$</property>
  </metric>
</metrics>
```

## groups element

An entity attribute group (also called a view attribute group) consists of a set of entity attributes (which have been set via `semanticAttribute` elements).

The entity attribute group can also have a set of properties (key-value pairs) that are associated with the group. These properties let you provide metadata for the group, which can then be used by your front-end application (such as Studio). For example, you can use this metadata for order control (i.e., specifying which attribute will be used to sort the results).

Each group is defined by a `group` element and consists of attributes from this entity. The syntax of the `metric` element is:

```
<groups>
  <group key="?" displayName="?">
     <semanticAttributeKey name="?">
     <property key="?">?</property>
  </group>
</groups>
```

The meanings of the `group` attributes are:

- `group key` is a unique identifier for the entity attribute group. The identifier does not have to follow the NCName format.
- `displayName` lets you specify a more user-friendly name for the group.
- `semanticAttributeKey` (via its `name` attribute) specifies which entity attribute is added to the group. Thus, the `name` attribute of `semanticAttributeKey` corresponds to the `name` attribute of the `semanticAttribute` element described above.
- `property` sets the name (`key`) and value of the string metadata for this group. The key name must be in the NCName format.

This example creates an entity named Product, which has an entity attribute group named ProdDescription:

```
<semanticEntity key="Product" displayName="Product" isActive = "true">
   <definition>
    DEFINE Product AS SELECT productId AS productId, description AS description, price AS price
   </definition>
   <attributes>
      <semanticAttribute name="productId" datatype="mdex:string"
         isDimension="true" isKeyColumn="true">
      </semanticAttribute>
      <semanticAttribute name="description" datatype="mdex:string"
         isDimension="true" isKeyColumn="false">
      </semanticAttribute>
      <semanticAttribute name="price" datatype="mdex:double"
         isDimension="true" isKeyColumn="false">
         <property key="currency">$</property>
      </semanticAttribute>
   </attributes>
   <metrics/>
   <groups>
      <group key="ProdDescription" displayName="ProductId and Description Group">
         <semanticAttributeKey name="productId"/>
         <semanticAttributeKey name="description"/>
         <property key="sortBy">productId</property>
      </group>
   </groups>
   <property key="SalesArea">North America</property>
</semanticEntity>
```

The group has two entity attributes as members ("productId" and "description"). It also has a metadata property (named "sortBy") whose value can be used to sort the results by the "productId" attribute.

#### Example of an entity

To put the previously described portions of an entity definition together, consider the following use case.

When you load a list of sales transactions, you also are loading information about customers, products, and suppliers. You can create entities for each of them. Consider creating a `Sales` entity as a virtual set of records derived from the following attributes: `SalesAmount`, `ProductLine`, and `FiscalYear`.

When you define the `Sales` entity, you also provide metrics for it, allowing business analysts to issue queries in EQL against this entity. These metrics could be the `TotalSales`, defined as a sum of `SalesAmount`, or the `AvgSales`, defined as an average of `SalesAmount`.

This example illustrates a `Sales` entity defined on top of several entity attributes and listing two metrics that could be used in subsequent analytic queries against this entity:

```
<semanticEntity key="Sales" displayName="Sales Transactions" isActive="true">
<definition>
```

```
  DEFINE Sales AS
  SELECT FactSales_SalesAmount AS SalesAmount,
  DimReseller_ProductLine AS ProductLine,
  DimSalesTerritory_SalesTerritoryCountry AS SalesTerritoryCountry,
  DimDate_FiscalYear AS FiscalYear,
  FactSales_SalesOrderNumber AS SaleOrderNumber
</definition>
<description>Sales transaction information</description>
<attributes>
  <semanticAttribute name="SalesAmount" displayName="Sales Amount"
      datatype="mdex:double" isDimension="false" isKeyColumn="true">
      <property key="locale">EN</property>
  </semanticAttribute>
  <semanticAttribute name="ProductLine" displayName="Product Line"
      datatype="mdex:string" isDimension="true" isKeyColumn="false">
  </semanticAttribute>
  <semanticAttribute name="SalesTerritoryCountry" displayName="Sales Territory Country"
      datatype="mdex:string" isDimension="true" isKeyColumn="false">
  </semanticAttribute>
  <semanticAttribute name="FiscalYear" displayName="Year" datatype="mdex:int"
      isDimension="true" isKeyColumn="false">
  </semanticAttribute>
  <semanticAttribute name="SaleOrderNumber" displayName="Sale Order Number"
      datatype="mdex:string" isDimension="false" isKeyColumn="false">
  </semanticAttribute>
</attributes>
<metrics>
  <metric name="TotalSales" displayName="Total Sale" datatype="mdex:double">
    <definition>sum(SalesAmount)</definition>
    <property key="currency">$</property>
  </metric>
  <metric name="AvgSales" displayName="Average Sale" datatype="mdex:double">
    <definition>avg(SalesAmount)</definition>
  </metric>
</metrics>
<groups/>
<property key="SalesArea">North America</property>
</semanticEntity>
```

### Cache re-freshing and re-validation

`listEntities` operations return the contents of the cache without attempting to pro-actively update the cache. The contents of the cache are refreshed only via Conversation Service queries. Note that if you create a new entity, it will appear in a subsequent `listEntities` call only if you first issue a Conversation Service query to refresh the cache.

Entities are re-validated during every cache refill. Any EQL query that uses an invalid entity will generate an error and the entity's `isValid` flag will be set to false. Note that Endeca Server does validation only on the `Definition` and the `Metrics` provided.

## Sample entity requests

This topic includes examples of requests for listing, adding, deleting, and validating entities.

#### Listing entities

The following example of the request lists all entities that are present in the corpus:

```
<listEntities>
   <language>en</language>
```

```
</listEntities>
```

The response to this request returns a list of entities that are already added. The following `listEntitiesResponse` shows one sample entity:

```
<ns3:listEntitiesResponse xmlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
                          xmlns:ns3="http://www.endeca.com/endeca-server/sconfig/3/0">
   <ns3:validatedSemanticEntity isValid="true" key="SalesTrans"
         displayName="Sales Transaction" isActive="true">
      <ns3:definition>DEFINE SalesTrans AS
      SELECT FactSales_SalesAmount AS SalesAmount,
      DimReseller_ProductLine AS ProductLine</ns3:definition>
      <ns3:description>Sales transaction information</ns3:description>
      <ns3:attributes>
         <ns3:semanticAttribute name="SalesAmount" displayName="Sales Amount" datatype="mdex:double"
               isDimension="false" isKeyColumn="true" description="Amounts">
            <ns3:property key="locale">EN</ns3:property>
         </ns3:semanticAttribute>
         <ns3:semanticAttribute name="ProductLine" displayName="Product Line" datatype="mdex:string"
               isDimension="true" isKeyColumn="false" description="prod"/>
         <ns3:semanticAttribute name="SaleOrderNumber" displayName="Sale Number"
               datatype="mdex:string" isDimension="false" isKeyColumn="false" description="Orders"/>
      </ns3:attributes>
      <ns3:metrics>
         <ns3:metric name="TotalSales" displayName="Total Sales" datatype="mdex:double" description
="Totals">
            <ns3:definition>sum(SalesAmount)</ns3:definition>
            <ns3:property key="currency">$</ns3:property>
         </ns3:metric>
         <ns3:metric name="AvgSales" displayName="Average Sales" datatype="mdex:double"
               description="Averages">
            <ns3:definition>avg(SalesAmount)</ns3:definition>
         </ns3:metric>
      </ns3:metrics>
      <ns3:groups/>
      <ns3:property key="SalesArea">North America</ns3:property>
      <ns3:parsedDefinition>
         <ns2:statements returnTable="false" statementKey="SalesTrans">
            <ns2:selects attributeKey="SalesAmount">
               <ns2:expression xsi:type="ns2:AttributeRefExpression"
                   attributeKey="FactSales_SalesAmount"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
            </ns2:selects>
            <ns2:selects attributeKey="ProductLine">
               <ns2:expression xsi:type="ns2:AttributeRefExpression"
                   attributeKey="DimReseller_ProductLine"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
            </ns2:selects>
         </ns2:statements>
      </ns3:parsedDefinition>
   </ns3:validatedSemanticEntity>
</ns3:listEntitiesResponse>
```

In the `validatedSemanticEntity` element, the `isValid` attribute specifies whether the entity has been successfully validated.

If the entity has a dependency, the `dependentEntities` element will list that dependency.

### Validating an entity

Before adding an entity, it is useful to validate the syntax of the EQL statements in the entity definition.

To validate an entity, issue a request either with `validateEntity` (which validates a single entity) or with `validateEntities` (which can validate multiple entities). With both operations, you specify the entities inside the `semanticEntity` elements. Note that the entity is validated regardless of whether it is active or inactive.

The following abbreviated example illustrates the `validateEntity` request:

```
<validateEntity>
   <semanticEntity key="Sales" displayName="Sales Data" isActive="true">
   ...
   </semanticEntity>
</validateEntity>
```

If the request validates successfully, `validateEntityResponse` does not contain errors. Note that if the entity validation fails, the `isValid` flag on the entity record is set to `false` (regardless of its previous setting).

### Adding an entity

You can add one entity using a `putEntity` operation, or add multiple entities using `putEntities`.

In this abbreviated example, a `putEntity` operation is used in a request to add one entity, `Sales`, with two metrics, `TotalSales` and `AvgSales`:

```
<putEntity xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
<semanticEntity key="Sales" displayName="Sales Data" isActive="true">
<definition>
   DEFINE Sales AS SELECT FactSales_SalesAmount
   AS SalesAmount, DimReseller_ProductLine AS ProductLine,
   DimSalesTerritory_SalesTerritoryCountry AS SalesTerritoryCountry,
   DimDate_FiscalYear AS FiscalYear,
   FactSales_SalesOrderNumber AS SaleOrderNumber
</definition>
<description>Sales territorial information</description>
<attributes>
    <semanticAttribute name="SalesAmount" displayName="Sales Amount" datatype="mdex:double"
   isDimension="false" isKeyColumn="true"/>
    <semanticAttribute name="ProductLine" displayName="Product Line" datatype="mdex:string"
 isDimension="true" isKeyColumn="true"/>
    <semanticAttribute name="SalesTerritoryCountry" displayName="Sales Territory Country"
 datatype="mdex:string" isDimension="true" isKeyColumn="false"/>
    <semanticAttribute name="FiscalYear" displayName="Year" datatype="mdex:int"
 isDimension="true" isKeyColumn="false"/>
    <semanticAttribute name="SaleOrderNumber" displayName="Sale Order Number"
 datatype="mdex:string" isDimension="false" isKeyColumn="false"/>
</attributes>
<metrics>
  <metric name="TotalSales" displayName="Total Sale" datatype="mdex:double">
    <definition>sum(SalesAmount)</definition>
  </metric>
  <metric name="AvgSales" displayName="Average Sale" datatype="mdex:double">
    <definition>avg(SalesAmount)</definition>
  </metric>
</metrics>
<groups/>
</semanticEntity>
</putEntity>
```

The abbreviated example response from the Entity and Collection Configuration Web Service informs you how many entities were created or replaced:

```
<putEntityResponse xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
  <entityAdditionInformation numEntitiesAdded="1" numEntitiesReplaced="0"/>
</putEntityResponse>
```

### Deleting an entity

To delete individual entities (one or more), use the `deleteEntities` operation, specifying keys for the entities to be deleted, as in this example:

```
<deleteEntities xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <semanticEntityKey key="SalesInfo"/>
</deleteEntities>
```

The response indicates the number of entities deleted:

```
<deleteEntitiesResponse xmlns="http://www.endeca.com/endeca-server/sconfig/3/0">
   <numEntitiesDeleted>1</numEntitiesDeleted>
</deleteEntitiesResponse>
```

# Part VI

## Search Features

Chapter 20

# Record Search

This section discusses record search, which is an Oracle Endeca Server equivalent of full-text search, and is one of the fundamental building blocks of Oracle Endeca Server search capabilities.



## Record search overview

Record search (also called text search) allows a user to perform a keyword search against specific attribute values assigned to records.

The resulting records that have matching attribute values are returned, along with any valid refinement values.

Because record search returns a navigation page, it is important to remember that the record search parameter acts as a record filter in the same way that an attribute value does, even though it is not a specific value.

### Controlling record search

The following statements describe various aspects of record search behavior and how you can control it:

- Record search works against named collections, as described in *Specifying a collection for record search on page 248*.
- Although search interfaces are not mandatory for record searches, you can create a search interface if you want to specify one or more standard attributes to search. For information on search interfaces, see *Search Interfaces on page 252*.
- There are no Dgraph configuration flags necessary to enable record searching. If an attribute was properly enabled for record searching, it will automatically be available for record searching.
- You can use the data domain `--search-max` configuration flag to specify the maximum number of terms for record search for the data domain profile. The default is 10.

- You can use the data domain `--search-char-limit` configuration flag to specify the maximum length (in characters) of a search term for record search. The default is 132 characters. Any term exceeding this length will not be indexed for any form of record search, and thus that term will not be found.

## Supported languages for record search

For the list of supported languages for record search, see *Supported languages on page 154*.

You can specify the language ID in the `Language` attribute of the `TextSearchFilter` type, as in this example:

```
<TextSearchFilter Key="PROD_CATEGORY" RelevanceRankingStrategy="numfields"
   Mode="AllPartial" EnableSnippeting="false" Language="en">
   hardware
</TextSearchFilter>
```

This example uses `en` (American English) as the language for the record search query.

## Example of record search

For example, consider the following records:

| Rec ID | Managed attribute values for the managed attribute BikeType: | Values for the attribute "Name": | Values for the attribute "Description": |
|---|---|---|---|
| 1 | Road Bikes | Road-450 | can do double-duty for racing or long-range mileage... |
| 2 | Road Bikes | Road-550-W | its speed comes at the sake of comfort... |
| 3 | Touring Bikes | Touring-1000 | combines comfort and performance... |
| 4 | Mountain Bikes | Mountain-500 | this mountain bike has serious racing performance... |

When the user performs a record search on the Description attribute using the keyword `comfort`, the following objects are returned:

- 2 records (records 2 and 3)
- 2 refinement attribute values (Road Bikes and Touring Bikes)

When performing a record search on the Description attribute using the keyword `racing`, these objects are returned:

- 2 records (records 1 and 4)
- 2 refinement attribute values (Road Bikes and Mountain Bikes)

**Note:** In addition to basic record search, other features affect the behavior of record search, such as spelling support, relevance ranking of results, wildcard syntax, multiple attribute record searches, and attribute group record searches. These are discussed in detail in their respective sections.

# Configuring attributes for record search

The first step in implementing basic record search is to configure a standard attribute for record searching using either the Configuration Web Service directly or Integrator ETL (whose components use this service).

The `mdex-property_IsTextSearchable` attribute of a PDR enables the attribute for record searching. The valid settings for this attribute are:

- If set to `true`, the attribute is enabled for record search.
- If set to `false`, the attribute is not enabled for record search. This is the default.

You can change the value for this attribute using the `updateProperties` operation of the Configuration Web Service. It is recommended to change this setting on a data domain that does not yet contain any source records. Running this operation on a data domain with a large number of existing records causes the Dgraph process to reindex the data domain and has performance impact.

# Enabling hierarchical record search

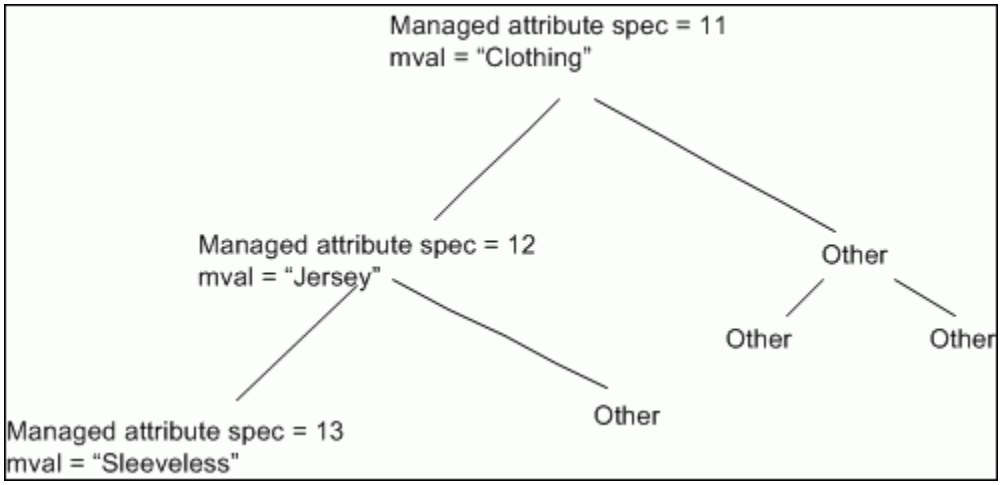
If you want to consider ancestor managed attribute values when matching a record search query, you can enable hierarchical record search.

By default, a record search that uses a managed attribute as the search key returns only those records that are assigned an attribute value whose text matches the search terms. In other words, when a managed attribute is used as the record search key, the text strings considered by record search for matching are the individual names of the attribute values within the attribute. The managed attribute name is automatically added as a searchable string.

(Additionally, if you added synonyms when creating managed attribute value records, end users can search for other text strings and still obtain the same records as they would get while searching for the original attribute value name.)

As part of creating searchable strings, record search does not consider implicit ancestor attribute values.

For example, consider the following managed attributes hierarchy:




In this hierarchy, the `Jersey` attribute (with an ID of 12) is an ancestor of the `Sleeveless` attribute (ID of 13). A search against the `Clothing` attribute for the keyword `sleeveless` matches any records assigned

the attribute value 13. But a search in `Clothing` for `sleeveless jersey` does not match these records, because record search does not normally consider implicit ancestor attribute value assignments.

In such cases, you may want record search to consider ancestor attribute values when matching a record search query. You can enable this sort of hierarchical record search by setting the `mdex-dimension_IsRecordSearchHierarchical` attribute to `true` in the managed attribute's DDR (Dimension Description Record), using the operations in the Configuration Web Service.

# Implementing record search in Studio

Record search queries in a Studio application are made from the **Search Box** component.

To make record search queries in Studio, you must add and configure the **Search Box** component. For details on this component, see the *Oracle Endeca Information Discovery Studio User's Guide*.

# Implementing record search with the API

This section describes how to issue record search queries using the Conversation Web Service API.

For more information on the Conversation Web Service interface, see the *Oracle Endeca Server API References*. This reference contains documentation generated from the interface WSDL document.

*Obtaining the available search keys*

*Record search filter*

## Obtaining the available search keys

The `AvailableSearchKeysConfig` complex type allows you to retrieve a list of the searchable attributes and search interfaces available in the data domain.

The `AvailableSearchKeysConfig` type identifies the items that are searchable — search interfaces and searchable properties. This type has the following format:

```
<AvailableSearchKeysConfig Id="?">
   <StateName>?</StateName>
</AvailableSearchKeysConfig>
```

where:

- `Id` is an optional attribute that provides an arbitrary identifier for this configuration.
- `StateName` is an optional attribute that specifies the name of a state in the request. Note that specifying a state has no effect on the results (even in a request with multiple states).

### Sample request for available search keys

This example shows how to make a request for available search keys using the `AvailableSearchKeysConfig` type:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State/>
   <AvailableSearchKeysConfig Id="MySearchKeys"/>
</Request>
```

Note that the `StateName` element is not used because the state is an empty, unnamed state.

### Response for available search keys

The response contains an `AvailableSearchKeys` component that lists all of the searchable keys in a single alphabetically ordered list, as shown in this example:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/3/0"
     xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <State xmlns="http://www.endeca.com/MDEX/conversation/3/0" x
       mlns:ns2="http://www.endeca.com/MDEX/eql_parser/types"
       xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"/>
   <cs:AvailableSearchKeys>
      <cs:AvailableSearchKey Interface="true">
         <cs:Key>AllWineSearch</cs:Key>
         <cs:DisplayName>AllWineSearch</cs:DisplayName>
      </cs:AvailableSearchKey>
      <cs:AvailableSearchKey Interface="false">
         <cs:Key>Description</cs:Key>
         <cs:DisplayName>Wine Description</cs:DisplayName>
      </cs:AvailableSearchKey>
      <cs:AvailableSearchKey Interface="false">
         <cs:Key>WineType</cs:Key>
         <cs:DisplayName>Wine Type</cs:DisplayName>
      </cs:AvailableSearchKey>
   </cs:AvailableSearchKeys>
</cs:Results>
```

Each `AvailableSearchKey` element lists the name of a searchable attribute or search interface (the `Key` sub-element), and the display name (which can have a non-NCName format). The `Interface` attribute distinguishes whether the search key is a searchable attribute or a search interface. If the search key is a search interface, the attribute is set to `true`. If the search key is not a search interface and is a searchable attribute, the attribute is set to `false`.

In this sample response, one search interface, `AllWineSearch`, and two attributes, `Description` and `WineType`, are listed as available search keys.

## Record search filter

A basic record search requires a `TextSearchFilter` type.

The syntax for a search request is shown in this example:

```
<TextSearchFilter Key="Prod_Category" RelevanceRankingStrategy="numfields"
   Mode="AllPartial" EnableSnippeting="true" SnippetLength="5" Language="en">
   electronics
</TextSearchFilter>
```

The text content of the `TextSearchFilter` type contains the search term(s). In the example, a record search is being made for the "electronics" keyword in the Prod_Category standard attribute.

The meanings of attributes for the `TextSearchFilter` type are as follows:

| Search attribute | Description |
|---|---|
| `Key` | Required. Specifies which standard or managed attribute will be evaluated when searching. You specify an attribute as a value for this parameter. You can also specify a search interface as a value. |
| `EnableSnippeting` | Optional. If set to `true`, enables snippeting. If set to `false`, disables snippeting. For details on snippeting, see *Snippeting in Record Searches on page 281*. |
| `SnippetLength` | Optional. Specifies the length of the snippet. |
| `Mode` | Optional. Specifies a match mode, which are described in *List of valid search modes on page 266*. If not specified, defaults to `All`. Note that the `Boolean` match mode cannot be used. |
| `RelevanceRankingStrategy` | Optional. Specifies a relevance ranking strategy. For details on relevance ranking, see *Relevance Ranking on page 310*. |
| `Language` | Optional. Specifies a language code for the search. Valid language codes are listed in the topic *Supported languages on page 154*. |
| *`searchTerm`* | Required. Specifies one or more terms to search for. The maximum number of characters in each search term is set by the `--search-char-limit` configuration flag (in the Endeca command `put-dd-profile`), which defaults to 132 characters. |

## Specifying a collection for record search

You can run a record search query against the records in a specific collection. To do so, specify the collection name in the `<CollectionName>` element of the `State` type.

For example, this query will search the records of the Sales collection:

```
<Request>
   <State>
     <Name>MySalesSearch</Name>
     <CollectionName>Sales</CollectionName>
     <TextSearchFilter Key="Prod_Category" RelevanceRankingStrategy="numfields"
        Mode="AllPartial" EnableSnippeting="true" SnippetLength="5" Language="en">
        electronics
     </TextSearchFilter>
   </State>
   <RecordListConfig Id="SalesList" MaxPages="20">
      <StateName>MySalesSearch</StateName>
      <Column>SalesAmount</Column>
      <RecordsPerPage>5</RecordsPerPage>
   </RecordListConfig>
</Request>
```

In the example, the name of the `State` is MySalesSearch and it specifies the Sales collection. In turn, the `RecordListConfig` type uses the `StateName` element to associate the MySalesSearch state to the config.

# Search query processing order

This section summarizes how the Dgraph process of the Oracle Endeca Server processes record search queries.

While this summary is not exhaustive, it covers the processing steps likely to occur is most application contexts. The process outlined here assumes that other features (such as spelling correction and thesaurus) are being used.

The Dgraph process uses the following high-level steps to process record search queries:

1. Record filtering
2. Tokenization
3. Spelling correction
4. Thesaurus expansion
5. Stemming
6. Primitive term and phrase lookup
7. Did you mean
8. Navigation filtering
9. EQL
10. Relevance ranking

**Note:** For Boolean search queries, tokenization, auto correction, and thesaurus expansion are replaced with a separate parsing phase.

## Step 1: Record filtering

If a record filter is specified, whether for security or any other reason, Endeca Server applies it before any search processing. The result is that the search query is performed as if the data set only contained records allowed by the record filter.

## Step 2: Tokenization

Tokenization is the process by which the Dgraph analyzes the search query string, yielding a sequence of distinct query terms.

## Step 3: Spelling correction

If spelling correction is enabled and triggered, the Dgraph implements them as part of the record search processing. If the spelling correction feature is enabled and triggered, the Dgraph creates spelling suggestions by enumerating (for each query term) a set of alternatives, and considering some of the combinations of term alternatives as whole-query alternatives. Each of these whole-query alternatives is subject to thesaurus expansion and stemming.

For example, if the tokenized query is `employee moral`, then `employee` may generate the set of alternatives {`employer`, `employee`, `employed`}, while `moral` may generate the set of alternatives {`moral`, `morale`}.

The two query alternatives generated as spelling suggestions might be `employer moral` and `employee morale`.

For details on the auto-correction feature, see *Spelling Correction and Did You Mean on page 295*.

### Step 4: Thesaurus expansion

The tokenized query, as well as each query alternative generated by spelling suggestion, is expanded by the Dgraph based on thesaurus matches. Thesaurus expansion replaces each expanded query term with an `OR` of alternatives.

For example, if the thesaurus expands `pentium` to `intel` and `laptop` to `notebook`, then the query `pentium laptop` will be expanded to:

```
(pentium OR intel) AND (laptop OR notebook)
```

This assumes the match mode is `All`. The other match modes (with the exception of `Boolean`) behave analogously.

If there is a multiple-word thesaurus match, then `OR` is used on the query itself to accommodate the various ways of partitioning the query terms.

For example, if `high speed` expands to `performance`, then the query `high speed laptop` will be expanded to:

```
(high AND speed AND (laptop OR notebook)) OR (performance
AND (laptop OR notebook))
```

Multiple-word thesaurus matches only apply when the words appear in exact sequence in the query. The queries `speed high laptop` and `high laptop speed` do not activate the expansion to `performance`.

For more details on thesaurus expansion, see *About the thesaurus feature on page 306*.

### Step 5: Stemming

Query terms, unless they are delimited with quotation marks to be treated as exact phrases, are expanded by the Dgraph using stemming. The expansion for stemming applies even to terms that are the result of thesaurus expansion. A stemmed query term is an `OR` expression of its word forms.

For example, if the query `pentium laptop` was thesaurus-expanded to:

```
(pentium OR intel) AND (laptop OR notebook)
```

it will be stemmed to:

```
(pentium OR intel) AND (laptop OR laptops OR notebook
OR notebooks)
```

assuming that only the improper nouns have plurals in the word form dictionary.

For more details on stemming, see *About the stemming feature on page 304*.

### Step 6: Primitive term and phrase lookup

Primitive term and phrase lookup is the lowest level of search processing. The Dgraph Server evaluates each search term as-is, and matches it to the set of documents containing that precise word or phrase (given the tokenization rules) in the data files being searched. Search is never case-sensitive, even for phrases.

### Step 7: Did You Mean

The Dgraph Server performs the "Did You Mean" processing as part of the record search processing. "Did You Mean?" processing is analogous to the spelling correction processing, only that the results are not included, but rather the spelling suggestions are returned.

For details on the "Did You Mean?" feature, see *Spelling Correction and Did You Mean on page 295*.

### Step 8: Navigation filtering

The Dgraph performs all filtering based on the navigation state after the search processing. This order is important, because it ensures that the spelling suggestions remain consistent as the navigation state changes.

### Step 9: EQL

The Endeca Query Language (EQL) builds on the core capabilities of Endeca Server to enable applications that examine aggregate information such as trends, statistics, analytical visualizations, comparisons, and so on, all within the Guided Navigation interface. If EQL is used, it is applied near the end of processing.

For more information about EQL, see the *Oracle Endeca Server EQL Guide*.

### Step 10: Relevance ranking

Relevance ranking is the last step in the processing for the record search. Each of the navigation-filtered search results is assigned a relevance score, and the results are sorted in descending order of relevance.

For details on this feature, see the section *Relevance Ranking on page 310*.

## Tips for troubleshooting record search

This topic includes tips for troubleshooting record search.

Due to the user-specified interaction of this feature (as opposed to the system-controlled interaction of Guided Navigation, in which Endeca Server controls the refinement values presented to the user), a user is allowed to submit a keyword search that does not match any records. Therefore, it is possible for a user to make a dead-end request with zero results when using record search. Applications utilizing record search need to account for this.

In production systems, these attributes are typically hard-coded at the application level, because the application requires specific search keys to be used for specific functionality.

If an attribute is not enabled for record searching but an application attempts to perform a record search against this attribute, Endeca Server successfully returns a null result set. The Dgraph process error log, however, outputs the following message: `In fulltext search: [Wed Sep 3 12:28:02 2010] [Warning] Invalid fulltext search key "Description" requested.`

The data domain configuration flag, `-v`, causes the Dgraph processes of the data domain to output detailed information about its record search configuration. To verify whether the data domain is recognizing a particular parameter, use the `-v` data domain flag in the data domain profile and check the output. You can specify this flag with the `endeca-cmd put-dd-profile --args -v` command.

Finally, record search can be enabled for standard attributes and for managed attribute values.

# Performance impact of record search

Each attribute enabled for record searching increases the size of the Dgraph index.

The specific size of the increase is related to the size of the unique word list generated by the specific attribute in the data set. Therefore, only attributes that are specifically needed by an application for record searching should be configured as such.

Chapter 21

# Search Interfaces

A search interface is a named collection of standard and managed attributes, each of which is enabled for record search.



## About search interfaces

A search interface allows you to control record search behavior for groups of one or more attributes.

A search interface may also contain:

- A number of attributes, such as name, cross-field information, and so on.
- An ordered collection of one or more ranking strategies.

Some of the features that can be specified for a search interface include:

- Relevance ranking
- Matching across multiple attributes
- Keyword in context results
- Partial match

You can use a search interface to control the behavior of search against a single standard or managed attribute, or to simultaneously search across multiple attributes.

For example, if a data set contains both an `Actor` standard attribute and `Director` managed attribute, a search interface can provide the user the ability to search for a person's name in both. A search interface's name is used just like a normal attribute when performing record searches. By default, a record search query on a search interface returns results that match any of the attributes in the interface.

## Implementing search interfaces

You implement search interfaces with Integrator ETL.

In Integrator ETL, you can use the **WebClient** component to send a request to Endeca Server using the Configuration Web Service. This request sends the `RECSEARCH_CONFIG` document to Endeca Server, thus creating a search interface. For information on how to configure a search interface using Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

If you are not using Integrator ETL, you can create a request with the `putConfigDocuments` operation of the Configuration Web Service and send the `RECSEARCH_CONFIG` XML document to Endeca Server. For information, see *Configuration Web Service operations on page 54*.

Before implementing search interfaces, make sure that all the attributes that are going to be included in a search interface have already been enabled for record search. In addition, if the search interface will include a relevance ranking strategy, make sure that the relevance ranking strategy has been configured.

If you are implementing wildcard search in a search interface, search interfaces can contain a mixture of wildcard-enabled and non-wildcard-enabled members (although only the former will return wildcard-expanded results).

You implement a search interface via the `RECSEARCH_CONFIG` XML configuration document. The resulting search interface should look similar to this example of a search interface named **AllFields** that uses a relevance ranking strategy named **All**:

```
<RECSEARCH_CONFIG>
  <SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="All" NAME="AllFields">
    <MEMBER_NAME RELEVANCE_RANK="4">ProductType</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="3">ProductName</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="2">SalesRegion</MEMBER_NAME>
    <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
  </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

# Options for allowing cross-field matches

The `CROSS_FIELD_BOUNDARY` attribute specifies when the Dgraph should try to match search queries across attribute boundaries.

The three settings for `CROSS_FIELD_BOUNDARY` are:

| Setting | Description |
|---|---|
| `ALWAYS` | The Dgraph always looks for matches across attribute boundaries, in addition to matches within an attribute. If you choose to use cross-field matching, the `ALWAYS` setting is recommended.<br>For example, in the `Sony camera` user query, if `CROSS_FIELD_BOUNDARY` is set to `ALWAYS`, the Dgraph returns all matches with `Brand` = `Sony` and `Product_Type` = `camera`. |
| `ON_FAILURE` | The Dgraph only tries to match queries across attribute boundaries if it fails to find any matches within a single attribute. Note that in most cases, the `ALWAYS` setting provides better results than the `ON_FAILURE` setting. |
| `NEVER` | The Dgraph does not look across boundaries for matches. This is the default. |

By default, record search queries using a search interface return the union of the results from the same record search query performed against each of the interface members.

For example, assume a search interface named MoviePeople that includes actor and director attributes. Searching for deniro against this interface returns the union of records that results from searching for deniro against the actor attribute and against the director attribute.

Less frequently, you may wish to allow a match to span multiple attributes. For example, in the same MoviePeople search interface, a query for clint eastwood returns records where either an actor standard attribute or a director attribute is assigned a value containing the words clint and eastwood. This behavior is useful for this query, where the search terms all relate to a single concept (the actor/director Clint Eastwood).

However, in some cases returning a union of the results from the same record search query performed against each search interface member is unnecessarily limiting. For example, in a home electronics catalog application, a customer searching for Sony camera might be interested in a broad range of products, but this record search would only return the few products that have the terms Sony and camera in the product name.

In such cases, you can use the CROSS_FIELD_BOUNDARY attribute when you create a search interface. This attribute specifies when the Dgraph should try to match search queries across attribute boundaries, but within the members of the search interface.

### How cross-field matches work in multi-assign cases

When a search interface member (that is, a searchable attribute) is multi-assigned on a record, the multi-assigns are treated by the Dgraph process of the Oracle Endeca Server as separate matches, just as if they were values from different attributes. A search that matches two or more terms in separate multi-assign values for the same attribute is treated as a cross-field match by the Dgraph.

For example, assume a record has the following attribute values:

```
P_Tag: Tom Brady
P_Tag: Jersey
```

A search against P_Tag for "tom brady jersey" is treated as a cross-field match, even though all results were found in the same attribute (P_Tag).

## Additional search interface options

You can configure other features for the search interface by specifying other match-related attributes to the SEARCH_INTERFACE element.

The following table lists the attributes (other than the CROSS_FIELD_BOUNDARY attribute) that you can specify with the SEARCH_INTERFACE element.

| Attribute | Purpose |
|---|---|
| DEFAULT_RELRANK_STRATEGY | For record search, assigns a default relevance scoring function to a search interface. |
| CROSS_FIELD_RELEVANCE_RANK | Specifies the relevance rank score for cross-field matches. The value should be an unsigned 32-bit integer.<br>The default value for CROSS_FIELD_RELEVANCE_RANK is 0. |

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

You can also use the PARTIAL_MATCH element to specify if partial query matches should be supported for the SEARCH_INTERFACE that contains this element.

Chapter 22

# Value Search

This section discusses how Endeca Server performs value search and how to configure it for your application.



## About value search

Value search allows users to perform keyword searches across attributes for values with matching names.

End users of applications powered by Endeca Server can search all types of attribute values, including values for standard and managed attributes. The front-end application can present these values to the end-user, allowing the user to select them and create a new navigation request.

Value search is enabled differently for attributes:

- Standard attributes. You can make standard attributes of type string value searchable. To configure a set of standard attributes of type string whose values will be considered for search, modify the values of the `IsPropertyValueSearchable` attribute on the PDRs.
- Managed attributes. All managed attributes are evaluated for value search by default, and you cannot disable value search for them.

## How value search works

***Value search*** returns single values that match the user's search terms, organized by attribute.

To be considered a valid result, a value must match all of the search terms that the user provides in the request.

### Example of value search

For example, a value search for `road` might return:

| Attribute | Values |
|---|---|
| Bikes | Road Bikes |
| Components | Road Frames, Road Gloves, Road Wheels |
| Reviews | Best all-around road bike |

# When to use value and record search

Value search is sometimes confused with record search. This topic provides examples of when to use each type of search.

Understanding the differences between the two basic types of keyword search (record search and value search) is important before creating a solution for a specific business problem. Use the following recommendations:

| Type of keyword search | When to use |
|---|---|
| Value search | In general, data sets with little descriptive text and extensive attribute values of type string that represent the most frequently searched terms (for example, autos) are a good fit for value search.<br>Keyword searches are usually suitable for such keywords as make, model, or year. These keywords are also likely candidates for being configured as managed attributes in your application. |
| Record search | Data sets with descriptive text or names (such as news articles) are better suited for record search. This is because a reasonable set of attribute values for such a data set cannot be expected to cover all the terms required to handle keyword search.<br>In such cases, text search allows an application to search directly against record text (such as the body of an article). |

For many applications, a combination of value search and record search is the best solution. In this case, separate value search and text search queries are executed simultaneously for the same keywords:

- If a value matches, the user is given the opportunity to select that value in place of the record search query to produce results.
- If no values match, the user is still left with the matching records for a record search query.

Keep in mind that navigation queries and value search queries are completely independent. In the scenario described above, where both queries are executed simultaneously, neither query affects the other. Record search is a variation of a navigation query. Record search could return results even though value search does not, and vice-versa.

Finally, keep in mind that both record search and value search can be run against named collections.

# Enabling value search

You enable a standard attribute for value search by changing the values in the `mdex-property-IsPropertyValueSearchable` attribute in the PDR.

Managed attributes are always enabled for value search in the Oracle Endeca Server. In addition, you can also enable standard attributes of type string for value search. In this case, these attributes are searched by the Oracle Endeca Server. Only the standard attributes of type string can be enabled for value search.

The `mdex-property-IsPropertyValueSearchable` attribute in the PDR specifies whether an attribute in your data set is value searchable. The valid settings for this attribute are:

- `true` means that the attribute is enabled for value search. This is the default.
- `false` means that the attribute is not enabled for value search.

If, in addition to enabling value search for specific attributes of type string, you also would like to enable wildcard search for all value search queries, set the `mdex-config_EnableValueSearchWildcard` attribute in the Global Configuration Record (GCR) to `true`.

To enable value search, you can send a request to change the attribute in the PDR using the Configuration Web Service, or you can use Integrator ETL.

For information on how to use the Configuration Web Service, see the section in this guide and the *Oracle Endeca Server API References* (which contains documentation for the WSDL).

For information on how to use Integrator ETL to enable a standard attribute for value search, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

Note that you can use the data domain `--search-char-limit` configuration flag to specify the maximum length (in characters) of a search term for value search. The default is 132 characters. Any term exceeding this length will not be indexed for any form of value search. Keep in mind that you specify this flag via the a `put-dd-profile` command.

# Utilizing value search in Studio

Value search supports the refinement search available in the **Available Refinements** component, and the ability to utilize typeahead search in the **Search Box** component.

For additional information on configuring Studio components that utilize value search, see the *Oracle Endeca Information Discovery Studio User's Guide*.

# Implementing value search with the API

This section provides examples of Conversation Web Service requests and responses, and describes parameters you can use for value search.

*Ordering results*

*Specifying relevance ranking strategy for results*

## Value search query format

To make a value search query, use a `ValueSearchConfig` type, specifying a `SearchTerm` element and, optionally, the attributes within which you would like to search.

A `ValueSearchConfig` type controls the behavior of a single value search query. The syntax for this type is:

```
<ValueSearchConfig Id="?" MaxPerProperty="?" RelevanceRankingStrategy="?" Mode="?" Language="?">
   <StateName>?</StateName>
   <SearchTerm>?</SearchTerm>
   <RestrictToProperties>
      <Property>?</Property>
   </RestrictToProperties>
</ValueSearchConfig>
```

The `SearchTerm` element specifies the search terms used by Endeca Server for a search either against all value-searchable attributes, or those that you specify in `RestrictToProperties`. You can optionally limit the number of search matches returned for each attribute using `MaxPerProperty`.

The `ValueSearchConfig` type has the following parameters (some of which are optional):

| Parameter | Description |
|---|---|
| `Id` | Optional. An identifier for this query configuration. |
| `MaxPerProperty` | Optional. Limits the number of matches returned per record attribute. If this attribute is omitted, all found matches for the record attribute are returned. |
| `RelevanceRankingStrategy` | Optional. Specifies a relevance ranking strategy to use on the results. If you omit this attribute and do not specify a relevance ranking strategy, the value for the strategy provided in the `DIMSEARCH_CONFIG` configuration document is used. If the document does not specify a strategy, the results are ranked using the following three strategies in this order (to break ties): `interp`, `exact`, and `static`. |
| `Mode` | Optional. Specifies a search mode, such as `Any`, or `AllPartial`. If `Mode` is not used, the query defaults to using the `All` search mode. |
| `Language` | Optional. Specifies a language ID for the search. Valid language IDs are listed in the topic *Supported languages on page 154*. |
| `RestrictToProperties` | Optional. If not specified, the request searches within all attributes. If specified, the request searches within specified attributes. |
| `SearchTerm` | Required. Contains the search term(s) (also known as keywords) used to conduct value search. The maximum number of characters in each search term is set by the `--search-char-limit` configuration flag (in the Endeca command `put-dd-profile`), which defaults to 132 characters. |

| Parameter | Description |
|---|---|
| StateName | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the StateName element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the StateName element is used to reference that named state (as the state will be used in any event in the RecordListConfig).<br>• If the request has an unnamed state, then the StateName element cannot be used. |

## Specifying a collection for value search

You can run a value search query against the records in a specific collection. To do so, specify the collection name in the <CollectionName> element of the State type, and then reference the name of the State in the StateName element of the ValueSearchConfig type.

## response

The results of a value search query are returned in a Results complex type, which includes the ValueSearch type. In the ValueSearch response, the following information is returned:

- The PropertyMatches element appears only for those standard and managed record attributes in which matches were found, and contains values for those matches.
- TotalValuesCount specifies the number of values returned for each value-searchable attribute.
- HasMore specifies whether there exist more attribute matches, beyond those that are returned. Because the request may limit the number of result values, the list of results returned may contain returned values and also indicate that a additional matching values exist that are not returned.

### Example of a value search query

The following example illustrates the format of a typical value search request in the Conversation Web Service:

```
<Request>
   <State>
     <Name>MyProductSearch</Name>
     <CollectionName>Products</CollectionName>
   </State>
   <ValueSearchConfig Id="ProdSearch" MaxPerProperty="5"
        RelevanceRankingStrategy="static (nbins,descending)" Mode="Any" Language="en">
      <StateName>MyProductSearch</StateName>
      <SearchTerm>aluminum</SearchTerm>
   </ValueSearchConfig>
</Request>
```

In this request, a search is conducted for the term envoy within the records of the Products collection. The number of requested results to return per attribute is set to 5 and English (en) is the language for the search.

In the example, the name of the `State` is MyProductSearch and it specifies the Products collection. In turn, the `ValueSearchConfig` type uses the `StateName` element to associate the MyProductSearch state to the config.

## Restricting value search to specific attributes

Value search queries could potentially contain many results.

You can use the `RestrictToProperties` attribute to limit the number of returned results to a list of one or more specified attributes. You can also use the `MaxPerProperty` attribute to help control the results returned from the corpus. Without these controls, the size of the resulting response from the Conversation Web Service could cause slow response times between your front-end application and Endeca Server.

If a managed attribute is searched, you can use `RestrictToProperties` to search within a whole managed attribute and its entire hierarchy of values, but you cannot restrict value search to a subtree within a particular root value in the hierarchy.

To restrict value search to searching specific attributes, use `RestrictToProperties`, as shown in this abbreviated example:

```
<ValueSearchConfig
   ...
   <RestrictToProperties>
      <Property>ProductCategory</Property>
      <Property>BikeRacks</Property>
   </RestrictToProperties>
</ValueSearchConfig>
```

## Limiting the number of results per attribute

Another way to limit value search results is to specify the number of values to return for each record attribute, using the `MaxPerProperty` element of `ValueSearchConfig`.

To set the number of attribute values to return for each attribute, use the `MaxPerProperty` attribute with an integer that specifies the number of values to return per attribute.

For example, the following query:

```
<ValueSearchConfig
    ...
    MaxPerProperty="2">
   <SearchTerm>Handlebars</SearchTerm>
   <RestrictToProperties>
   ...
</ValueSearchConfig>
```

returns 2 results for each attribute.

## Retrieving the number of matching results

The standard response to any value search request always includes information about the total number of matched values found, and whether all of them have been returned in this request. This information is returned in the `TotalValuesCount` and `HasMore` attributes on the `PropertyMatches` element.

A `PropertyMatches` element appears in the response only for those attributes in which matches were found, and contains attribute values for those matches. It contains two attributes that provide information on the number of values found and returned:

| Attribute | Description |
|---|---|
| `TotalValuesCount` | Specifies the total number of matched values found per property. |
| `HasMore` | Specifies whether any results were cut off because of a limit specified in the request with `MaxPerProperty`. |

In the response, the `Match` type element lists details of a single value within a particular attribute that matched a value search. Additionally, if the matched value belongs to a managed attribute, then the `FullPath` is also present in the response, as in the following abbreviated example of the response:

```
<cs:PropertyMatches Name="ProductType" DisplayName="Product Category"
    TotalValuesCount="1" HasMore="false">
 <cs:Match>
    <cs:MatchingValue DisplayName="Gloves">20</cs:MatchingValue>
    <cs:FullPath>
       <cs:DimensionValue DimensionName="ProductType" Spec="/">ProductType</cs:DimensionValue>
       <cs:DimensionValue DimensionName="ProductType" Spec="CAT_CLOTHING">Clothing<
/cs:DimensionValue>
       <cs:DimensionValue DimensionName="ProductType" Spec="20">Gloves</cs:DimensionValue>
    </cs:FullPath>
 </cs:Match>
</cs:PropertyMatches>
```

## Ordering results

Value search results consist of values grouped by record attribute. Attributes in the result list are returned in ascending alphabetical order.

The ordering of values, within each attribute, is as follows:

- If you specify a relevance ranking strategy, the order of results is ranked according to it.
- If you do not specify a relevance ranking strategy, the Dgraph uses the value for this strategy provided in the `DIMSEARCH_CONFIG` configuration document (you can send an updated version of this document to the Dgraph by using the Configuration Web Service).
- Further, if the document does not provide a strategy, the Dgraph ranks the results using the three strategies in this order to break ties: `interp`, `exact`, and `static(nbins,descending)`.

### Specifying relevance ranking strategy for results

To rank the order of results received in response for a value search request, you can use the `RelevanceRankingStrategy` attribute.

If you specify a relevance ranking strategy, the order of results is ranked according to it. To rank the order of results of the value search request, specify the value for the `RelevanceRankingStrategy` attribute in the `ValueSearchConfig` type of your Conversation Web Service request, as in this abbreviated example:

```
<ValueSearchConfig
   ...
   RelevanceRankingStrategy="static (nbins,descending)">
   ...
</ValueSearchConfig>
```

## Interaction of value search and wildcard search

By default, value search allows wildcards at the end of the search term (such as `gua*` for the search term `guarantee`).

To enable wildcards elsewhere in a search term, you must set the `mdex-config_EnableValueSearchWildcard` attribute in the Global Configuration Record (GCR) to `true`, for the standard attribute in your records.

The following examples illustrate how the Dgraph treats wildcards in value searches:

- A wildcard search at the end of the search term, such as `gua*`, is conducted by the Dgraph for all standard attributes for which value search is enabled.
- Wildcard searches of type `*uara` and `g*ara` are conducted by the Dgraph only if the GCR attribute `mdex-config_EnableValueSearchWildcard` is set to `true` for the corresponding standard attribute on your records. The default value for this attribute is `false`, meaning that wildcard search is disabled for value search.

## Performance impact of value search

This topic discusses value search and its impact on Endeca Server performance.

### Limit value search scope and the number of returned results

If you submit a value search query, the query is generally very fast. The runtime performance of value search directly corresponds to the number of values and the size of the resulting set of matching values. In general, this feature performs at a much higher number of operations per second than navigation requests. The most common performance problem is when the resulting set of values is exceptionally large (greater than 1,000), thus creating a large results page. To avoid it, limit the number of results per request, using value search parameters.

The query will be faster if you limit the scope and the number or results returned. You can do this using the options for configuring search configurations and type-ahead suggestions on the **Search Box** component in Studio.

## Decide which attributes to make value searchable

All managed attributes are always value searchable (you cannot toggle the value search setting for them). In addition, standard attributes of type string can be made value searchable. The `mdex-property_IsPropertyValueSearchable` attribute on the PDR controls whether the attribute in your record set is enabled for value search.

Before changing a value search setting for an attribute, examine your data to decide which of the attributes in your record set need to be value searchable. Next, turn off value search for attributes you will not be using for navigation, such as those standard attributes that contain long chunks of text.

Chapter 23

# Search Modes

By default, search operations return results that contain text matching all user search terms. In other words, search is conjunctive by default. However, in some cases a less restrictive matching is desirable, so that results are returned that contain fewer user search terms. This section describes the available search modes for record search and value search operations.



## List of valid search modes

The search mode can be specified independently for each record search operation contained in a navigation query, as well as for the value search query.

Valid search modes are the following:

| Search mode | Description |
|---|---|
| `All` | Match all user search terms (that is, perform a conjunctive search). This is the default mode. |
| `Partial` | Match some user search terms. |
| `Any` | Match at least one user search term. |
| `AllAny` | Match all user search terms if possible, otherwise match at least one. The `AllAny` search mode is not recommended in cases where queries can exceed two words. For example, a query on `womens small brown shoes` would return results on each of these four words and thus be essentially useless. In general, `AllPartial` is a better strategy. |
| `AllPartial` | Match all user search terms if possible, otherwise match some. Because you can configure this mode to match at least two or three words in a multi-word query, `AllPartial` is generally a better choice than `AllAny`. |
| `PartialMax` | Match a maximal subset of user search terms. |
| `Boolean` | Match using a Boolean query. |

## All mode

In `All` mode (the default mode), results must contain text matching each user search query term.

## Partial mode

In `Partial` mode, results must contain text matching at least a certain number of user search query terms, according to the rules listed in this topic.

In `Partial` mode, results must contain text matching search query terms, according to the following rules:

- The `MIN_WORDS_INCLUDED` setting specifies the minimum number of user query terms that each result must match. If there are not enough terms in the original query to satisfy this rule, then the entire query must match.
- The `MAX_WORDS_OMITTED` setting specifies the maximum number of user query terms that can be ignored in the user query. If `MAX_WORDS_OMITTED` value is set to zero, any number of words can be ignored.

You can specify both of these settings with the `PARTIAL_MATCH` element in a `SEARCH_INTERFACE` configuration.

In `Partial` mode, result sets always include all of the results that an `All` query have produced, and possibly additional results as well.

### Interaction of Partial mode and stop words

The presence of a stop word in a query reduces the minimum term count requirement for a document to match when `Partial` mode is used. Endeca Server treats stop words in a query as terms that match every document in the entire document set when counting how many terms must match a given query. Therefore, the presence of a stop word in a query reduces the minimum term count requirement for a document to match by one, the presence of two stop words reduces it by two, and so on. In practical terms, it means the result set may be both larger and more general than expected.

For example, consider a four-term query (such as `Medical Society of America`) against a search interface configured to allow `Partial` modes to require three terms to match. If one of those four terms (in this case `of`) is a stop word, only two of the other terms have to match, meaning results such as `Botanical Society of America` or `Medical Society Reunion` would be included in the set.

## AllPartial mode

In `AllPartial` mode, Endeca Server first uses `All` mode to return results matching all search terms, if any are available.

If no such `All` results are available, Endeca Server returns the results that `Partial` would have produced. This allows a more conservative matching policy than `Partial`, because high-quality conjunctive results are returned if they exist and `Partial` results are used as a fallback on conjunctive misses.

This behavior, however, can be affected if cross-field matches are applied to the search interface. A search that matches "any" or "partial" inside of the same field might be returned before a search that matches "all" of the terms but has to cross field boundaries to do so.

In addition, spelling correction can also alter the results. A search that matches any or partial spell-corrected terms in the same field may return before a non-spell-corrected search that matches all terms in different fields. To the user, this looks like there were no records matching all of the terms, even though there may be many that match cross-field.

**Note:** `AllPartial` is recommended for record search in a typical catalog application. The default configuration for `Partial`, which works well, can be adjusted to be more inclusive or conservative.

## Any mode

In `Any` mode, results need only match a single user search term.

An `Any` result set always includes all of the results that an `All` or `Partial` query have produced, and possibly additional results as well.

**Note:** The `Any` mode is not recommended for use with record search in typical catalog applications.

## AllAny mode

In `AllAny` mode, Endeca Server first uses `All` mode to return results matching all search terms, if any are available.

If no such `All` results are available, Endeca Server returns the results that `Any` would have produced.

**Note:** The `AllAny` mode is useful for value search.

## PartialMax mode

`PartialMax` mode is a variant of the `AllPartial` mode: `All` results are returned if they exist.

If no such `All` results exist, then results matching all but one term are returned; otherwise, results matching all but two terms are returned; and so forth.

`PartialMax` mode is subject to the `MIN_WORDS_INCLUDED` and `MAX_WORDS_OMITTED` settings used in the `Partial` mode. Hence, a `PartialMax` result set includes results if (and only if) the corresponding `Partial` result set includes results, and it contains a subset of the `Partial` results (possibly the entire set).

### Boolean mode

The Boolean search mode implements Boolean search, which allows users to specify complex expressions that describe the exact search criteria with which they would like to search.

## Configuring search modes in Studio

You configure search modes in the edit view of the **Search Box** component in Studio.

In addition, if you want to configure the minimum number of words for partial match modes and maximum number of words that may be omitted for partial match modes, you can specify these settings with the `PARTIAL_MATCH` element in a `SEARCH_INTERFACE` XML configuration element, which is part of `RECSEARCH_CONFIG`. For information, see *Recsearch_config elements on page 338*.

## Configuring search modes in the API

In the Conversation Web Service, the `SearchMode` simple type enumerates the search modes available when performing a text search.

You can specify a specific type of search mode as a value of the `Mode` attribute in the `TextSearchFilter` type of your request, and in the `ValueSearchConfig` type.

The following example uses the `AllPartial` search mode in a `TextSearchFilter`:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
      <State>
        <TextSearchFilter Key="MySearchInterface" RelevanceRankingStrategy="numfields"
          Mode="AllPartial" EnableSnippeting="false" Language="en">
            aluminum
        </TextSearchFilter>
      </State>
      <RecordListConfig Id="RecordList" MaxPages="30">
         <Column>ProductCategory</Column>
         <RecordsPerPage>5</RecordsPerPage>
      </RecordListConfig>
    </Request>
  </soap:Body>
</soap:Envelope>
```

Chapter 24

# Boolean Search

This section describes how to enable Boolean search for record search and value search.



## About Boolean search

The `Boolean` search mode implements Boolean search. It lets users specify complex expressions describing the exact search criteria for their searches.

Endeca Server search operations use the `All` mode by default, which results in conjunctive searches. However, users often want more precise control over their exact search query.

For example, consider the following request: `"Show me all records that match either red or blue and also match the word car."`

To express this request, the following query is required: (`red` OR `blue`) AND `car`. The OR in this query is a disjunctive operator that matches all records that are either `red` or `blue`. This set is then intersected with the set of results for the word `car` and the result of that operation is returned from the Oracle Endeca Server.

Unlike the `All` and `Any` modes, Boolean search also lets users specify negation in their queries.

For example, the query `camcorder AND NOT digital` will search for all records in the data domain that have the word `camcorder` and will then remove all records that have the word `digital` from that set before returning the result.

The set of Boolean operators implemented by the Oracle Endeca Server are:

- AND
- OR
- NOT
- NEAR, used for unordered proximity search
- ONEAR, used for ordered proximity search

In addition, you can use parentheses to create sub-expressions such as:

```
red AND NOT (blue OR green)
```

As with other search query modes, you can also run Boolean search queries against search interfaces; however, they may only be run against a single search interface.

Finally, the colon (:) character is a key restrict operator that you can use to limit a search to a single attribute regardless of whether or not these attributes are included in the same search interface.

# Boolean query syntax

The complete grammar for expressing Boolean queries, in a BNF-like format, is included in this topic.

The following sample code expresses Boolean queries, in a BNF-like format:

```
orexpr:      andexpr ;
               | andexpr OR orexpr ;
andexpr:      parenexpr ;
               | parenexpr andexpr ;
               | parenexpr AND andexpr ;
               | parenexpr andnotexpr ;
andnotexpr:      AND NOT orexpr ;
               | NOT orexpr ;
parenexpr:      LPAREN orexpr RPAREN ;
               | terms ;
terms:     word_or_phrase KEY_RESTRICT keyexpr ;
               | word_or_phrase NEAR/NUM word_or_phrase ;
               | word_or_phrase ONEAR/NUM word_or_phrase ;
               | multiple_word_or_phrase ;
multiple_word_or_phrase:      word_or_phrase ;
               | word_or_phrase multiple_word_or_phrase ;
keyexpr:     LPAREN nr_orexpr RPAREN ;
               | word_or_phrase ;
nr_orexpr:      nr_andexpr ;
               | nr_andexpr OR nr_orexpr ;
nr_andexpr:      nr_parenexpr ;
               | nr_parenexpr nr_andexpr ;
               | nr_parenexpr AND nr_andexpr ;
               | nr_parenexpr nr_andnotexpr ;
nr_andnotexpr:      AND NOT nr_orexpr ;
               | NOT nr_orexpr ;
nr_notexpr:      nr_parenexpr ;
               | NOT nr_parenexpr ;
nr_parenexpr:      LPAREN nr_orexpr RPAREN ;
               | nr_terms ;
nr_terms:     multiple_word_or_phrase ;
word_or_phrase:      word ;
               | phrase ;

AND:     '[Aa]' '[Nn]' '[Dd]' ;
OR:      '[Oo]' '[Rr]' ;
NOT:     '[Nn]' '[Oo]' '[Tt]' ;
```

```
NEAR:    '[Nn]' '[Ee]' '[Aa]' '[Rr]' ;
ONEAR:   '[Oo]' '[Nn]' '[Ee]' '[Aa]' '[Rr]' ;

NUM:     '[0-9] ;
               | NUM NUM ;
LPAREN:  '(' ;
RPAREN:  ')' ;
KEY_RESTRICT:    ':' ;
```

# Key restrict operator

This topic explains how to use the key restrict operator (`:`) in queries that contain Boolean search.

The colon (`:`) character is a key restrict operator that is used to limit a search to specified attributes, regardless of whether the attributes are included in the search interface.

For example, if you have two attributes (`Actor` and `Director`), you can issue a query that involves a Boolean expression consisting of both the `Actor` and `Director` attributes (for example, `"Search for records where the director was DeNiro and the actor does not include Pacino."`). The two attributes do not need to be included in the same search interface.

Users can successfully conduct a search on this using the following query, which will return the desired result:

```
Actor:Deniro AND NOT Director:Pacino
```

The key restrict feature is useful because it allows you to search for attributes that are outside of the search interface configuration.

The key restrict operator (:) binds only to the words or expressions adjacent to it. The resulting search is case-sensitive. The key restrict syntax is:

```
attribute:value
```

Note that there cannot be spaces between the attribute and colon, nor between the colon and the value.

To illustrate how the operator binds only to the words or expressions adjacent to it, consider this query:

```
car maker:aston martin
```

The query will search for the word "car" against the specified search interface, the word "aston" against the attribute named `maker`, and the word "martin" against the specified search interface.

If you intend to search for the phrase "aston martin" against the attribute named `maker`, then you would use double quotes for the phrase:

```
maker:"aston martin"
```

You can also use the conjunctive search format using parentheses:

```
maker:(aston martin)
```

This query does a conjunctive (All) search for the words "aston" and "martin" against the `maker` attribute.

# About proximity search

The proximity operators, NEAR and ONEAR, allow users to search for a pair of terms that must occur within a given distance from each other in a document.

The document is matched if both terms are present in the document, and if the terms are within the specified number of words from each other.

Wildcards are not supported in term specifications.

The syntax for using the proximity operators is as follows:

```
term1 NEAR/num term2
term1 ONEAR/num term2
```

In this example:

- Each term (`term1` and `term2`) can be a single word or a multi-word phrase (which must be specified within quotation marks).
- The `num` parameter is an integer that specifies the maximum number of words between the two terms. That is, if `num` is 5, then `term1` and `term2` can be separated by no more than five words.

*Example of using NEAR for unordered matching*

*Example of using ONEAR for ordered matching*

## Example of using NEAR for unordered matching

Use the NEAR operator for unordered proximity searches.

That is, `term1` can appear within `num` words before or after `term2` in the document.

For example, if a user specifies:

```
"Mark Twain" NEAR/8 Hartford
```

Then both of these sentences will be considered matches:

```
"Mark Twain wrote some of his best books in Hartford."
 "Tour the Hartford, Connecticut home where Mark Twain lived
 and worked from 1874 to 1891."
```

Phrases are treated as one word. In the first sentence, for example, the software starts counting with the word `"wrote"` (not `"Twain"`).

## Example of using ONEAR for ordered matching

Use the ONEAR operator for ordered proximity searches.

`term1` must appear within `num` words before `term2` in the document.

For example, if a user specifies:

```
"Mark Twain" ONEAR/8 Hartford
```

The following sentence would not be considered a match:

```
"Tour the Hartford,
Connecticut home where Mark Twain lived and
```

```
 worked from 1874 to 1891."
```

It is not a match because the word "Hartford" must appear after the phrase "Mark Twain" in the text (assuming that the next eight words are not "Hartford").

# Proximity operators and nested sub-expressions

This topic contains examples of using proximity operators with nested sub-expressions.

Using the two proximity operators as sub-expressions to the other Boolean operators is supported. For example, the expression:

```
(chardonnay NEAR/5 California) AND Sonoma
```

is a valid expression because NEAR is being used as a sub-expression to the AND operator.

However, you cannot use the non-proximity operators (AND, OR, NOT) as sub-expressions to the NEAR and ONEAR operators.

For example, the following is not a valid expression:

```
(chardonnay OR merlot) NEAR/5 California
```

This invalid expression, however, could be specified as:

```
(chardonnay NEAR/5 California) OR (merlot NEAR/5 California)
```

The proximity operators are therefore leaf operators. That is, they accept only words and phrases as sub-expressions, but not the other Boolean operators.

Using proximity operators with the key restrict operator also has the same limitations when used as sub-expressions.

For example, the following query is not valid:

```
("car maker" : aston) NEAR/3 martin
```

However, the following format for a key restrict operator is acceptable:

```
"car maker" : (aston NEAR/3 martin)
```

# Boolean query semantics

This topic discusses the meaning of AND, OR, AND NOT, and other operators allowed in Boolean search queries.

The following statements describe semantics of Boolean query operators:

- The AND operator executes an intersection of its two operands.
- The OR operator executes a union of the two operands.
- The AND NOT operator executes a set subtract, subtracting the second operand from the first.
- The parentheses operators have two meanings, depending on their usage:
  - They can be used to group sub-expressions, as in "(red or blue) and car"
  - Or, they can be used as AND operators in themselves.

For example, the query `"(red or blue) car"` automatically treats the `")"` as a `") AND"`. Thus the query would be treated as `"(red or blue) and car"`.

The same is true for usage of the left parenthesis.

- Words or phrases grouped together without any explicit operators (such as `"red car or blue bicycle"`) are also queried conjunctively.

  Thus the example query would return the results for `"(red and car) or (blue and bicycle)"`. Similarly, `"red car" "blue bicycle"` will return the results for `"red car" AND "blue bicycle"`.

- As the examples demonstrate, operator names are not case sensitive, although field names are.

# Operator precedence

The `NOT` operator has the highest precedence, followed by the `AND` operator, followed by the `OR` operator. You can always control the precedence by using parentheses.

For example, the expression `"A OR B AND C NOT D"` is interpreted as `"A OR (B AND C AND (NOT D))"`.

# Interaction of Boolean search with other features

The following table describes whether various features are supported for queries that execute a Boolean search (including the proximity operators).

| Feature | Support with Boolean search | Comments |
|---|---|---|
| Stemming | Yes | |
| Thesaurus matching | No | |
| Misspelling correction | No | Auto-correct and “Did you mean?” are not supported. |
| Relevance ranking | No | |
| Wildcard search | Yes for the `AND`, `OR`, and `NOT` operators. | Proximity operators do not support wildcards. |
| Stop words | No | Stop words are treated as normal words and are not filtered from queries. |
| Phrase search | Yes | |

# Error messages for Boolean search

Syntactically invalid queries generate error messages described in this topic.

| Sample query | Error message | Comments |
|---|---|---|
| `NOT sony` | `Top-level negation is not allowed.` | The final result set is not allowed to be the result of a negation operation. |
| `(` | `Unexpected end of expression.` | |
| `Sony OR NOT Aiwa` | `The <first \| second> clause of the OR at position <position> is a negation. Neither clause of an OR expression may be a negation.` | Neither clause of an OR expression can be the result of a negation operation. |
| `Sony OR` | `Unexpected end of expression.` | |
| `Sony AND` | `Unexpected end of expression.` | |
| `Sony NOT` | `Unexpected end of expression. Expecting an opening left parenthesis, a word, or a phrase.` | |
| `(Sony` | `Unexpected end of expression. Expecting closing right parenthesis.` | |
| `Manufacturer:(Sony OR Item: Camera)` | `The key restrict operator may not be used within another key restrict expression.` | |
| `Manufacturer:` | `Unexpected end of expression. The key restrict operator must be followed by a word, a phrase, or a left parenthesis.` | |
| `Manufacturer:OR` | `The key restrict operator must be followed by a word, a phrase, or a left parenthesis.` | |
| `Foo:Sony` | `Unknown search index name "Foo" used for restrict operator` | The name must exactly match the name used in the data. |
| `Sony AND OR Aiwa` | `Expecting a term or phrase.` | Repeated operators are an error. |

# Implementing Boolean search in Studio

You configure Boolean search in Studio, in the edit view of the **Search Box** component.

When you set up the attributes to search on in the **Search Box** component, you also set a match mode that should be used for search. To use Boolean search for the search mode, you set the match mode to Boolean.

Attributes should be configured appropriately for record search and/or value search.

# Implementing Boolean search with the API

Using requests to the Conversation Web Service, you can specify a Boolean search mode for any search request that performs value or attribute search, or a search request against defined search interfaces. You can also use Boolean search in record and attribute filters. This topic includes examples of these requests.

Before using search on any attributes, ensure that attributes are configured for either record search and/or managed attribute value search. For information, see *Enabling value search on page 259*.

### Boolean search in value search

In this example, Boolean search mode is used for a value search made with the `ValueSearchConfig` type:

```
<ValueSearchConfig Id="ValSearch" MaxPerProperty="5" Mode="Boolean" Language="en">
   <StateName>MySearch</StateName>
   <SearchTerm>"Bike Racks" AND "Handlebars"</SearchTerm>
</ValueSearchConfig>
```

### Boolean search in attribute filters

When you set up the attributes for which to search using the `TextSearchFilter` type, you also set a match mode that should be used for search. To use boolean search for the search mode, you set the match mode to Boolean. The following example illustrates the `TextSearchFilter` that uses Boolean search:

```
<State>
   <Name>MyRecSearch</Name>
   <TextSearchFilter Key="Description" Mode="Boolean" EnableSnippeting="false"
     Language="en">"peach" AND "apple"</TextSearchFilter>
</State>
```

### Boolean search in search interfaces

Before you use Boolean search against search interfaces, you need to configure one or more search interfaces that include all of the attributes that you want to search. This is done through the `putConfigDocuments` operation of the Configuration Web Service, by sending in an XML configuration document `RECSEARCH_CONFIG`. For information on how to send XML configuration documents to the Oracle Endeca Server, see *Loading configuration documents on page 60*.

Now you can create a single Boolean search request against the defined search interfaces:

```
<State>
   <Name>MyRecSearch</Name>
   <TextSearchFilter Key="AllSales" Mode="Boolean" EnableSnippeting="false"
     Language="en">English : one AND Spanish : dos</TextSearchFilter>
</State>
```

# Troubleshooting Boolean search

If you encounter unexpected behavior while using Boolean search, use the data domain configuration flag, `-v`, when starting the data domain in the Oracle Endeca Server. This flag prints detailed output, including standard errors, describing its execution of the Boolean query. You can specify it as `endeca-cmd put--dd-profile --args -v`.

# Performance impact of Boolean search

The performance of Boolean search is a function of the number of records associated with each term in the query, and also the number of terms and operators in the query.

As the number of records increases, and as the number of terms and operators increase, queries become more expensive.

The performance of proximity searches is as follows:

- Searches using the proximity operators are slower than searches using the other Boolean operators.
- Proximity searches that operate on phrases are slower than other proximity searches and slower than normal phrase searches.
- Searches using the `NEAR` operator are about twice as slow as searches using the `ONEAR` operator, because word positioning must be calculated forwards and backwards from the target term.

Chapter 25

# Phrase Search

Phrase search allows users to specify a literal string to be searched.



## About phrase search

Phrase search allows users to enter queries for text matching of an ordered sequence of one or more specific words.

By default, an Endeca Server search query matches any text containing all of the search terms entered by the user. Order and location of the search words in the matching text is not considered. For example, a search for `John Smith` returns matches against text containing the string `John Smith` and also against text containing the string `Jane Smith` and `John Doe`.

In some cases, the user may want location and order to be considered when matching searches. If one were searching for documents written by `John Smith`, one would want hits containing the text `John Smith` in the author field, but not results containing `Jane Smith` and `John Doe`.

Phrase search allows the user to put double-quote characters around the search term, thus specifying a literal string to be searched. Results of a phrase search contain all of the words specified in the user's search (not stemming, spelling, or thesaurus equivalents) in the exact order specified.

For example, if the user enters the phrase query `"run fast"`, the search finds text containing the string `run fast`, but not text containing strings such as `fast run`, `run very fast`, or `running fast`, which might be returned by a normal non-phrase query.

Additionally, phrase search queries do not ignore stop words. For example, if the word `the` is configured as a stop word, a phrase search for `"the car"` does not return results containing simply `car` (not preceded by `the`).

Also, phrase search enables stop words to be disabled. For example, if `the` is a stop word, a phrase search for `"the"` can retrieve text containing the word `the`.

Because phrase searches only consider exact matches for contained words, phrase search also provides a means to return only true matches for a particular word, avoiding matches due to features such as stemming, thesaurus, and spelling.

For example, a normal search for the word `corkscrew` might also return results containing the text `corkscrews` or `wine opener`. Performing a phrase search for the word `"corkscrew"` only returns results containing the word `corkscrew` verbatim.

# About positional indexing

To enable faster phrase search performance and faster relevance ranking with the Phrase module, your project builds data files out of word positions. This process is called positional indexing.

The Oracle Endeca Server creates a set of positional data files for standard and managed attribute values.

Phrase search is automatically enabled in Endeca Server at all times. For phrase search query processing, Endeca Server examines potential matching text to verify the presence of the requested phrase query string. This examination process can be slow in the following cases:

- The amount of text data is large (perhaps containing attribute values representing lengthy descriptions)
- The text that is being processed is offline (in the case of document text)

Endeca Server uses positional data files to improve performance in these scenarios. Positional indexing improves performance of multi-word phrase search, proximity search, and certain relevance ranking modules. The thesaurus uses phrase search, so positional indexing improves performance of multi-word thesaurus expansions as well. Positional indexing is enabled by default for attributes in your data domain.

# Handling of punctuation in phrase search

Unless they are included as special characters, all punctuation characters are stripped out, during both indexing and query processing. When punctuation is stripped out during query processing, the previously connected terms have to remain in their original order.

# Examples of phrase search queries

You request phrase matching in the Conversation Web Service requests by enclosing a set of one or more search terms in quotation marks.

You can include phrase search queries in either record search or value search operations. You can combine phrase search with non-phrase search terms or other phrase terms.

The following examples illustrate a phrase search query.

A record search for a phrase `"Mountain Bikes"` is as follows:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
      <State>
        <TextSearchFilter Key="MySearchInterface" RelevanceRankingStrategy="numfields"
          Mode="AllPartial" EnableSnippeting="false" Language="en">
          "Mountain Bikes"</TextSearchFilter>
      </State>
      <RecordListConfig Id="RecordList" MaxPages="30">
         <Column>ProductCategory</Column>
         <RecordsPerPage>5</RecordsPerPage>
      </RecordListConfig>
    </Request>
  </soap:Body>
</soap:Envelope>
```

A value search for values containing the phrase `"Touring Bikes"` is similar to the following abbreviated example:

```
<Request>
```

```
   <State>
     <Name>MyProductSearch</Name>
     <CollectionName>Products</CollectionName>
   </State>
   <ValueSearchConfig Id="ProdSearch" MaxPerProperty="5"
        RelevanceRankingStrategy="static (nbins,descending)" Mode="Any" Language="en">
      <StateName>MyProductSearch</StateName>
      <SearchTerm>"Touring Bikes"</SearchTerm>
      <RestrictToProperties>
        <Property>ProductCategory</Property>
      </RestrictToProperties>
   </ValueSearchConfig>
</Request>
```

# Performance impact of phrase search

Phrase search queries are generally more expensive to process than normal conjunctive search queries.

In addition to the work associated with a conjunctive query, a phrase search operation must verify the presence of the exact requested phrase.

The cost of phrase search operations depends mostly on how frequently the query words appear in the data. Searches for phrases containing relatively infrequent words (such as proper names) are generally very rapid, because the base conjunctive search narrows the results to a small set of candidate hits, and within these hits relatively few possible match positions need to be considered.

On the other hand, searches for phrases containing only very common words are more expensive. For example, consider a search for the phrase `"to be or not to be"` on a large collection of documents. Because all of these words are quite common, the base conjunctive search does not narrow the set of candidate hit documents significantly. Then, within each candidate result document, numerous possible word positions need to be scanned, because these words tend to be frequently reused within a single document.

Even very difficult queries (such as `"to be or not to be"`) are handled by Endeca Server within a few seconds (depending on hardware), and possibly faster on moderate sized data sets. If such queries are expected to be very common, adequate hardware must be employed to ensure sufficient throughput. In most applications, phrase searches tend to be used far less frequently than normal searches. Also, most phrase searches performed tend to contain at least one information-rich, low-frequency word, allowing results to be returned rapidly (that is, in less than a second).

Chapter 26

# Snippeting in Record Searches

Snippeting provides the ability to return an excerpt from a record in context, as a result of a user query.



## About snippeting

The snippeting feature provides the ability to return an excerpt from a record — called a snippet — to an application user who performs a record search query.

A snippet contains the search terms that the user provided, along with a portion of the term's surrounding content to provide context. With the added context, users can more quickly choose the individual records they are interested in.

A snippet can be based on the term itself or on any thesaurus or spell-correction equivalents. At least one instance of a term or equivalent is highlighted per snippet, regardless of the number of times the term or its equivalents appear in the snippet. A thesaurus or spell-corrected alternative may be highlighted instead of the term itself, even if both appear within the snippet.

You enable snippeting on individual members (fields) in a search interface that typically have many lines of content. For example, fields such as Description, Abstract, DocumentBody, and so on are good candidates to provide snippeting results. You can also enable snippeting on a per-query basis.

The result of a query with snippeting enabled contains at least one snippet in which enough terms are highlighted to satisfy the user's query. That is, if it is an AND query, the result contains at least one of each term, and if it is an OR query, it contains at least one of the alternatives.

In Studio, only the **Data Explorer**, **Results List**, and **Results Table** components support snippeting. On these components, for attributes that support snippeting, when users perform a search, the search snippet is displayed.

## Snippet formatting and size

A snippet consists of search terms, surrounding context words, and ellipses.

A snippet can contain any number of search terms bracketed by `SnippetTerm` tags. The tags call out search terms and allow you to more easily reformat the terms for display in your front-end application.

The snippet size is the total number of search terms and surrounding context words. Although you can configure the total number of words in a snippet In order to adhere to the size setting for a snippet, it is possible that Endeca Server may omit some search terms and context words from a snippet. This situation becomes more likely if an application user provides a large number of search terms and the maximum snippet size is comparatively small.

A snippet consists of one or more segments. If there are multiple segments, they are delimited by ellipses in between them. Ellipses (`...`) indicate that there is text omitted from the snippet occurring before or after the ellipses.

### Example of a snippet

For example, here is a snippet made up of two segments with a maximum size set at 20 words. The snippet resulted from a search for the search terms, `Scotland` and `British`, which are enclosed within `<SnippetTerm>` tags.

```
<SearchSnippet>
   <SnippetText>...in Edinburgh </SnippetText>
   <SnippetTerm>Scotland</SnippetTerm>
   <SnippetText>, and has been employed by Ford for 25 years...He first joined Ford's
   </SnippetText>
   <SnippetTerm>British</SnippetTerm>
   <SnippetText> operation. Mazda motor...</SnippetText>
</SearchSnippet>
```

# Enabling snippeting

You enable snippeting globally for Endeca Server via the `RECSEARCH_CONFIG` configuration document.

Endeca Server has several configuration documents that configure some features. You can edit them using the format specified in the *Dgraph Configuration Reference* appendix in this guide. After these documents are edited, you can send them to Endeca Server using the Configuration Web Service or Integrator ETL.

The `RECSEARCH_CONFIG` document allows inclusion of `SEARCH_INTERFACE`, which in turn lets you specify the snippet size for each of its members. The following example shows the syntax:

```
<RECSEARCH_CONFIG>
   <SEARCH_INTERFACE NAME="MySearch">
     <MEMBER_NAME SNIPPET_SIZE="12">Description</MEMBER_NAME>
   </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

The presence of the `SNIPPET_SIZE` attribute enables snippeting for the `MEMBER_NAME` attribute, and the value of `SNIPPET_SIZE` specifies the maximum number of words a snippet can contain. Omitting this attribute or setting its value to zero disables snippeting.

Each member of a search interface is enabled and configured separately. In other words, snippeting results are enabled and configured for each member of a search interface and not for all members of a single search interface.

**Note:** A search interface member is an attribute that has been enabled for search and that has been added to the `SEARCH_INTERFACE` element.

You can enable and configure any number of individual search interface members. Each member that you enable produces its own snippet. Enabling a member in one search interface does not affect that member if it

appears in other search interfaces. For example, enabling the Description attribute for Search Interface A does not affect the Description attribute in Search Interface B.

To enable snippeting:

1. In any text editor, edit the `RECSEARCH_CONFIG` document, similar to the following example:

```
<RECSEARCH_CONFIG>
  <SEARCH_INTERFACE NAME="MySearch">
    <MEMBER_NAME SNIPPET_SIZE="10">Description</MEMBER_NAME>
  </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

   In this example, the snippet size is set to 10 for an attribute "Description".

2. Use the Configuration Web Service or Integrator ETL to send the `RECSEARCH_CONFIG` document to the data domain in Endeca Server.

   For information on the Configuration Web Service, see the section in this guide, or the *Oracle Endeca API References*. For information on Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

# Tuning tips for snippeting

Enabling snippeting affects query runtime performance.

You can minimize the performance impact on query runtime by limiting the number of words in an attribute that Endeca Server evaluates to identify the snippet. This approach is especially useful in cases where a snippet-enabled attribute stores large amounts of text.

Provide the data domain configuration flag, `--snippet-cutoff`, to the data domain profile, to restrict the number of words that Endeca Server evaluates in an attribute. For example, `endeca-cmd put-dd-profile --snippet-cutoff 300` evaluates the first 300 words of the attribute to identify the snippet.

If the `--snippet-cutoff` flag is not specified in the data domain profile, or is specified without a value, the snippeting feature defaults to a cutoff value of 500 words.

If a snippet is too short, and you are not seeing enough context words in it, increase the value for `SNIPPET_SIZE` in the configuration document. See *Enabling snippeting on page 283*, for the detailed format of the configuration.

# Retrieving snippets per query with the API

You can enable snippets for a particular attribute on a per-query basis using the `TextSearchFilter` type in the Conversation Web Service.

To request snippets with the Conversation Web Service, use the `TextSearchFilter` type with the specified search interface or text-search enabled attribute. If you specify a search interface, it will be one for which snippeting is enabled for its members in the `RECSEARCH_CONFIG` XML document. The Conversation Web Service returns snippets as part of the `RecordListEntry` element (which also returns the records themselves).

Setting the `EnableSnippeting` attribute to `true` in the `TextSearchFilter` type enables snippeting per query, for the specified attribute or search interface. The `SnippetLength` attribute sets the length of the snippet; the search term specifies the snippet term:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
      <State>
        <TextSearchFilter Key="Description" Mode="AllPartial"
          EnableSnippeting="true" SnippetLength="4" Language="en">
            gearing
        </TextSearchFilter>
      </State>
      <RecordListConfig Id="RecordList" MaxPages="30">
         <Column>Description</Column>
         <RecordsPerPage>5</RecordsPerPage>
      </RecordListConfig>
    </Request>
  </soap:Body>
</soap:Envelope>
```

**Note:** Use these settings only if you need to specify snippeting information for a single attribute for which there is no search interface configured. These settings do not override the settings that globally enable snippeting for members of the search interface in the `RECSEARCH_CONFIG` > `SEARCH_INTERFACE` elements.

## Response

The following response from the Conversation Web Service returns snippeting information as part of the `RecordListEntry`:

```
<cs:ContentElement xsi:type="cs:RecordList" Id="RecordList"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<cs:NumRecords>61157</cs:NumRecords>
<cs:TotalPages>1224</cs:TotalPages>
<cs:RecordRange First="751" Last="800"/>
    ...
<cs:RecordListEntry>
  <cs:Record>
    <Description type="mdex:string">A true multi-sport bike that offers
      streamlined riding and a revolutionary design. Aerodynamic design lets you
      ride with the pros, and the gearing will conquer hilly roads.
    </Description>
    <FactSales_RecordSpec type="mdex:string">SO44563-19</FactSales_RecordSpec>
  </cs:Record>
  <cs:ComputedProperties>
    <cs:SearchSnippets Key="Description">
       <cs:SearchSnippet>
         <cs:SnippetText>...and the gearing will conquer hilly </cs:SnippetText>
         <cs:SnippetTerm>gearing<cs:SnippetTerm>
         <cs:SnippetText> the gearing will conquer...</cs:SnippetText>
       </cs:SearchSnippet>
     </cs:SearchSnippets>
   </cs:ComputedProperties>
</cs:RecordListEntry>
```

Chapter 27

# Wildcard Search

Wildcard search allows users to match query terms to fragments of words in text processed by an Endeca data domain.



## About wildcard search

Wildcard search is the ability to match user query terms to fragments of words in indexed text.

Normally, Endeca Server search operations (such as record search and value search) match user query terms to entire words in the indexed text. For example, searching for the word `run` only returns results containing the specific word `run`. Text containing `run` as a substring of larger words (such as `running` or `overrun`) does not result in matches.

With wildcard search enabled, the user can enter queries containing the special asterisk or star operator (`*`). The asterisk operator matches any string of zero or more characters. Users can enter a search term such as:

```
*run*
```

This will match any text containing the string `run`, even if it occurs in the middle of a larger word such as `brunt`.

Wildcard search is useful for performing text search on data fields such as part numbers, ISBNs, and SKUs. Unlike cases where search is performed against normal linguistic text, in searches against data fields it may be convenient or even necessary for the user to enter partial string values. Details on how data fields that include punctuation characters are processed are provided in this section.

For example, suppose users were searching a database of integrated circuits for Intel `486 CPU` chips. The database might contain records with part numbers such as `80486SX` and `80486DX`, because these are the full part numbers specified by the manufacturer. But to end users, these chips are known by the more generic number `486`. In such cases, wildcard search is a natural feature to bridge the gap between user terminology and the source data.

**Note:** To optimize performance, the Dgraph process performs wildcard indexing for words that are shorter than 1024 characters. Words that are longer than 1024 characters are not indexed for wildcard search.

# Interaction of wildcard search with other features

The table in this topic describes whether various features are supported for queries that execute a wildcard search.

| Feature | Support with wildcard search | Comments |
|---|---|---|
| Stemming | No | |
| Thesaurus matching | No | |
| Misspelling correction | No | Auto-correct and “Did You Mean?” are not supported. |
| Relevance ranking | Yes | |
| Snippeting | No | |
| Phrase search | No | |
| Why Did It Match | Yes | |
| Word interp | Yes | |

# Ways to configure wildcard search

To send the configuration for wildcard search to Endeca Server, you can use the Configuration Web Service directly or use Integrator ETL.

For information on how to load the schema using the Configuration Web Service, see the section in this guide.

For information on how to configure wildcard search in record search for attributes, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

*Configuring wildcard search in record search*

*Configuring wildcard search in value search*

*Configuring wildcard search for a search interface*

## Configuring wildcard search in record search

You make an attribute wildcard searchable in record searches by changing the value of the `mdex-property_TextSearchAllowsWildcards` attribute in the PDR, using either a request to the Configuration

Web Service for loading the schema, or by sending this configuration to Endeca Server through a connector in Integrator ETL.

The `mdex-property_TextSearchAllowsWildcards` attribute of a PDR enables wildcard searches in record search against the attribute. The valid settings for this attribute are:

- If set to `true`, an attribute is wildcard searchable during record searches.
- If set to `false`, an attribute is not wildcard searchable during record searches. The default is `false`.

Note that an attribute must be record searchable in order for it to allow wildcard search in record searches. This means that before you set `mdex-property_TextSearchAllowsWildcards` to `true`, make sure that `mdex-property_IsTextSearchable` is set to `true`.

**Note:** Enabling wildcard search for an attribute that contains large portions of text (either via numerous or large attribute values with text content) can take a long time to process. Before making this change, examine your data to decide which of the attributes in your record set need to be wildcard searchable. Also, turn off wildcard search in record searches for those attributes on which it won't be used by the users of your front-end application.

#### Example: enabling wildcard search for an attribute

For example, the following web service request to the Configuration Web Service enables wildcard search for the attribute `VehicleModel`:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <config:configTransaction
     xmlns:config="http://www.endeca.com/MDEX/config/services/types/3/0"
     xmlns:mdex="http://www.endeca.com/MDEX/config/XQuery/2009/09">
     <config:updateProperties>
       <mdex:record>
        <mdex-property_Key>VehicleModel</mdex-property_Key>
        <mdex-property_TextSearchAllowsWildcards>true</mdex-property_TextSearchAllowsWildcards>
       </mdex:record>
     </config:updateProperties>
    </config:configTransaction>
  </soap:Body>
</soap:Envelope>
```

## Configuring wildcard search in value search

You configure wildcard search during value searches in the Global Configuration Record (GCR), using either a request to the Configuration Web Service or Integrator ETL.

Unlike the option for enabling wildcard search in text search, which is performed by editing each attribute in its PDR (which affects only a single attribute), the GCR globally affects the enablement of wildcard search in value search for all attributes.

The `mdex-config_EnableValueSearchWildcard` attribute in the GCR specifies whether wildcard search should be enabled or disabled for value search across all attributes in the Endeca Server data domain. The valid settings for this attribute are:

- If set to `true`, wildcards are supported for value search.
- If set to `false`, wildcards are not supported for value search. The default is `false`.

If you change this setting for the GCR on a data set with a large number of existing records, the operation can take a long time to process, due to re-indexing. Thus, it is recommended to change this setting in the data

domain before you load records (after evaluating whether your application requires wildcard search on value search for all attributes).

Wildcard queries at the end of the search term (for example, `gua*` for the search term `guarantee`) are always enabled even if wildcard search is disabled for value search for the attribute.

## Configuring wildcard search for a search interface

You can enable wildcard matching for a search interface by adding one or more wildcard-enabled attributes to the search interface.

First, add the desired attributes. Wildcard search can be partially enabled for a search interface. That is, some members of the search interface are wildcard-enabled while the others are not.

Searches against a partially wildcard-enabled search interface follow these rules:

- The search results from a given member follow the rules of its configuration. That is, results from a wildcard-enabled member follow the rules of wildcard search, while results from non-wildcard members follow the rules for non-wildcard searches.
- The final result is a union of the results of all the members (whether or not they are wildcard-enabled).

You should keep these rules in mind when analyzing search results. For example, assume that in a partially wildcard-enabled search interface, `Property-W` is wildcard-enabled while `Property-X` is not. In addition, the asterisk (`*`) is not configured as a search character. A record search issued for `woo*` against that search interface may return the following results:

- `Property-W` returns records with `woo`, `wood`, and `wool`.
- `Property-X` only returns records with `woo`, because the query against this attribute treats the asterisk as a word break. However, it does not return records with `wool` and `wood`, even though records with those words exist.

However, because the returned record set is a union, the user will see all the records. A possible source of confusion might be that if snippeting is enabled, the records from `Property-X` will not have `wood` and `wool` highlighted (if they exist), while the records from `Property-W` will have all the search terms highlighted.

To enable wildcard search in a search interface:

1. Enable wildcard search in text search for members of the search interface. (This is controlled by the `mdex-property_TextSearchAllowsWildcards` attribute on the PDR, for each attribute member of the search interface).

   Wildcard search in text search can be partially enabled for a search interface. That is, some members of the search interface can be enabled for wildcard search in text search, while the others are not.
2. Add the desired attributes to the search interface in the `RECSEARCH_CONFIG` document. For the structure of this document, see the appendix in this guide.
3. Use the Configuration Web Service or Integrator ETL to send this document to Endeca Server. For information, see the section about the Configuration Web Service in this guide, or the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.

# Dgraph flags for wildcard search

There are no Dgraph flags required to enable wildcard search. If wildcard is enabled in record search for an attribute, and is also enabled for value search, the Dgraph process automatically enables the use of the asterisk operator (*) in appropriate search queries.

The following considerations apply to wildcard search queries that contain punctuation, such as `abc*.d*f`:

The Dgraph process rejects and does not process queries that contain only wildcard characters and punctuation or spaces, such as `*.`, `* *`. Queries with wildcards only are also rejected.

The maximum number of matching terms for a wildcard expression is 100 by default. You can modify this value with the data domain configuration flag, `--wildcard-max`.

In case of wildcard search with punctuation, you may want to increase `--wildcard-max`, if you would like to increase the number of returned matched results. You can specify the value for this flag when creating a data domain profile: `endeca-cmd --put-dd-profile --wildcard-max 110`.

# Using wildcard search in Studio

No specific Studio development is required to use wildcard search.

If wildcard search is enabled for record search and value search, users can use the **Search Box** component to enter search queries containing asterisk operators to request partial matching. If wildcard search is enabled for value search, type-ahead suggestions can be used in the **Search Box**.

Whereas the simplest use of wildcard search requires users to explicitly include asterisk operators in their search queries, some applications automate the inclusion of asterisk operators as a convenience, or control the use of asterisk operators using higher-level interface elements.

For example, an application might render a radio button next to the search box with options to select Whole-word Match or Substring Match. In Substring Match mode, the application might automatically add asterisk operators onto the ends of all user search terms. Interfaces such as this make wildcard search more easily accessible to less sophisticated user communities, for which use of the asterisk operator might be unfamiliar.

# Performance impact of wildcard search

To optimize performance of wildcard search, use the following recommendations.

- **Account for increased time needed for indexing**. In general, if wildcard search is enabled in Endeca Server (even if it is not used by the users), it increases the time and disk space required for indexing. Therefore, consider first the business requirements for your Endeca application to decide whether you need to use wildcard search.

  **Note:** To optimize performance, the Dgraph performs wildcard indexing for words that are shorter than 1024 characters. Words that are longer than 1024 characters are not indexed for wildcard search.

- In addition, if the wildcard search is not enabled before you load the records, and you issue a Configuration Web Service request to enable it, after a large number of records exist in the data domain, this causes re-indexing and is associated with increased processing time.

- **Do not use "low information" queries**. For optimal performance, use wildcard search queries with at least 2-3 non-wildcarded characters in them, such as `abc*` and `ab*de`, and avoid wildcard searches with one non-wildcarded character, such as `a*`. Wildcard queries with extremely low information, such as `a*`, require a significant amount of time to process.

  Queries that contain only wildcards, or only wildcards and punctuation or spaces, such as `*.` (star followed by period), or `* *` (star space star), are rejected by the Oracle Endeca Server.

- **Analyze the format of your typical wildcard query cases**. This lets you be aware of performance implications associated with one specific wildcard search pattern.

  Do you have queries that contain punctuation syntax in between strings of text, such as `ab*c.def*`?

  For strings with punctuation, the Dgraph processes of the data domain generate lists of words that match each of the punctuation-separated wildcard expressions. Only in this case, Dgraph processes of the data domain use the `--wildcard-max <count>` setting to optimize their performance. Increasing the `--wildcard-max <count>` improves the completeness of results returned by wildcard search for strings with punctuation, but negatively affects performance. Thus you may want to find the number that provides a reasonable trade-off. You can specify this flag for the data domain profile, with `endeca-cmd --put-dd-profile --wildcard-max <value>`.

Chapter 28

# Search Characters

This section describes the semantics of matching search queries to result text.



## About search characters

Endeca Server supports configurable handling of punctuation and other non-alphanumeric characters in search queries.

This section does the following:

- Describes the semantics of matching search queries to result text (that is, records in record search or attribute values in value search) when either the query or result text contains non-alphanumeric characters.
- Explains how you can control this behavior using the search characters feature of Endeca Server.

**Note:** Search characters are supported only for the `unknown` language identifier. If the search query is tagged with one of the supported language codes, the search character feature is not used.

## Implementing search characters

Search indexing distinguishes between alphanumeric characters and non-alphanumeric characters, and supports the ability to mark some non-alphanumeric characters as significant for search operations.

You mark a non-alphanumeric character as a search character in the Global Configuration Record.

Search characters are configured globally for all search operations. For example, adding the plus (+) character marks it as a search character for value search and record search operations.

To mark a non-alphanumeric character as a search character:

1. Edit the contents of the `mdex-config_SearchChars` element of the Global Configuration Record in any text editor, as in the following example.

   This example marks "+" and "_" characters as search characters. You can add more than one character; they are not separated by any delimiters.

```
<mdex-config_SearchChars>+_</mdex-config_SearchChars>
```

2. To send the changes to Endeca Server, use the Configuration Web Service or Integrator ETL.

# Query matching semantics

The semantics of matching search queries to text containing special non-alphanumeric characters in Endeca Sever is based on indexing various forms of source text containing such characters.

Basically, user query terms are required to match exactly against indexed forms of the words in the source text to result in matches. Thus, to understand the behavior of query matching in the presence of non-alphanumeric characters, one must understand the set of forms indexed for source text.



## Categories of characters in indexed text

Endeca Server treats characters in indexed text based on three categories.

The categories are:

- Alphanumeric characters including ASCII characters as well as non-punctuation characters in ISO-Latin1.
- Non-alphanumeric search characters (configured using the search characters feature, as described below).
- Other non-alphanumeric characters (this category is the default for all non-alphanumeric characters not explicitly configured to be in group 2).

During data processing, each word in the source text (that is, searchable attributes for record search, attribute values for value search) is indexed based on the alternatives for handling characters from the three categories, which is described in subsequent topics.

## Indexing alphanumeric characters

Alphanumeric characters are included in all forms.

Because Endeca Server search operations are not case sensitive, alphabetic characters are always included in lowercase form, a technique commonly referred to as case folding.

## Indexing search characters

Search characters are non-alphanumeric characters that are specified as searchable.

Search characters are included as part of the token.

## Indexing non-alphanumeric characters

How non-alphanumeric characters that are not defined as search characters are treated depends on whether they are considered punctuation characters or symbols.

- Non-alphanumeric characters considered to be punctuation are treated as white space. In a multi-word search with the words separated by punctuation characters, word order is preserved as if it were a phrase search. The following characters are considered to be punctuation: ! @ # & ( ) – [ { } ] : ; ', ? / *
- Non-alphanumeric characters that are considered to be symbols are also treated as white space. However, unlike punctuation characters, they do not preserve word order in a multi-word search. If a symbol character is adjacent to a punctuation character, the symbol character is ignored. That is to say, the combination of the symbol character and the punctuation character is treated the same as the punctuation character alone.

  For example, a search on ice-cream would return the same results as a phrase search for "ice cream", while a search for ice~cream would return the same results as simply searching for ice cream. A search on ice-~cream would behave the same way as a search on ice-cream.

  Symbol characters include the following: ` ~ $ ^ + = < > "

# Search query processing

The semantics of matching search query terms to result text containing non-alphanumeric characters are described in this topic.

- During query processing, each user query term is transformed to replace all non-alphanumeric characters that are not marked as search characters with delimiters (spaces).
  - Non-alphanumeric characters considered to be punctuation (! @ # & ( ) – [ { } ] : ; ', ? / *) are treated as white space and preserve word order. This means that the equivalent of a quoted phrase search is generated. For that reason, all search features that are incompatible with quoted phrase search, such as spelling correction, stemming, and thesaurus expansion, are not activated. (For details, see *Phrase Search on page 278*.)
  - Non-alphanumeric characters that are considered to be symbols (` ~ $ ^ + = < > “) are also treated as white space. However, unlike punctuation characters, they do not preserve word order in a multi-word search.
- Alphabetic characters in the user query are replaced with lowercase equivalents, to ensure that they match against case-folded indexed strings.
- Each query term in the transformed query must exactly match some indexed string from the given source text for the text to be considered a hit.

As noted above, when parsing user-entered search terms, a query with non-searchable characters is transformed to replace all non-alphanumeric characters (that are not marked as search characters) with white space, but the treatment of word order depends on whether the character in question is considered to be a punctuation character or a symbol. The search behavior preserves the word order and proximity of the search term only in the case of punctuation characters.

For example, a search query for ice-cream will replace the hyphen (a punctuation character) with white space and return only records with this text:

- ice-cream

- ice cream

Records with this text are not returned because the word order and word proximity of text do not match the original query term:

- cream ice
- ice in the cream container

However, assuming the match mode is `All`, a search for ice~cream would return non-contiguous results for [ice AND cream].

# Dgraph flags for search characters

There are no Dgraph flags necessary to enable the search characters feature. Endeca Server automatically detects the additional search characters.

# Performance impact of setting search characters

Implementing search characters is an operation that changes the attribute on the GCR, and thus is a configuration change. If you issue this operation in an empty data domain, it runs quickly. If you issue this operation on an index with a large number of existing records using the Configuration Web Service, the processing of this operation can have performance impact largely caused by re-indexing.

Performance impact of this operation ( as well as other updating or configuration changing operations) consists of the following aspects:

- Such operations cause re-indexing, which is associated with CPU and memory usage costs. In particular, the operation for changing an attribute on the GCR will not finish until re-indexing is completed.
- Such operations are considered updating operations, which means that no other updates can be processed until a specific update is completed (for example, all updating Web service requests will wait in the queue during this update).

Chapter 29

# Spelling Correction and Did You Mean

This section describes the behavior of the Spelling Correction and Did You Mean features.



## About Spelling Correction and Did You Mean

Endeca Server supports two complementary forms of Spelling Correction.

The two forms of Spelling Correction are:

- Automatic Spelling Correction for record search and value search.
- Explicit spelling suggestions ("Did You Mean?") for record search.

The Automatic Spelling Correction feature enables search queries to return expected results when the spelling used in the query terms does not match the spelling used in the result text (that is, when the user misspells search terms). The Did You Mean feature returns suggested versions of a misspelled word, so that the user can re-issue the query with the new spelling.

Both features can be used in a single application, and both are supported by the same underlying spelling engine and Spelling Correction module.

Automatic Spelling Correction operates by computing alternate spellings for user query terms, evaluating the likelihood that these alternate spellings are the best interpretation, and then using the best alternate spell-corrected query forms to return extra search results. For example, a user might search for records containing the text *Abrham Lincoln*. With spelling correction enabled, the Oracle Endeca Server returns the expected results: those containing the text *Abraham Lincoln*.

Did You Mean (DYM) functionality allows an application to provide the user with explicit alternative suggestions for a keyword search. For example, assume that a user gets six results when searching for *valle* in a data set. The terms *valley* and *vale*, however, are much more prevalent in that data set. When the DYM feature is enabled, Endeca Server will respond with the six results for *valle*, but will also suggest that *valley* or

*vale* may be what the end-user actually intended. If multiple suggestions are returned, they will be sorted and presented according to the closeness of the match.

The behavior of spelling correction features is application-aware, because the spelling dictionary for a given data set is derived directly from the indexed source text, populated with the words found in all searchable values and attributes. Endeca Server returns spelling-corrected results as normal search results, for both value search and record search operations.

For example, in a set of records containing computer equipment, a search for *graphi* might spell-correct to *graphics*. In a different data set for sporting equipment, the same search might spell-correct to *graphite*.

Also keep in mind that the results of spelling corrections and DYM suggestions depend on the language specified for the query, which is set with the `Language` element of the `TextSearchFilter` type (for record search) and the `ValueSearchConfig` type (for value search). For example, using the search term `pech` in an English language search may result in a spelling correction (such as to `peach`), while no correction might be done for a French language search (because `pêche` is a word in French).

# Logic used for spelling correction

At a high level, the spelling engine in Endeca Server performs the following steps related to spelling correction for a given search query.

1. If the search terms generate more than a certain number of hits without any correction, then the spelling engine does not generate any corrections or suggestions.

   For the automatic correction, the threshold for the number of hits is 1. For the Did You Mean feature, the threshold for the number of hits is 20.

2. For each term in the query, the spelling engine finds the 32 corrections with the lowest spelling scores. A low spelling score signifies that the correction is similar to the search term.

   For the Aspell mode that the Dgraph uses for English, the spelling score is based on phonetic distance. The 32 corrections are pruned to corrections with a spelling score below a certain threshold. For the automatic correction, the spelling threshold is 125, for Did You Mean, the spelling threshold is 175.

3. The spelling engine tests each correction in place of the original search term it corrects. Only those corrections which increase the number of hits (relative to the original query) without reducing the number of terms matched are eligible to be returned.

4. The spelling engine selects the best correction based on which of the eligible corrections has the highest number of hits. For record search, this is the number of records matched. For value search, this is the number of records associated with the set of values matched.

   **Note:** For more information about the difference in the treatment of results between record search and value search, see the topic *How value search treats number of results on page 298*.

To change the Dgraph configuration for Automatic Spelling Correction and DYM, you can rebuild the spelling dictionary with the `updateSpellingDictionaries` operation of the Data Ingest Web Service.

Suggestions for automatic correction are not exposed by the Oracle Endeca Server, that is, you cannot update the dictionary manually in the installed product.

In the Global Configuration Record, you can configure the indexing parameters such as minimum word occurrences and maximum and minimum word length. These parameters let you set boundaries to indicate to the Dgraph which words to include in the spelling dictionary.

*How value search treats number of results*

## How value search treats number of results

Value search results may vary if spelling correction is performed.

An important note applies to the options and behavior associated with value search spelling correction: in situations where the number of results is evaluated by an option or in the scoring of words or queries performed by the spelling engine, value search uses an alternate definition of number of results. Instead of using the simple number of hits returned to the user as this value (which is perfectly reasonable in the case of record search), value search instead uses the number of records associated with the set of value search results computed for a given query.

In other words, value search follows an additional level of indirection to weight the value of results computed by spelling suggestion queries according to the number of records that these values would lead to if selected in a navigation query. This alternate definition of number of results allows consistent behavior between spelling corrections computed for value and record search operations when given the same query terms.

# Enabling spelling correction and updating spelling dictionaries

The `updateSpellingDictionaries` operation of the Data Ingest Web Service performs two functions at once: when it runs, it enables spelling correction and DYM features, and also rebuilds the spelling dictionaries for this data domain.

The operation allows you to rebuild the spelling dictionaries for spelling correction from the data corpus while continuing to issue queries and updates to Endeca Server, and without stopping and restarting the Dgraph. Run this operation after you have added data records to the data domain, to enable spelling correction in the Dgraph for this data domain.

**Important:** In a cluster of Dgraphs serving a specific data domain (also known as the data domain cluster), this operation runs successfully on the Dgraph node that can accept updating requests for this data domain (this is the leader Dgraph node). Endeca Server automatically routes this operation to the leader node in the data domain cluster.

During the data ingest process, you can run the `updateSpellingDictionaries` operation periodically to update the spelling dictionary used by the Dgraph for Automatic Spelling Correction and DYM.

The `updateSpellingDictionaries` operation performs the following actions:

- Crawls the text search index for all terms which meet the constraint settings.

  The constraint settings include minimum word occurrences and maximum and minimum number of characters, for records and attribute values. The Dgraph uses these constraints to update the spelling dictionary. You can change them in the Global Configuration Record.

- Updates the spelling dictionaries stored in the Dgraph for processing of all queries arriving after this update to the data files. The Dgraph uses these updated dictionaries when processing all future queries.

The Dgraph applies the updated settings while continuing to run queries and without needing to restart.

For information on how to run the `updateSpellingDictionaries` operation directly on Endeca Server, see the *Oracle Endeca Server Data Loading Guide*, or, if you use Integrator ETL for data loading, see its documentation.

# Spelling mode (Aspell)

Spelling features of Endeca Server compute contextual suggestions at the full query level.

That is, suggestions may include one or more corrected query terms, which can depend on context such as other words used in the query. To determine these full query suggestions, the Dgraph relies on the low-level Aspell spelling module to compute single-word suggestions, that is, words similar to a given user query term and contained within the application-specific dictionary.

## Aspell spelling module

Endeca Server supports one internal spelling module, Aspell. It supports sound-alike corrections (using English phonetic rules). It does not support corrections to non-alphabetic/non-ASCII terms (such as *café*, *1234*, or *A&M*).

# Retrieving spelling corrections and DYM in query results

You can retrieve automatic spelling corrections and suggested corrections (Did You Mean) information in a query using the `SearchAdjustmentConfig` type in your Conversation Web Service request.

If spelling is enabled in the data domain and you want the Conversation Web Service response to contain supplemental information about spelling corrections and spelling suggestions (DYM), a `SearchAdjustmentConfig` type is required. If it is included, spelling corrections and/or DYM suggestions are returned as part of the response.

It is important to realize that if spelling is enabled, spelling auto-correction occurs even if the `SearchAdjustmentConfig` type is not included. However, while spelling correction takes place, the spelling corrections and DYM suggestions are not returned in the response.

## SearchAdjustmentConfig syntax

The format for a `SearchAdjustmentConfig` is:

```
<SearchAdjustmentConfig Id="?">
   <StateName>?</StateName>
</SearchAdjustmentConfig>
```

where the attributes mean:

| Attribute | Meaning |
|---|---|
| `Id` | A required attribute that provides an arbitrary identifier for this configuration. |

| Attribute | Meaning |
|---|---|
| StateName | Specifies an existing named state in the request, using these rules:<br>• If the request has multiple named states, then the StateName element must reference one (and only one) of the named states.<br>• If the request has only one named state, then it is optional as to whether the StateName element is used to reference that named state (as the state will be used in any event in the RecordListConfig).<br>• If the request has an unnamed state, then the StateName element cannot be used. |

## SearchAdjustmentConfig example

This record search example includes the SearchAdjustmentConfig type, to ensure that spelling correction adjustments and DYM suggestions are returned in the response:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <TextSearchFilter Key="Flavors" Mode="AllPartial" EnableSnippeting="false" Language="en">pech<
/TextSearchFilter>
   </State>
   <RecordListConfig Id="RecList" MaxPages="10">
      <Column>Flavors</Column>
      <RecordsPerPage>5</RecordsPerPage>
   </RecordListConfig>
   <SearchAdjustmentConfig Id="SpellCorrect"/>
</Request>
```

Note that in the example, the word "peach" is misspelled as "pech".

## Automatic spelling correction response

The response from the Conversation Web Service contains the original State from the request with search filter applied, as well as the original (not yet spelling-corrected) term of "pech". The response also includes the SearchAdjustments type:

```
<cs:SearchAdjustments Id="SpellCorrect">
   <cs:AppliedAdjustment>
       <cs:TextSearchFilter Key="Flavors" Mode="AllPartial">pech</cs:TextSearchFilter>
       <cs:AdjustedTerms>peach</cs:AdjustedTerms>
   </cs:AppliedAdjustment>
</cs:SearchAdjustments>
```

The SearchAdjustments response includes the automatically-corrected term "peach" in the AdjustedTerms element for AppliedAdjustment. The name of the element ("AppliedAdjustment") tells you that the corrected spelling was automatically applied to the query, and the results (2593 records in this example) reflect that correction.

## Explicit spelling suggestions (DYM) response

For explicit spelling suggestions, the SearchAdjustments response contains a SuggestedAdjustment element (instead of an AppliedAdjustment element), as in this example from a search that used the misspelling of "pechr" instead of "peach":

```
<cs:SearchAdjustments Id="SpellCorrect">
```

```
   <cs:SuggestedAdjustment RecordCountIfApplied="2593">
       <cs:TextSearchFilter Key="Flavors" Mode="AllPartial">pechr</cs:TextSearchFilter>
       <cs:SuggestedTerms>peach</cs:SuggestedTerms>
   </cs:SuggestedAdjustment>
</cs:SearchAdjustments>
```

The name of the element ("SuggestedAdjustment") tells you that the term is just a DYM suggestion, which means that it was not applied to the query. The `RecordCountIfApplied` element shows that if the suggestion had been applied, 2593 records would have been returned. To retrieve those records, re-send the record search query with the suggested term of "peach" for the search term.

# Configuring constraints for spelling dictionaries

Endeca Server selects words for the spelling dictionary based on pre-defined constraints. Modifying these constraints can be useful for improving performance of spell-corrected searches.

The constraint settings are available in the Global Configuration Record.

You can use these configuration settings to tune and improve the types of spelling corrections produced by Endeca Server. For example, setting the minimum number of word occurrences can direct the attention of the spelling correction algorithm away from infrequent terms and towards more popular (frequently occurring) terms, which might be deemed more likely to correspond to intended user search terms.

To configure the settings which the Dgraph uses to generate spelling dictionary entries:

1. In the editor of your choice, edit the constraints in the GCR that the Dgraph should use for adding words to the spelling dictionary.

   You can separately edit settings for entries in the dictionary for record search and value search. In other words, for each attribute assignment on a record, and for each attribute value, you could specify the following settings in the Global Configuration Record:

| Attribute | Type | Description |
|---|---|---|
| `mdex-config_SpellingRecordMinWordOccur` | Int | Specifies the minimum number of times a word must occur in a standard attribute value (record assignment on an attribute) for it to be indexed for spelling correction. The default value is 4. |
| `mdex-config_SpellingRecordMinWordLength` | Int | Specifies the minimum number of characters that a word must contain in a standard attribute value (record assignment on an attribute) for it to be indexed for spelling correction. The default value is 3. |
| `mdex-config_SpellingRecordMaxWordLength` | Int | Specifies the maximum number of characters that a word may contain for it to be indexed for spelling correction. The default value is 16. |

| Attribute | Type | Description |
|---|---|---|
| `mdex-config_SpellingDValMinWordOccur` | Int | Specifies the minimum number of times a word must occur in a managed attribute value for it to be indexed for spelling correction. The default value is 1. |
| `mdex-config_SpellingDValMinWordLength` | Int | Specifies the minimum number of characters that a word must contain in a managed attribute value for it to be indexed for spelling correction. The default value is 3. |
| `mdex-config_SpellingDValMaxWordLength` | Int | Specifies the maximum number of characters that a word may contain for it to be indexed for spelling correction. The default value is 16. |

2. To send the updated GCR to the Oracle Endeca Server, use the Configuration Web Service directly or use Integrator ETL. For information, see either the section on Configuration Web Service in this guide, or, if you are using Integrator ETL, see the *Oracle Endeca Information Discovery Integrator ETL User's Guide*.
3. Run the `updateSpellingDictionaries` operation of the Data Ingest Web Service on the data domain in order for these changes to take effect.

# About word-break analysis

Word-break analysis allows the Spelling Correction feature to consider alternate queries computed by changing the word divisions in the user's query.

For example, if the query is `Back Street Boys`, word-break analysis could instruct Endeca Server to consider the alternate `Backstreet Boys`.

The following statements describe how word-break analysis works in the Dgraph:

- It is enabled by default.
- As part of the word-break analysis, the Dgraph removes breaks from the original term, or adds breaks to the original term if needed.
- The maximum number of word breaks that the Dgraph adds to or removes from a query is one.
- The minimum length for a new term created by word-break analysis is two characters. The Dgraph does not correct words that are smaller than 2 characters. For example, it does not correct `anear` to a `near`. It could correct to `an ear` if there are actual terms in the data corpus that match both `an` and `ear`.
- When word-break analysis is applied to a query, it requires that the substrings that the term is broken up into appear in the data in succession. For example, starting with the query *box17*, word-break analysis would find *box 17*, as well as *box-17*, assuming that the hyphen (-) has not been specified as a search character. However, it would not find *17 old boxes*, because the target terms do not appear in order.

# Troubleshooting Spelling Correction and Did You Mean

If spell-corrected results are not returned for words with expected spell-corrected options in the data, use these suggestions for troubleshooting.

- When debugging spelling behavior, pay close attention to the errors of the Dgraph on startup, where problems in spelling configuration are typically reported.
- Did You Mean can in some cases correct a word to one on the stop words list.

# Performance impact for Spelling Correction and Did You Mean

Spelling correction performance is impacted by the size of the dictionary in use.

Spell-corrected keyword searches with many words, in systems with very large dictionaries, can take a disproportionately long time to process relative to other Endeca Server requests. Those searches can cause requests that immediately follow such a search to wait while the spelling recommendations are being sought and considered.

It is important to carefully analyze the performance of the system together with application requirements before deploying a production application.

Chapter 30

# Stemming and Thesaurus

This section describes the tasks involved in implementing the Stemming and Thesaurus features.



## Overview of stemming and thesaurus

Endeca Server supports stemming and thesaurus features that allow keyword search queries to match text containing alternate forms of the query terms or phrases.

The definitions of these features are as follows:

- The stemming feature allows the system to consider alternate forms of individual words as equivalent for the purpose of search query matching. For example, it is often desirable for singular nouns to match their plural equivalents in the searchable text, and vice versa.
- The thesaurus feature allows the system to return matches for related concepts to words or phrases contained in user queries. For example, a thesaurus entry may allow searches for *Mark Twain* to match text containing the phrase *Samuel Clemens*.

Both the thesaurus and stemming features rely on defining equivalent textual forms that are used to match user queries to searchable text data. Because these features are based on similar concepts, and because they are typically configured to operate in conjunction to achieve desired query matching effects, both features and their interactions are discussed in one section.

## About the stemming feature

The stemming feature broadens search results to include word roots and word derivations.

Stemming is enabled by default in an Endeca data domain.

Stemming is intended to allow words with a common root form (such as the singular and plural forms of nouns) to be considered interchangeable in search operations. For example, search results for the word *shirt* will include the derivation *shirts*, while a search for *shirts* will also include its word root *shirt*.

Stemming equivalences are defined among single words. For example, stemming is used to produce an equivalence between the words *automobile* and *automobiles* (because the first word is the stem form of the

second), but not to define an equivalence between the words *vehicle* and *automobile* (this type of concept-level mapping is done via the thesaurus feature).

Stemming equivalences are strictly two-way (that is, all-to-all). For example, if there is a stemming entry for the word *truck*, then searches for *truck* will always return matches for both the singular form (*truck*) and its plural form (*trucks*), and searches for *trucks* will also return matches for *truck*. In contrast, the thesaurus feature supports one-way mappings in addition to two-way mappings.

**Note:** The stemming implementation does not include decompounding. Decompounding is the ability to decompose a compound word (such as *kindergarten*) into its single word components (*kinder* and *garten*) and then find occurrences based on the smaller words.

### Supported languages for stemming

The list of supported languages for stemming is in the topic, *Supported languages on page 154*.

You should specify a language ID for each of your attributes (via the `mdex-property_Language` property in the attribute's PDR). At ingest time, the Dgraph creates a separate stemming dictionary for each configured language. The dictionaries are stored in the Endeca data domain and cannot be modified by the user.

*Types of stemming matches and sort order*

## Types of stemming matches and sort order

Stemming can produce one of three match types.

If stemming is enabled, a search on a given term (*T*) will produce one or more of these results:

- **Literal matches:** Any occurrence of *T* will always produce a match.
- **Stem form matches:** Matches will occur on the stem form of *T* (assuming that *T* is not a stem form). For example, if *T* is *children*, then *child* (the stem form) will also match.
- **Inflected form matches:** Matches will occur on all inflected forms of the stem form of *T*. For example, if *T* is the verb *ran* (as in *Jane ran in the Boston Marathon*), then matches will include the stem form (*run*) and inflected forms (such as *runs* and *running*). (Note that although this example is in English, stemming for inflected verb forms is not supported for English; see below for support details.)

The order of the returned results depends on the sorting configuration:

- If relevance ranking is enabled and the Interpreted (interp) module is used, literal matches will always have higher priority than stem form and inflected form matches.
- If relevance ranking is not enabled but you have set a record sort order, the results will come back in that sort order.
- If relevance ranking is not enabled and there is no record sort order, the order of the results is completely arbitrary.

# About the thesaurus feature

The thesaurus feature allows you to configure rules for matching queries to text containing equivalent words or concepts.

The thesaurus is intended for specifying concept-level mappings between words and phrases. Even a modest number of well-thought-out thesaurus entries can greatly improve your users' search experience.

**Note:** Only one global thesaurus is supported for an Endeca data domain. In other words, language-specific thesauruses are not supported (for example, one thesaurus for English, a second for French, and so on).

The thesaurus feature is at a higher level than the stemming feature, because thesaurus matching and query expansion respects stemming equivalences, whereas the stemming module is unaware of thesaurus equivalences.

For example, if you define a thesaurus entry mapping the words *automobile* and *car*, and there is a stemming equivalence between *car* and *cars*, then a search for *automobile* will return matches for *automobile*, *car*, and *cars*. The same results will also be returned for the queries *car* and *cars*.

The thesaurus supports specifying multi-word equivalences. For example, an equivalence might specify that the phrase *Mark Twain* is interchangeable with the phrase *Samuel Clemens*. It is also possible to mix the number of words in the phrase-forms for a single equivalence. For example, you can specify that *wine opener* is equivalent to *corkscrew*.

Multi-word equivalences are matched on a phrase basis. For example, if a thesaurus equivalence between *wine opener* and *corkscrew* is defined, then a search for *corkscrew* will match the text *stainless steel wine opener*, but will not match the text *an effective opener for wine casks*.

Thesaurus equivalences can be either one-way or two-way:

- One-way mapping specifies only one direction of equivalence. That is, one "From" term is mapped to one or more "To" terms, but none of the "To" terms are mapped to the "From" term. Only one "From" term can be specified.

  For example, assume you define a one-way mapping from the phrase *red wine* to the phrases *merlot* and *cabernet sauvignon*. This one-way mapping ensures that a search for *red wine* also returns any matches containing the more specific terms *merlot* or *cabernet sauvignon*. But you avoid returning matches for the more general phrase *red wine* when the user specifically searches for either *merlot* or *cabernet sauvignon*.

- Two-way (or all-to-all) mapping means that the direction of a word mapping is equivalent between the words. For example, a two-way mapping between *stove*, *range*, and *oven* means that a search for one of these words will return all results matching any of these words (that is, the mapping marks the forms as strictly interchangeable).

  When you define a two-way mapping, you do not specify a "From" term. Instead, you specify two or more "To" terms.

Unlike the stemming module, the thesaurus feature lets you define multiple equivalences for a single word or phrase. These multiple equivalences are considered independent and non-transitive.

For example, we might define one equivalence between *football* and *NFL*, and another between *football* and *soccer*. With these two equivalences, a search for *NFL* will return hits for *NFL* and hits for *football*, a search for *soccer* will return hits for *soccer* and *football*, and a search for *football* will return all of the hits for *football*, *NFL*, and *soccer*. However, searches for *NFL* will not return hits for *soccer* (and vice versa).

This non-transitive nature of the thesaurus is useful for defining equivalences containing ambiguous terms such as *football*. The word *football* is sometimes used interchangeably with *soccer*, but in other cases *football* refers to American football, which is played professionally in the NFL. In other words, the term *football* is ambiguous.

When you define equivalences for ambiguous terms, you do not want their specific meanings to overlap into one another. People searching for *soccer* do not want hits for *NFL*, but they may want at least some of the hits associated with the more general term *football*.

Thesaurus entries are essentially used to produce alternate forms of the user query, which in turn are used to produce additional query results. Note that a maximum of three terms in a single search query are subject to thesaurus replacement. This means that up to 3 words in a user's search query can be replaced with thesaurus entries. If more than three words match thesaurus entries, none of the extra words will be expanded by the thesaurus engine. This thesaurus-expansion limit cannot be changed.

This behavior is particularly important in the presence of overlapping thesaurus forms. For example, suppose that you define an equivalence between *red wine* and *vino rosso*, and a second equivalence between *wine opener* and *corkscrew*. The query *red wine opener* might match the thesaurus entries in two different ways: *red wine* could be mapped to *vino rosso* based on the first entry; or *wine opener* could be mapped to *corkscrew* based on the second entry.

Using the maximal-expansion rule, this issue is resolved by expanding to all possible queries. In other words, Endeca Server returns hits for all of the queries: *red wine opener*, *vino rosso opener*, and *red corkscrew*.

*Adding, modifying, or deleting thesaurus entries*

*Troubleshooting the thesaurus*

## Adding, modifying, or deleting thesaurus entries

Thesaurus entries are added in the THESAURUS XML document.

All XML configuration documents are present in the data files of the Oracle Endeca Server. You can edit them using the format specified in the *Dgraph Configuration Reference*, found in this guide. After these documents are edited, you can send them to Endeca Server using the Configuration Web Service or Integrator ETL, thus specifying the configuration you want.

To add a one-way or two-way thesaurus entry, or modify and delete existing thesaurus entries:

1. In any editor, edit the contents of the THESAURUS XML document.
2. Use Integrator ETL or the request created with the Configuration Web Service to send the THESAURUS document to the Oracle Endeca Server.

## Troubleshooting the thesaurus

The following thesaurus clean-up rules should be observed to avoid performance problems related to expensive and non-useful thesaurus search query expansions.

- Do not create a two-way thesaurus entry for a word with multiple meanings. For example, *khaki* can refer to a color as well as to a style of pants. If you create a two-way thesaurus entry for `khaki = pants`, then a user's search for *khaki towels* could return irrelevant results for *pants*.
- Do not create a two-way thesaurus entry between a general and several more-specific terms, such as:

```
top = shirt = sweater = vest
```

This increases the number of results the user has to go through while reducing the overall accuracy of the items returned. In this instance, better results are attained by creating individual one-way thesaurus entries between the general term top and each of the more-specific terms.

- A thesaurus entry should never include a term that is a substring of another term in the entry.

  For example, consider the two-way equivalency:

  ```
  Adam and Eve = Eve
  ```

  If users type *Eve*, they get results for *Eve or (Adam and Eve)* (that is, the same results they would have gotten for *Eve* without the thesaurus). If users type *Adam and Eve*, they get results for *(Adam and Eve) or Eve*, causing the *Adam and* part of the query to be ignored.

- Stop words such as *and* or *the* should not be used in single-word thesaurus forms. For example, if *the* has been configured as a stop word, an equivalency between *thee* and *the* is not useful.

  You can use stop words in multi-word thesaurus forms, because multi-word thesaurus forms are handled as phrases. In phrases, a stop word is treated as a literal word and not a stop word.

- Avoid multi-word thesaurus forms where single-word forms are appropriate. In particular, avoid multi-word forms that are not phrases that users are likely to type, or to which phrase expansion is likely to provide relevant additional results.

  For example, the two-way thesaurus entry:

  ```
  Aethelstan, King Of England (D. 939) = Athelstan, King Of England (D. 939)
  ```

  should be replaced with the single-word form:

  ```
  Aethelstan = Athelstan
  ```

- Thesaurus forms should not use non-searchable characters. For example, the one-way thesaurus entry:

  ```
  Pikes Peak -> Pike’s Peak
  ```

  should be used only if the apostrophe (') is enabled as a search character.

# Dgraph flags for stemming and thesaurus

Stemming and thesaurus data that has been configured is automatically enabled for use during text indexing and search query processing. In addition, there is no Endeca Server configuration necessary to configure thesaurus and stemming information.

# Interactions with other search features

As core features of the Endeca Server search subsystem, stemming and the thesaurus have interactions with other search features.

The following sections describe the types of interactions between the various search features.

## Search characters

The search character set configured for the application dictates the set of available characters for stemming and thesaurus entries. By default, only alphanumeric ASCII characters may be used in stemming and thesaurus entries. Additional punctuation and other special characters may be enabled for use in stemming and thesaurus entries by adding these characters to the search character set.

Endeca Server matches user query terms to thesaurus forms using the following rule: all alphanumeric and search characters must match against the stemming and thesaurus forms exactly; other characters in the user search query are treated as word delimiters. For details on search characters, see *Search Characters on page 291*.

## Spelling

Spelling correction is a closely-related feature to stemming and thesaurus functionality, because spelling auto-correction essentially provides an additional mechanism for computing alternate versions of the user query. In the Dgraph, spelling is handled as a higher-level feature than stemming and thesaurus. That is, spelling correction considers only the raw form of the user query when producing alternate query forms.

Alternate spell-corrected queries are then subject to all of the normal stemming and thesaurus processing. For example, if the user enters the query *telvision* and this query is spell-corrected to *television*, the results will also include results for the alternate forms *televisions*, *tv*, and *tvs*.

Note that in some cases, the thesaurus feature is used as a replacement or in addition to the system's standard spelling correction features. In general, this technique is discouraged. The vast majority of actual misspelled user queries can be handled correctly by the spelling correction subsystem. But in some rare cases, the spelling correction feature cannot correct a particular misspelled query of interest; in these cases it is common to add a thesaurus entry to handle the correction. If at all possible, such entries should be avoided as they can lead to undesirable feature interactions.

## Stop words

Stop words are words configured to be ignored by the Dgraph search query engine. A stop word list typically includes words that occur too frequently in the data to be useful (for example, the word *bottle* in a wine data set), as well as words that are too general (such as *clothing* in an apparel-only data set).

If *the* is marked as a stop word, then a query for *the computer* will match to text containing the word *computer*, but possibly missing the word *the*.

Stop words are not currently expanded by the stemming and thesaurus equivalence set. For example, suppose you mark *item* as a stop word and also include a thesaurus equivalence between the words *item* and *items*. This will not automatically mark the word *items* as a stop word; such expansions must be applied manually.

Stop words are respected when matching thesaurus entries to user queries. For example, suppose you define an equivalence between *Muhammad Ali* and *Cassius Clay* and also mark *M* as a stop word (it is not uncommon to mark all or most single letter words as stop words). In this case, a query for *Cassius M. Clay* would match the thesaurus entry and return results for *Muhammad Ali* as expected.

## Phrase search

A phrase search is a search query that contains one or more multi-word phrases enclosed in quotation marks. The words inside phrase-query terms are interpreted strictly literally and are not subject to stemming or thesaurus processing. For example, if you define a thesaurus equivalence between *Jennifer Lopez* and *JLo*, normal (unquoted) searches for *Jennifer Lopez* will also return results for *JLo*, but a quoted phrase search for *"Jennifer Lopez"* will not return the additional *JLo* results.

### Relevance ranking

It is typically desirable to return results for the actual user query ahead of results for stemming and/or thesaurus transformed versions of the query. This type of result ordering is supported by the Relevance Ranking modules. In particular, the module that is affected by thesaurus expansion and stemming is **Interp**. The module that is not affected by thesaurus and stemming is **Freq**.

## Performance impact of stemming and thesaurus

Stemming and thesaurus equivalences generally add little or no time to data processing and indexing, and introduce little space overhead (beyond the space required to store the raw string forms of the equivalences).

In terms of online processing, both features will expand the set of results for typical user queries. While this generally slows search performance (search operations require an amount of time that grows linearly with the number of results), typically these additional results are a required part of the application behavior and cannot be avoided.

The overhead involved in matching the user query to thesaurus and stemming forms is generally low, but could slow performance in cases where a large thesaurus (tens of thousands of entries) is asked to process long search queries (dozens of terms). Typical applications exhibit neither extremely large thesauri nor very long user search queries.

Because matching for stemming entries is performed on a single-word basis, the cost for stemming-oriented query expansion does not grow with the size of the stemming database or with the length of the query.

Chapter 31

# Relevance Ranking

This section describes the tasks involved in implementing the Relevance Ranking feature.



## About the relevance ranking feature

Relevance ranking allows you to control the order in which search results are displayed to the end user of a front-end application powered by Endeca Server.

Typically, the relevance ranking feature is used to ensure that the most important search results are displayed earliest to the user, because users of search-oriented information retrieval systems are often unwilling to page through large result sets.

Relevance ranking can be used to independently control the result ordering for both record search and value search queries. You can establish a system-default relevance ranking for both record search and value search. In addition, you can assign relevance ranking on a per-query basis for both search types.

The importance of a search result is generally an application-specific concept. Thus, the relevance ranking feature provides a flexible, configurable set of result ranking modules. These modules can be used in combinations (called *relevance ranking strategies*) to produce a wide range of relevance ranking effects. Results are scored according to the order of ranking modules within the strategy.

**Note:** Because relevance ranking is a complex and powerful feature, this documentation provides recommended strategies that you can use as a point of departure for further development. For details, see the "Recommended strategies" topic in this section.

## Relevance ranking modules

Relevance ranking modules are the building blocks from which you build the relevance ranking strategies to apply to your search interfaces.

This section describes the available set of relevance ranking modules and their scoring behaviors.

## Exact

The `Exact` module provides a finer grained (but more computationally expensive) alternative to the `Phrase` module.

The `Exact` module groups results into three strata based on how well they match the query string:

- The highest stratum contains results whose complete text matches the user's query exactly.
- The middle stratum contains results that contain the user's query as a subphrase.
- The lowest stratum contains other hits (such as normal conjunctive matches). Any match that would not be a match without query expansion lands in the lowest stratum. Also in this stratum are records that do not contain relevance ranking terms.

The `Exact` module is computationally expensive, especially on large text fields. It is intended for use only on small text fields (such as managed attribute values or small managed attribute values like part IDs). This module should not be used with large or offline documents. Use of this module in these cases will result in very poor performance and/or application failures due to request timeouts. The `Phrase` module, with and without approximation turned on, does similar but less sophisticated ranking that can be used as a higher performance substitute.

## Field

The `Field` module ranks documents based on the search interface field with the highest priority in which it matched.

Only the best field in which a match occurs is considered. The `Field` module is often used in relevance ranking strategies for catalog applications, because the category or product name is typically a good match. `Field` assigns a score to each result based on the static rank of the standard or managed attribute member

(or members) of the search interface that caused the document to match the query. Static field ranks are assigned based on the order in which members of a search interface are listed in the search interface configuration. The first member has the highest rank.

By default, matches caused by cross-field matching are assigned a score of zero. The score for cross-field matches can be set explicitly in the `CROSS_FIELD_RELEVANCE_RANK` attribute of the `SEARCH_INTERFACE` element. This element is used only for search interfaces that have the Field module and are configured to support cross-field matches. All non-zero ranks must be non-equal and only their order matters.

For example, a search interface might contain both Title and DocumentContent standard attributes, where hits on Title are considered more important than hits on DocumentContent (which in turn are considered more important than cross-field matches). Such a ranking is implemented by assigning the highest rank to Title, the next highest rank to DocumentContent, and setting the `CROSS_FIELD_RELEVANCE_RANK` attribute to a low integer such as 0 or 1.

The `Field` module is only valid for record search operations. This module assigns a score of zero to all results for other types of search requests. In addition, `Field` treats all matches the same, whether or not they are due to query expansion.

## First

Designed primarily for use with unstructured data, the `First` module ranks documents by how close the query terms are to the beginning of the document.

The `First` module groups its results into variably-sized strata. The strata are not the same size, because while the first word is probably more relevant than the tenth word, the 301st is probably not so much more relevant than the 310th word. This module takes advantage of the fact that the closer something is to the beginning of a document, the more likely it is to be relevant.

The `First` module works as follows:

- When the query has a single term, `First`'s behavior is straight-forward: it retrieves the first absolute position of the word in the document, then calculates which stratum contains that position. The score for this document is based upon that stratum; earlier strata are better than later strata.
- When the query has multiple terms, `First` behaves as follows: The first absolute position for each of the query terms is determined, and then the median position of these positions is calculated. This median is treated as the position of this query in the document and can be used with stratification as described in the single word case.
- With query expansion (using stemming, spelling correction, or the thesaurus), the First module treats expanded terms as if they occurred in the source query. For example, the phrase *glucose intolerence* would be corrected to *glucose intolerance* (with *intolerence* spell-corrected to *intolerance*). First then continues as it does in the non-expansion case. The first position of each term is computed and the median of these is taken.
- In a partially matched query, where only some of the query terms cause a document to match, `First` behaves as if the intersection of terms that occur in the document and terms that occur in the original query were the entire query. For example, if the query *cat bird dog* is partially matched to a document on the terms *cat* and *bird*, then the document is scored as if the query were *cat bird*. If no terms match, then the document is scored in the lowest strata.

**Note:** The `First` module does not work with Boolean searches, cross-field matching, or wildcard search. It assigns all such matches a score of zero.

## Frequency

The Frequency (`Freq`) module provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text.

Results with more occurrences of the user search terms are considered more relevant.

The score produced by the Frequency module for a result record is the sum of the frequencies of all user search terms in all fields (standard or managed attributes in the search interface in question) that match a sufficient number of terms. The number of terms depends on the match mode, such as all terms in a query with search mode `All`, a sufficient number of terms in a query with search mode `Partial`, and so on. Cross-field match records are assigned a score of zero. Total scores are capped at 1024; in other words, if the sum of frequencies of the user search terms in all matching fields is greater than or equal to 1024, the record gets a score of 1024 from the `Freq` module.

For example, suppose we have the following record:

```
{Title="test record", Abstract="this is a test", Text="one test this is"}
```

An `All` search for *test this* would cause Frequency to assign a score of 4, since *this* and *test* occur a total of 4 times in the fields that match all search terms (Abstract and Text, in this case). The number of phrase occurrences (just one in the Text field) doesn't matter, only the sum of the individual word occurrences. Also note that the occurrence of *test* in the Title field does not contribute to the score, since that field did not match all of the terms.

An `All` search for *one record* would hit this record, assuming that cross field matching was enabled. But the record would get a score of zero from `Freq`, because no single field matches all of the terms. Freq ignores matches due to query expansion (that is, such matches are given a rank of 0).

**Note:** Due to performance issues, do not use the Frequency module with standalone relevance ranking (that is, per-query relevance ranking).

## Glom

The `Glom` module ranks single-field matches ahead of cross-field matches and also ahead of non-matches (records that do not contain the search term).

The `Glom` module serves as a useful tie-breaker function in combination with the Maximum Field module. It is only useful in conjunction with record search operations. If you want a strategy that ranks single-field matches first, cross-field matches second, and no matches third, then use the `Glom` module followed by the Number of Terms (`Nterms`) module.

`Glom` treats all matches the same, whether or not they are due to query expansion.

### Glom interaction with search modes

The `Glom` module considers a single-field match to be one in which a single field has enough terms to satisfy the conditions of the match mode. For this reason, in the `Any` search mode, cross-field matches are impossible, because a single term is sufficient to create a match. Every match is considered to be a single-field match, even if there were several search terms.

For `Partial` search mode, if the required number of matches is two, the `Glom` module considers a record to be a single-field match if it has at least one field that contains two or more or the search terms. You cannot rank results based on how many terms match within a single field.

For more information about search modes, see *Search Modes on page 265*.

## Interpreted

Interpreted (`interp`) is a general-purpose module that assigns a score to each result record based on the query processing techniques used to obtain the match.

Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching.

Specifically, the Interpreted module ranks results as follows:

1. All non-partial matches are ranked ahead of all partial matches. For more information, see *Search Modes on page 265*.
2. Within the above strata, all single-field matches are ranked ahead of all cross-field matches. For more information, see *Search Interfaces on page 252*.
3. Within the above strata, all non-spelling-corrected matches are ranked above all spelling-corrected matches. See *Spelling Correction and Did You Mean on page 295* for more information.
4. Within the above strata, all thesaurus matches are ranked below all non-thesaurus matches. See *Stemming and Thesaurus on page 303* for more information.
5. Within the above strata, all stemming matches are ranked below all non-stemming matches. See *Stemming and Thesaurus on page 303* for more information.

## Maximum Field

The Maximum Field (`maxfield`) module behaves identically to the `Field` module, except in how it scores cross-field matches.

Unlike `Field`, which assigns a static score to cross-field matches, Maximum Field selects the score of the highest-ranked field that contributed to the match.

Note the following:

- Because Maximum Field defines the score for cross-field matches dynamically, it does not make use of the cross-field setting in the search interface.
- Maximum Field is only valid for record search operations. This module assigns a score of zero to all results for other types of search requests.
- Maximum Field treats all matches the same, whether or not they are due to query expansion.

## Number of Fields

The Number of Fields (`Numfields`) module ranks results based on the number of fields in the associated search interface in which a match occurs.

Note that we are counting whole-field rather than cross-field matches. Therefore, a result that matches two fields matches each field completely, while a cross-field match typically does not match any field completely.

**Note:** `Numfields` treats all matches the same, whether or not they are due to query expansion. The `Numfields` module is only useful in conjunction with record search operations.

## Number of Terms

The Number of Terms (or `Nterms`) module ranks matches according to how many query terms they match.

For example, in a three-word query, results that match all three words will be ranked above results that match only two, which will be ranked above results that match only one, which will be ranked above results that had no matches.

With multiple term searches, `Nterms` only ranks the terms in the field with the most existence of the term. For example, assume that a search is made for 5 terms (a, b, c, d, and e) and you have a record with two fields:

```
Field 1: a b c
Field 2: d e
```

This record is ranked as if it matched three terms, the maximum number that matched in any single field.

Note the following about `Nterms`:

- The `Nterms` module is only applicable to search modes where results can vary in how many query terms they match. These include `Any`, `Partial`, `Any`, and `AllPartial`. For details on these search modes, see *Search Modes on page 265*.
- `Nterms` treats all matches the same, whether or not they are due to query expansion.

## Phrase

The `Phrase` module states that results containing the user's query as an exact phrase, or a subset of the exact phrase, should be considered more relevant than matches simply containing the user's search terms scattered throughout the text.

Records that have the phrase are ranked higher than records that do not contain the phrase.



### Configuring the Phrase module

The `Phrase` module is configured by editing the `RELRANK_PHRASE` XML element.

You add a `Phrase` module with the `RELRANK_PHRASE` element, which is a sub-element of the `RELRANK_STRATEGY` element.

The following example shows a relevance ranking strategy named PhraseMatch with a `Phrase` module:

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

To configure the Phrase module:

1. In any editor, edit the contents of the RELRANK_STRATEGIES configuration document to add or modify the RELRANK_PHRASE element.

   For details on these elements, see the appendix in this guide. The resulting contents should look similar to the example above.

2. Send the changes to the Oracle Endeca Server using the Configuration Web Service or Integrator ETL.

Details on the three options are explained in the following topic.

*Phrase module options*

*Summary of Phrase option interactions*

### Phrase module options

The Phrase module has a variety of options that you use to customize its behavior.

These options are configured via Boolean attributes:

- The APPROXIMATE attribute sets the use of approximate subphrase/phrase matching.
- The QUERY_EXPANSION attribute determines whether to apply query expansion (spell correction, thesaurus, and stemming).
- The SUBPHRASE attribute enables ranking based on length of subphrases.

These attributes belong to the RELRANK_PHRASE element.

## Approximate matching

Approximate matching provides higher-performance matching, as compared to the standard Phrase module, with somewhat less exact results.

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

The approximate setting is appropriate in cases where the runtime performance of the standard Phrase module is inadequate because of large result contents and/or high site load.

## Query expansion

Applying spelling correction, thesaurus, and stemming adjustments to the original phrase is generically known as query expansion. With query expansion enabled, the Phrase module ranks results that match a phrase's expanded forms in the same stratum as results that match the original phrase.

Consider the following example:

- A thesaurus entry exists that expands "US" to "United States".
- The user queries for "US government".

The query "US government" is expanded to "United States government" for matching purposes, but the `Phrase` module gives a score of two to any results matching "United States government" because the original, unexpanded version of the query, "US government", only had two terms.

## Subphrasing

Subphrasing ranks results based on the length of their subphrase matches. In other words, results that match three terms are considered more relevant than results that match two terms, and so on.

A subphrase is defined as a contiguous subset of the query terms the user entered, in the order that he or she entered them. For example, the query "fax cover sheets" contains the subphrases "fax", "cover", "sheets", "fax cover", "cover sheets", and "fax cover sheets", but not "fax sheets".

Content contained inside nested quotes in a phrase is treated as one term. For example, consider the following phrase:

```
the question is "to be or not to be"
```

The quoted text ("to be or not to be") is treated as one query term, so this example consists of four query terms even though it has a total of nine words.

When subphrasing is not enabled, results are ranked into two strata: those that matched the entire phrase and those that did not.

### Summary of Phrase option interactions

The three configuration settings for the `Phrase` module can be used in a variety of combinations for different effects.

The following matrix describes the behavior of each combination.

| Subphrase | Approximate | Expansion | Description |
|---|---|---|---|
| Off | Off | Off | Default. Ranks results into two strata: those that match the user's query as a whole phrase, and those that do not. |
| Off | Off | On | Ranks results into two strata: those that match the original, or an extended version, of the query as a whole phrase, and those that do not. |
| Off | On | Off | Ranks results into two strata: those that match the original query as a whole phrase, and those that do not.<br>Looks only at the first possible phrase match within each record. |
| Off | On | On | Ranks results into two strata: those that match the original, or an extended version, of the query as a whole phrase, and those that do not.<br>Looks only at the first possible phrase match within each record. |

| Subphrase | Approximate | Expansion | Description |
|---|---|---|---|
| On | Off | Off | Ranks results into N strata, where N equals the length of the query and each result's score equals the length of its matched subphrase. |
| On | Off | On | Ranks results into N strata, where N equals the length of the query and each result's score equals the length of its matched subphrase.<br>Extends subphrases to facilitate matching, but ranks based on the length of the original subphrase (before extension).<br>Note that this combination can have a negative performance impact on query throughput. |
| On | On | Off | Ranks results into N strata, where N equals the length of the query and each result's score equals the length of its matched subphrase.<br>Looks only at the first possible phrase match within each record. |
| On | On | On | Ranks results into N strata, where N equals the length of the query and each result's score equals the length of its matched subphrase.<br>Expands the query to facilitate matching, but ranks based on the length of the original subphrase (before extension).<br>Looks only at the first possible phrase match within each record. |

**Note:** You should only use one `Phrase` module in any given search interface and set all of your options in it.

## Phrase module behavior

This topic describes some aspects of the behavior of the `Phrase` module with other features of the Oracle Endeca Server.

## Effect of search modes

Oracle Endeca Server provides a variety of search modes to facilitate matching during search (`Any`, `All`, `Partial`, and so on). These modes only determine which results match a user's query, they have no effect on how the results are ranked after the matches have been found. Therefore, the `Phrase` module works as described in this section, regardless of search mode. The one exception to this rule is `Boolean`. Phrase, like the other relevance ranking modules, is never applied to the results of Boolean queries.

### Results with multiple matches

If a single result has multiple subphrase matches, either within the same field or in several different fields, the result is slotted into a stratum based on the length of the longest subphrase match.

### Stop words

When using the `Phrase` module, stop words are always treated like non-stop word terms and stratified accordingly.

For example, the query "raining cats and dogs" will result in a rank of two for a result containing "fat cats and hungry dogs" and a rank of three for a result containing "fat cats and dogs" (this example assumes subphrase is enabled).

### Cross-field matches

An entire phrase, or subphrase, must appear in a single field in order for it to be considered a match. In other words, matches created by concatenating fields are not considered by the `Phrase` module.

### Notes about the Phrase module

Keep the following points in mind when using the `Phrase` module:

- If a query contains only one word, then that word constitutes the entire phrase and all of the matching results will be put into one stratum (score = 1). However, the module can rank the results into two strata: one for records that contain the phrase, and a lower-ranking stratum for records that do not contain the phrase.
- Because of the way hyphenated words are positionally indexed, it is recommended to enable subphrase if your results contain hyphenated words.

### Treatment of wildcards with the Phrase module

The `Phrase` module translates each wildcard in a query into a generic placeholder for a single term.

For example, the query "sparkling w* wine" becomes "sparkling * wine" during phrase relevance ranking, where "*" indicates a single term. This generic wildcard replacement causes slightly different behavior depending on whether subphrasing is enabled.

When subphrasing is not enabled, all results that match the generic version of the wildcard phrase exactly are still placed into the first stratum. It is important, however, to understand what constitutes a matching result from the `Phrase` module's point of view.

Consider the search query "sparkling w* wine" with the Any mode enabled. In Any mode, search results only need to contain one of the requested terms to be valid, so a list of search results for this query could contain phrases that look like this:

```
sparkling white wine
sparkling refreshing wine
sparkling wet wine
sparkling soda
wine cooler
```

When phrase relevance ranking is applied to these search results, the `Phrase` module looks for matches to "sparkling * wine" not "sparkling w* wine". Therefore, there are three results—"sparkling white wine", "sparkling

refreshing wine", and "sparkling wet wine"—that are considered phrase matches for the purposes of ranking. These results are placed in the first stratum. The other two results are placed in the second stratum.

When subphrasing is enabled, the behavior becomes a bit more complex. Again, we have to remember that wildcards become generic placeholders and match any single term in a result. This means that any subphrase that is adjacent to a wildcard will, by definition, match at least one additional term (the wildcard). Because of this behavior, subphrases break down differently. The subphrases for "cold sparkling w* wine" break down into the following (note that w* changes to *):

```
cold
sparkling *
* wine
cold sparkling *
sparkling * wine
cold sparkling * wine
```

Notice that the subphrases "sparkling", "wine" and "cold sparkling" are not included in this list. Because these subphrases are adjacent to the wildcard, we know that the subphrases will match at least one additional term. Therefore, these subphrases are subsumed by the "sparkling *", "* wine", and "cold sparkling *" subphrases.

Like regular subphrase, stratification is based on the number of terms in the subphrase, and the wildcard placeholders are counted toward the length of the subphrase. To continue the example above, results that contain “cold” get a score of one, results that contain "sparkling *" get a score of two, and so on. Again, this is the case even if the matching result phrases are different, for example, "sparkling white" and "sparkling soda".

Finally, it is important to note that, while the wildcard can be replaced by any term, a term must still exist. In other words, search results that contain the phrase "sparkling wine" are not acceptable matches for the phrase "sparkling * wine", because there is no term to substitute for the wildcard. Conversely, the phrase "sparkling cold white wine" is also not a match, because each wildcard can be replaced by one, and only one, term. Even when wildcards are present, results must contain the correct number of terms, in the correct order, for them to be considered phrase matches by the `Phrase` module.

## Proximity

Designed primarily for use with unstructured data, the `Proximity` module ranks how close the query terms are to each other in a document by counting the number of intervening words.

Like the `First` module, this module groups its results into variable sized strata, because the difference in significance of an interval of one word and one of two words is usually greater than the difference in significance of an interval of 21 words and 22. If no terms match, the document is placed in the lowest stratum.

Single words and phrases get assigned to the best stratum because there are no intervening words. When the query has multiple terms, `Proximity` behaves as follows:

1. All of the absolute positions for each of the query terms are computed.
2. The smallest range that includes at least one instance of each of the query terms is calculated. This range’s length is given in number of words.

   The score for each document is the strata that contains the difference of the range’s length and the number of terms in the query; smaller differences are better than larger differences.

Under query expansion (that is, stemming, spelling correction, and the thesaurus), the expanded terms are treated as if they were in the query, so the proximity metric is computed using the locations of the expanded terms in the matching document.

For example, if a user searches for *big cats* and a document contains the sentence, "Big Bird likes his cat" (stemming takes *cats* to *cat*), then the proximity metric is computed just as if the sentence were, "Big Bird likes his cats."

`Proximity` scores partially matched queries as if the query only contained the matching terms. For example, if a user searches for *cat dog fish* and a document is partially matched that contains only *cat* and *fish*, then the document is scored as if the query *cat fish* had been entered.

**Note:** `Proximity` does not work with Boolean searches, cross-field matching, or wildcard search. It assigns all such matches a score of zero.

## Spell

The `Spell` module ranks spelling-corrected matches below other kinds of matches.

`Spell` assigns a rank of 0 to matches from spelling correction, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

## Static

The `Static` module assigns a static or constant data-specific value to each search result, depending on the type of search operation performed and depending on optional parameters that can be passed to the module.

For record search operations, the first parameter to the module specifies an attribute, which will define the sort order assigned by the module. The second parameter can be specified as ascending or descending to indicate the sort order to use for the specified attribute.

For example, using the module `Static(Availability,descending)` would sort result records in descending order with respect to their assignments from the Availability standard attribute. Using the module `Static(Title,ascending)` would sort result records in ascending order by their Title standard attribute assignments.

In a catalog application, setting the `Static` module by Price, descending leads to more expensive products being displayed first.

For value search, the first parameter should be specified as `nbins`. Specifying `nbins` causes the static module to sort result values by the number of associated records in the full data set.

## Stem

The `Stem` module ranks matches due to stemming below other kinds of matches.

`Stem` assigns a rank of 0 to matches from stemming, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

## Thesaurus

The `Thesaurus` module ranks matches due to thesaurus entries below other sorts of matches.

`Thesaurus` assigns a rank of 0 to matches from the thesaurus, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

## Weighted Frequency

Like the Frequency module, the Weighted Frequency (`Wfreq`) module scores results based on the frequency of user query terms in the result.

Additionally, the Weighted Frequency module weights the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term. Less frequent query terms (that is, terms that would result in fewer search results) are weighted more heavily than more frequently occurring terms.

The Weighted Frequency module ignores matches due to query expansion (that is, such matches are given a rank of 0).

**Note:** Due to performance issues, it is not recommended to use the Weighted Frequency module with standalone relevance ranking (that is, per-query relevance ranking).

# Relevance ranking strategies

Relevance ranking modules define the primitive search result ordering functions provided by Endeca Server. These primitive modules can be combined to compose more complex ordering behaviors called relevance ranking strategies.

You may also define and apply a strategy that consists of a single module, rather than a group of modules.

You can specify a relevance ranking strategy either in the request issued by the Conversation Web Service, and/or in the `RECSEARCH_CONFIG` configuration XML document.

The scores assigned by a strategy are composed from the scores assigned by its constituent modules. This composite score is constructed so that records are first ordered by the first module. After that, ties are broken by the subsequent modules in order. If any ties remain after all modules have been consulted, they are resolved by the default sort. If after that any ties still remain, the order of records is determined by the system.

Note that the order of results returned for a query where there are multiple text searches with relevance ranking enabled in the query is that the relevance rank of a given record will be the maximum of the relevance ranks of the searches for that record.

Relevance ranking strategies are used in two main contexts in Endeca Server:

- You can configure relevance ranking to a search interface in the `RECSEARCH_CONFIG` configuration document, and send this document to Endeca Server using the Configuration Web Service or Integrator ETL.
- You can specify a relevance ranking strategy for a particular attribute to override the strategy specified for the selected search interface. This allows relevance ranking behavior to be fully customized on a per-query basis. In other words, in a Conversation Web Service request, you can send a per-query relevance ranking strategy. For details, see *Specifying relevance ranking for record and value searches on page 326*.

*Creating relevance ranking strategies*

## Creating relevance ranking strategies

You create relevance ranking strategies by modifying the RELRANK_STRATEGIES configuration document.

All configuration documents are present in the data files of the Oracle Endeca Server, for a particular data domain and its corresponding Dgraph process. You can edit them using the format specified in the *Dgraph Configuration Reference* appendix in this guide. After these documents are edited, you can send them to the Dgraph using the Configuration Web Service or Integrator ETL, thus specifying the configuration you want.

You create a relevance ranking strategy by adding one or more RELRANK_STRATEGY elements to the root RELRANK_STRATEGIES document.

Each RELRANK_STRATEGY element, in turn, contains one or more relevance ranking module elements, such as the RELRANK_INTERP and RELRANK_FIELD module elements in this WineMatch example:

```
<RELRANK_STRATEGIES>
  <RELRANK_STRATEGY NAME="WineMatch">
    <RELRANK_INTERP/>
    <RELRANK_STATIC NAME="Flavors" ORDER="ASCENDING"/>
    <RELRANK_FIELD/>
  </RELRANK_STRATEGY>
</RELRANK_STRATEGIES>
```

Keep in mind that the order of the module sub-elements defines the order in which the strategies are applied to the search results.

To create a relevance ranking strategy:

1. Edit the contents of the RELRANK_STRATEGIES document to add or modify the RELRANK_STRATEGY elements.

   For details on these elements, see the appendix in this guide. The resulting contents of the edited document should look similar to the example above.
2. Send the RELRANK_STRATEGIES document to the Dgraph using the Configuration Web Service or Integrator ETL.

The new relevance ranking strategy can now be added to a search interface.

# Implementing relevance ranking

You can create and control relevance ranking for both record search and value search at a system-default level.

You can apply record search relevance ranking as you are creating a search interface, or afterwards. A search interface is a named group of at least one attribute. You create search interfaces so you can apply behavior such as relevance ranking across a group.

You set the search interface for record search by modifying the RECSEARCH_CONFIG configuration document and sending it to the Oracle Endeca Server with the Configuration Web Service, or using a connector in Integrator ETL. For information about configuring relevance ranking in search interfaces, see *Search Interfaces on page 252*.

For information on using relevance ranking for value search, see *Search Interfaces on page 252*.

*Adding a Static module*

*Ranking order for Field and Maximum Field modules*

## Adding a Static module

Keep the following in mind when you add a Static module to the ranking strategy.

The Static module is the only one that you can add multiple times. When you add a Static module, be sure to set the two Static attributes:

- The `NAME` attribute sets the name of an attribute that is used for static relevance ranking.
- The `ORDER` attribute specifies how records should be sorted with respect to the specified Endeca attribute sets. The two values are `ASCENDING` and `DESCENDING`.

## Ranking order for Field and Maximum Field modules

The Field and Maximum Field modules rank results based on which attribute member of the selected search interface caused the match.

In a search interface, higher relevance-ranked values (in the `RELEVANCE_RANK` attribute of the `MEMBER_NAME` element) correspond to greater importance. This behavior means that the Field and Maximum Field modules will score results caused by higher-ranked Endeca attributes ahead of those caused by lower-ranked attributes.

In this example:

```
<MEMBER_NAME RELEVANCE_RANK="2">P_Type</MEMBER_NAME>
<MEMBER_NAME RELEVANCE_RANK="1">P_Description</MEMBER_NAME>
```

records tagged with P_Type will be ranked ahead of records with P_Description.

To change the relevance ranking behavior for these modules, change the `RELEVANCE_RANK` integer settings as appropriate. For example, change P_Description to a `RELEVANCE_RANK="2"` setting and P_Type to a `RELEVANCE_RANK="1"` setting.

## How relevance ranking score ties between search interfaces are resolved

In the case of multiple search interfaces and relevance ranking score ties, ties are broken based on the relevance ranking sort strategy of the search interface with the highest relevance ranking score for a given record.

If two different records belong to different search interfaces, the record from the search interface specified earlier in the query comes first.

## Implementing relevance ranking for value search

You can define a system-default relevance ranking strategy for value search operations.

To define a system-default relevance ranking strategy for value search operations, modify the `RELRANK_STRATEGY` attribute of the `DIMSEARCH_CONFIG` configuration document. To do so, create a text file

with the configuration document and send it to Endeca Server, using the Configuration Web Service or Integrator ETL.

The RELRANK_STRATEGY attribute specifies the name of a relevance ranking strategy for value search. The content of this attribute should be a relevance ranking string, as in this example:

```
<DIMSEARCH_CONFIG FILTER_FOR_ANCESTORS="FALSE" RELRANK_STRATEGY="interp,exact"/>
```

The default ranking strategy for value search operations, which is applied if you do not make any changes to it, is:

```
interp,exact,static
```

## Specifying relevance ranking for record and value searches

You can specify a relevance ranking strategy for both record search queries and value search queries in the Conversation Web Service.

Both types of queries let you specify either an existing relevance ranking strategy or the names of the relevance ranking modules.

### Record search

For record search, the RelevanceRankingStrategy attribute of the TextSearchFilter element lets you specify a relevance ranking strategy for the query, as in this example:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/3/0">
   <State>
      <TextSearchFilter Key="Flavors" Mode="AllPartial" RelevanceRankingStrategy="exact" Language
="en">
        grapefruit
      </TextSearchFilter>
   </State>
   <RecordListConfig Id="RecList" MaxPages="10">
      <Column>Flavors</Column>
      <RecordsPerPage>5</RecordsPerPage>
   </RecordListConfig>
   <SearchAdjustmentConfig Id="SpellCorrect"/>
</Request>
```

For more information on the TextSearchFilter, see *Record search filter on page 247*.

### Value search

For value search, the RelevanceRankingStrategy attribute of the ValueSearchConfig type allows you to specify a relevance ranking strategy for the query.

```
<Request>
   <State/>
   <ValueSearchConfig Id="ProdSearch" MaxPerProperty="5"
        RelevanceRankingStrategy="static (nbins,descending)" Mode="Any" Language="en">
      <SearchTerm>racks</SearchTerm>
      <RestrictToProperties>
         <Property>ProductCategory</Property>
         <Property>BikeRacks</Property>
      </RestrictToProperties>
   </ValueSearchConfig>
</Request>
```

For more information on the ValueSearchConfig type, see *Value search query format on page 260*.

# Relevance ranking sample scenarios

This section contains two examples of relevance ranking behavior to further illustrate the capabilities of this feature.

In the first example, we first look at the effects of various relevance ranking strategies on a small sample data set that supports record search, examining the range of possible result orderings possible using only a limited set of ranking modules.

In the second example, we look at how adding a simple relevance ranking strategy can affect user results in the reference implementation.

**Note:** These extremely simple scenarios are provided for illustrative purposes only. For more realistic examples, see *Recommended strategies on page 330*.



## Example 1: Using a small data set

This scenario shows the effects of various relevance ranking strategies on a small data set.

This example illustrates the richness of relevance ranking tuning possible with the modular relevance ranking system of the Oracle Endeca Server. Using two modules on a data set of three records, we found that all four possible combinations of the modules into strategies resulted in different orderings, all of which were different from the default ordering.

The example uses the following example record set:

| Record | Title attribute | Author attribute |
|---|---|---|
| 1 | Great Short Stories | Mark Twain and other authors |
| 2 | Mark Twain | William Lyon Phelps |
| 3 | Tom Sawyer | Mark Twain |

### Creating the search interface

In a text editor, we have defined a search interface named Books, which contains both Title and Author standard attributes. The relevance rank is determined by the order in which the Endeca attributes appear in the members list.

Assume that we have not defined an explicit default sort order for the records, in which case their default order is determined by the system.

### Without relevance ranking

Suppose that the user enters a record search query against the Books search interface for *Mark Twain*. Clearly all three of the records are hits, because each record has at least one searchable attribute value containing at least one occurrence of both the words Mark and Twain. But in what order should the results be

presented to the user? Without relevance ranking enabled, the results will be returned in their default order: 1, 2, 3.

If relevance ranking were enabled, the order depends on the relevance ranking strategy selected.

### With an Exact ranking strategy

Suppose we have selected the `Exact` relevance ranking strategy, either by assigning this as the default strategy for the Books search interface or by using query-level search options.

In this case, the order of results would be based only on whether results were Exact, Phrase, or other matches. Because records 2 and 3 have attributes whose complete values exactly match the user query *Mark Twain*, these results would be returned ahead of record 1, with the tie being broken by the default sort set by the system (remember that we have not defined a default sort).

### With a Field ranking strategy

Now, assume that we have selected the `Field` relevance ranking strategy.

The order of results would be based only on which Endeca attribute caused the match, with Author matches being prioritized over Title matches. Because records 1 and 3 match on Author, these are returned ahead of record 2 (again, with ties broken by the default sort imposed by the system).

### With a Field,Exact ranking strategy

Now, consider using a combination of these two strategies: `Field,Exact`.

In this case, the primary sort is determined by the first module, `Field`, which again dictates that records 1 and 3 should be returned ahead of record 2. But in this case, the Field tie between records 1 and 3 is resolved by the `Exact` module, which prioritizes record 3 ahead of record 1. Thus, the order of results returned is: 3, 1, 2.

### With an Exact,Field ranking strategy

Finally, consider combining the same two modules but in a different priority order: `Exact,Field`.

In this case, the primary sort is determined by the `Exact` module, which again prioritizes records 2 and 3 ahead of record 1. In this case, the `Exact` tie between records 2 and 3 is resolved by the `Field` module, which orders record 3 ahead of record 2 because record 3 is an Author match. Thus, the order of results returned is: 3, 2, 1.

## Example 2: UI reference implementation

This scenario shows how adding a relevance ranking module can change the order of the returned records.

This example, which is somewhat more realistically scaled, uses a wine data set. It demonstrates how relevance ranking can affect the results displayed to your users.

In this scenario, we use the thesaurus and relevance ranking features to enable end users' access to Flavor results similar to the one they searched on, while still seeing exact matches first.

First, we establish the following two-way thesaurus entries:

```
<THESAURUS>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>cab</THESAURUS_FORM>
    <THESAURUS_FORM>cabernet</THESAURUS_FORM>
```

```
  </THESAURUS_ENTRY>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>cinnamon</THESAURUS_FORM>
    <THESAURUS_FORM>spice</THESAURUS_FORM>
    <THESAURUS_FORM>nutmeg</THESAURUS_FORM>
  </THESAURUS_ENTRY>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>tangy</THESAURUS_FORM>
    <THESAURUS_FORM>tart</THESAURUS_FORM>
    <THESAURUS_FORM>sour</THESAURUS_FORM>
    <THESAURUS_FORM>vinegary</THESAURUS_FORM>
  </THESAURUS_ENTRY>
  <THESAURUS_ENTRY>
    <THESAURUS_FORM>dusty</THESAURUS_FORM>
    <THESAURUS_FORM>earthy</THESAURUS_FORM>
  </THESAURUS_ENTRY>
</THESAURUS>
```

Before applying these thesaurus equivalencies, if we search on the Dusty flavor, 83 records are returned, and if we search on the Earthy flavor, 3,814 records are returned.

After applying these thesaurus equivalencies, if we search on the Dusty attribute, results for both Dusty and Earthy are returned. (Because some records are flagged with both the Dusty and Earthy descriptors, the number of records is not an exact total of the two.)

| **Wine (by order returned)** | **Relevant attribute** |
|---|---|
| A Tribute Sonoma Mountain | Earthy |
| Against the Wall California | Earthy |
| Aglianico Irpinia Rubrato | Dusty |
| Aglianico Sannio | Earthy |

Because the application is sorting on Name in ascending order, the Dusty and Earthy results are intermingled. That is, the first two results are for Earthy and the third is for Dusty, even though we searched on Dusty, because the two Earthy records came before the Dusty one when the records were sorted in alphabetical order.

Now, suppose that while we want our users to see the synonymous entries, we want records that exactly match the search term Dusty to be returned first. We therefore would use the Interpreted ranking module to ensure that outcome.

| **Wine (by order returned)** | **Relevant attribute** |
|---|---|
| Aglianico Irpinia Rubrato | Dusty |
| Bandol Cuvee Speciale La Miguoa | Dusty |
| Beaujolais-Villages Reserve du Chateau de Montmelas | Dusty |
| Beauzeaux Winemaker’s Collection Napa Valley | Dusty |

With the Interpreted ranking strategy, the results are different. When we search on Dusty, we see the records that matched for Dusty sorted in alphabetical order, followed by those that matched for Earthy. The wine Aglianico Irpinia Rubrato, which was returned third in the previous example, is now returned first.

# Recommended strategies

This section provides some recommended strategies that depend on the implementation type.

Relevance ranking behavior is complex and powerful and requires careful, iterative development. Typically, selection of the ideal relevance ranking strategy for a given application depends on extensive experimentation during application development. The set of possible result ranking strategies is extremely rich, and because setting ranking strategies is highly dependent on the quantity and type of data you are working with, a strategy that works well in one situation could be unsatisfactory in another.

For this reason, this documentation provides recommended strategies for different types of implementations and suggests that you use them as a point of departure in creating your own strategies. The following sections describe recommended general strategies for each product in detail.

### Testing your strategies

When testing your own strategies, it is a good idea to try searching on diverse examples: single word terms, multi-word terms that you know are an exact match for records in your data, and multi-word terms that contain additional words to the ones in your data. In this way you will see the full range of relevance ranking effects.

*Recommended strategy for retail catalog data*

*Recommended strategy for document repositories*

## Recommended strategy for retail catalog data

This topic describes a good starting strategy to try if you are a retailer working with a catalog data set.

The strategy assumes the following:

- The search mode is `AllPartial`. By using this mode, you ensure that a user's search would return a two-words-out-of-five match as well as a four-words-out-of-five match, just at a lower priority.
- The strategy is based on a search interface with members such as Category, Name, and Description, in that order. The order is significant because a match on the first member ranks more highly than a cross-field match or match on the second or third member. For details, see *Search Interfaces on page 252*.

The strategy is as follows:

- `NTerms`
- `MaxField`
- `Glom`
- `Exact`
- `Static`

The modules in this strategy work like this:

1. `NTerms`, the first module, ensures that in a multi-word search, the more words that match the better.

2. `MaxField` puts cross-field matches as high in priority as possible, to the point where they could tie with non-cross-field matches.
3. The next module, `Glom`, decomposes cross-field matches, effectively breaking any ties resulting from `MaxField`. Together, `MaxField` and `Glom` provide the proper ordering, depending upon what matched.
4. Applying the `Exact` module means that an exact match in a highly-ranked member of the search interface is placed higher than a partial or cross-field match.
5. Optionally, the `Static` module can be used to sort remaining ties by criteria such as Price or SalesRank.

## Recommended strategy for document repositories

This topic describes a good starting strategy to try if you are working with a document repository.

The strategy assumes the following:

- The search mode is `AllPartial`. By using this mode, you ensure that a user's search would return a two-words-out-of-five match as well as a four-words-out-of-five match, just at a lower priority.
- The strategy is based on a search interface with members such as Title, Summary, and DocumentText, in that order. The order is significant because a match on the first member ranks more highly than a cross-field match or match on the second or third member.

The strategy is as follows:

- `NTerms`
- `MaxField`
- `Glom`
- `Phrase` (with or without approximate matching enabled)
- `Static`

The modules in this strategy work like this:

1. `NTerms`, the first module, ensures that in a multi-word search, the more words that match the better.
2. `MaxField` puts cross-field matches as high in priority as possible, to the point where they could tie with non-cross-field matches.
3. The next module, `Glom`, decomposes cross-field matches, effectively breaking any ties resulting from `MaxField`. Together, `MaxField` and `Glom` provide the proper ordering, depending upon what matched.
4. Applying the `Phrase` module ensures that results containing the user's query as an exact phrase are given a higher priority than matching containing the user's search terms sprinkled throughout the text.
5. Optionally, the `Static` module can be used to sort the remaining ties by criteria such as ReleaseDate or Popularity.

# Performance impact of relevance ranking

Relevance ranking can impose a significant computational cost in the context of affected search operations (that is, operations where relevance ranking is actually enabled).

You can minimize the performance impact of relevance ranking in your implementation by making module substitutions when appropriate, and by ordering the modules you do select sensibly within your relevance ranking strategy.

## Making module substitutions

Because of the linear cost of relevance ranking in the size of the result set, the actual cost of relevance ranking depends heavily on the set of ranking modules used. In general, modules that do not perform text evaluation introduce significantly lower computational costs than text-matching-oriented modules.

Although the relative cost of the various ranking modules is dependent on the nature of your data and the number of records, the modules can be roughly grouped into four tiers:

- `Exact` is very computationally expensive.
- `Proximity`, `Phrase` with Subphrase or Query Expansion options specified, and `First` are all high-cost modules, presented in the order of decreasing cost.
- `WFreq` can also be costly in some situations.
- The remaining modules (`Static`, `Phrase` with no options specified, `Freq`, `Spell`, `Glom`, `Nterms`, `Interp`, `Numfields`, `Maxfield`, and `Field`) are generally relatively cheap.

In order to maximize the performance of your relevance ranking strategy, consider a less expensive way to get similar results. For example, replacing `Exact` with `Phrase` may improve performance in some cases with relatively little impact on results.

**Note:** Choose the set of modules used for relevance ranking most carefully when the data set is large or contains large/offline file content that is used for search operations.

## Ordering modules sensibly

Relevance ranking modules are only evaluated as needed. When higher-priority ranking modules determine the order of records, lower-priority modules do not need to be calculated. This can have a dramatic impact on performance when higher-cost modules have a lower priority than a lower-cost module.

While you have the freedom to order modules as you like, for best performance, make sure that the cheaper modules are placed before the more expensive ones in your strategy.

# Part VII

## References

Chapter 32

# Dgraph Configuration Reference

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



## XML elements

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



### COMMENT

The COMMENT element provides a way to add inline XML comments.

#### Format

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

#### Attributes

The COMMENT element has no attributes.

### Sub-elements

The COMMENT element has no sub-elements.

#### Example

This example includes an informational comment.

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

## DIMNAME

The DIMNAME element specifies the name of a managed attribute.

### Format

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

### Attributes

The DIMNAME element has no attributes.

### Sub-elements

The DIMNAME element has no sub-elements.

#### Example

This example shows the name of a managed attribute.

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

## PROP

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

### Format

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

### Attributes

The PROP element has the following attributes.

#### NAME

Identifies the name of the standard attribute.

### Sub-elements

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

#### Example

This example shows a standard attribute name.

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

## PROPNAME

The PROPNAME element represents an Endeca standard attribute.

### Format

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

### Attributes

The PROPNAME element has no attributes.

### Sub-elements

The PROPNAME element has no sub-elements.

### Example

This example shows a standard attribute name.

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

## PVAL

The PVAL element represents a standard attribute value.

### Format

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

### Attributes

The PVAL element has no attributes.

### Sub-elements

The PVAL element has no sub-elements.

### Example

This example shows a standard attribute value.

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

# Dimsearch_config elements

The Dimsearch_config element controls how value searches behave.

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

*DIMSEARCH_CONFIG*

## DIMSEARCH_CONFIG

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

### Format

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

### Attributes

The DIMSEARCH_CONFIG element has the following attributes.

#### FILTER_FOR_ANCESTORS

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

#### RELRANK_STRATEGY

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

### Sub-elements

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

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

### Example

This example shows a configuration that displays ancestor attribute values.

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

# Recsearch_config elements

The Recsearch_config element configures record search.

*RECSEARCH_CONFIG*

## RECSEARCH_CONFIG

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

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

### Format

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

### Attributes

The RECSEARCH_CONFIG element has the following attributes.

**WORD_INTERP**

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

### Sub-elements

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

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

### Example

This example shows the configuration for a business implementation.

```
<RECSEARCH_CONFIG>
   <SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="All" NAME="All">
     <MEMBER_NAME RELEVANCE_RANK="4">ProductType</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="3">Name</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="2">Region</MEMBER_NAME>
     <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
   </SEARCH_INTERFACE>
</RECSEARCH_CONFIG>
```

# Relrank_strategies elements

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

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

For more information, see *Relevance Ranking on page 310*.

*RELRANK_EXACT*

*RELRANK_FIELD*

*RELRANK_FIRST*

*RELRANK_FREQ*

*RELRANK_GLOM*

*RELRANK_INTERP*

*RELRANK_MAXFIELD*

*RELRANK_MODULE*

*RELRANK_NTERMS*

*RELRANK_NUMFIELDS*

*RELRANK_PHRASE*

## RELRANK_EXACT

The RELRANK_EXACT element implements the `Exact` relevance ranking module.

This module groups results into strata based on how well they match a query string, with the highest stratum containing results that match the user's query exactly.

### Format

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

### Attributes

The RELRANK_EXACT element has no attributes.

### Sub-elements

The RELRANK_EXACT element has no sub-elements.

### Example

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

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

## RELRANK_FIELD

The RELRANK_FIELD element implements the `Field` relevance ranking module.

This module assigns a score to each result based on the static rank of the standard attribute or managed attribute member of the search interface that caused the document to match the query.

### Format

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

### Attributes

The RELRANK_FIELD element has no attributes.

### Sub-elements

The RELRANK_FIELD element has no sub-elements.

### Example

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

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

## RELRANK_FIRST

The RELRANK_FIRST element implements the `First` relevance ranking module.

This module ranks documents by how close the query terms are to the beginning of the document. This module takes advantage of the fact that the closer something is to the beginning of a document, the more likely it is to be relevant.

### Format

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

### Attributes

The RELRANK_FIRST element has no attributes.

### Sub-elements

The RELRANK_FIRST element has no sub-elements.

### Example

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

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

# RELRANK_FREQ

The RELRANK_FREQ element implements the Frequency (`Freq`) relevance ranking module.

This module provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text.

## Format

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

## Attributes

The RELRANK_FREQ element has no attributes.

## Sub-elements

The RELRANK_FREQ element has no sub-elements.

## Example

This example implements a strategy called Frequency.

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

# RELRANK_GLOM

The RELRANK_GLOM element implements the `Glom` relevance ranking module.

This module ranks single-field matches ahead of cross-field matches.

## Format

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

## Attributes

The RELRANK_GLOM element has no attributes.

## Sub-elements

The RELRANK_GLOM element has no sub-elements.

## Example

This example implements a strategy called Single_Field.

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

# RELRANK_INTERP

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

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

## Format

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

## Attributes

The RELRANK_INTERP element has no attributes.

## Sub-elements

The RELRANK_INTERP element has no sub-elements.

## Example

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

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

# RELRANK_MAXFIELD

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

This module is similar to the `Field` strategy module, except it selects the static field-specific score of the highest-ranked field that contributed to the match.

## Format

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

## Attributes

The RELRANK_MAXFIELD element has no attributes.

## Sub-elements

The RELRANK_MAXFIELD element has no sub-elements.

### Example

This example implements a strategy called High_Rank.

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

## RELRANK_MODULE

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

### Format

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

### Attributes

The RELRANK_MODULE element has the following attribute.

**NAME**

NAME refers to another defined relevance ranking module.

### Sub-elements

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

### Example

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

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

## RELRANK_NTERMS

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

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

This module applies only to search modes where the number of results can vary in how many query terms they match. These search modes include `Partial`, `Any`, `AllPartial`, and `AllAny`.

### Format

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

### Attributes

The RELRANK_NTERMS element has no attributes.

### Sub-elements

The RELRANK_NTERMS element has no sub-elements.

### Example

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

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

## RELRANK_NUMFIELDS

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

This module ranks results based on the number of fields in the associated search interface in which a match occurs.

### Format

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

### Attributes

The RELRANK_NUMFIELDS element has no attributes.

### Sub-elements

The RELRANK_NUMFIELDS element has no sub-elements.

### Example

This example implements the `Numfields` relevance ranking module.

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

# RELRANK_PHRASE

The RELRANK_PHRASE element implements the `Phrase` relevance ranking module.

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

## Format

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

## Attributes

The RELRANK_PHRASE element has the following attributes.

### SUBPHRASE

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

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

### APPROXIMATE

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

### QUERY_EXPANSION

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

## Sub-elements

The RELRANK_PHRASE element has no sub-elements.

## Example

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

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

# RELRANK_PROXIMITY

The RELRANK_PROXIMITY element implements the `Proximity` relevance ranking module.

This module ranks how close the query terms are to each other in a document by counting the number of intervening words.

## Format

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

## Attributes

The RELRANK_PROXIMITY element has no attributes.

## Sub-elements

The RELRANK_PROXIMITY element has no sub-elements.

## Example

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

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

# RELRANK_SPELL

The RELRANK_SPELL element implements the `Spell` relevance ranking module.

This module ranks matches that do not require spelling correction ahead of spelling-corrected matches.

## Format

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

## Attributes

The RELRANK_SPELL element has no attributes.

## Sub-elements

The RELRANK_SPELL element has no sub-elements.

## Example

This example implements a strategy called TrueMatch.

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

```
   <RELRANK_SPELL/>
</RELRANK_STRATEGY>
```

# RELRANK_STATIC

The RELRANK_STATIC element implements the `Static` relevance ranking module.

This module assigns a constant score to each result, depending on the type of search operation performed.

## Format

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

## Attributes

The RELRANK_STATIC element has the following attributes.

**NAME**

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

**ORDER**

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

## Sub-elements

The RELRANK_STATIC element has no sub-elements.

## Example

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

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

# RELRANK_STRATEGIES

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

Each strategy is specified in a RELRANK_STRATEGY element.

## Format

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

### Attributes

The RELRANK_STRATEGIES element has no attributes.

### Sub-elements

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

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

### Example

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

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

## RELRANK_STRATEGY

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

Each sub-element of RELRANK_STRATEGY represents a specific type of strategy. If you want several relevance ranking strategies to affect search results, then the order of the sub-elements, which represent the strategies, is significant. The order of the sub-elements defines the order in which the strategies are applied to the search results.

### Format

```
<!ELEMENT RELRANK_STRATEGY (
      RELRANK_STATIC
    | RELRANK_EXACT
    | RELRANK_PHRASE
    | RELRANK_APPROXPHRASE
    | RELRANK_GLOM
    | RELRANK_SPELL
    | RELRANK_FIELD
    | RELRANK_MAXFIELD
```

```
    | RELRANK_INTERP
    | RELRANK_FREQ
    | RELRANK_WFREQ
    | RELRANK_NTERMS
    | RELRANK_PROXIMITY
    | RELRANK_FIRST
    | RELRANK_NUMFIELDS
    | RELRANK_MODULE
    )+>
<!ATTLIST RELRANK_STRATEGY
    NAME               CDATA            #REQUIRED
>
```

## Attributes

The RELRANK_STRATEGY element has the following attribute.

**NAME**

Specifies the name of the strategy.

## Sub-elements

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

| Sub-element | Brief description |
|---|---|
| RELRANK_STATIC | Assigns a constant score to each result, depending on the type of search operation perform. |
| RELRANK_EXACT | Groups results into strata based on how well they match the query string, with the highest stratum containing results that match the user's query exactly. |
| RELRANK_PHRASE | Considers results containing the user’s query as an exact phrase, or a subset of the exact phrase, to be more relevant than matches simply containing the user’s search terms scattered throughout the text. |
| RELRANK_APPROXPHRASE | Not supported. |
| RELRANK_GLOM | Ranks single-field matches ahead of cross-field matches. |
| RELRANK_SPELL | Ranks true matches ahead of spelling-corrected matches. |
| RELRANK_FIELD | Assigns a score to each result based on the static rank of the attribute member of the search interface that caused the document to match the query. |
| RELRANK_MAXFIELD | Similar to the `Field` strategy, except it selects the static field-specific score of the highest-ranked field that contributed to the match. |

| Sub-element | Brief description |
|---|---|
| RELRANK_INTERP | A general-purpose strategy that assigns a score to each result document based on the query processing techniques used to obtain the match.<br>Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching. |
| RELRANK_FREQ | Provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text. |
| RELRANK_WFREQ | Scores results based on the frequency of user query terms in the result, while weighing the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term. |
| RELRANK_NTERMS | Assigns a score to each result record based on the number of query terms that the result record matches. |
| RELRANK_PROXIMITY | Ranks how close the query terms are to each other in a document by counting the number of intervening words. |
| RELRANK_FIRST | Ranks documents by how close the query terms are to the beginning of the document. |
| RELRANK_NUMFIELDS | Ranks results based on the number of fields in the associated search interface in which a match occurs. |
| RELRANK_MODULE | Used to refer to other RELRANK elements and compose them into cohesive strategies. |

### Example

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

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

## RELRANK_WFREQ

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

This module scores results based on the frequency of user query terms in the result, while weighing the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term.

### Format

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

### Attributes

The RELRANK_WFREQ element has no attributes.

### Sub-elements

The RELRANK_WFREQ element has no sub-elements.

### Example

This example implements a strategy called Term_Freq.

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

# Search_interface elements

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

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



## MEMBER_NAME

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

For information on search interfaces, see *Search Interfaces on page 252*.

### Format

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

### Attributes

The MEMBER_NAME element has the following attributes.

**RELEVANCE_RANK**

RELEVANCE_RANK is an unsigned integer that specifies the relevance rank of a match on the specified Endeca standard or managed attribute. Higher numbers correspond to greater importance.

**SNIPPET_SIZE**

The presence of SNIPPET_SIZE enables snippeting for a MEMBER_NAME and the value of SNIPPET_SIZE specifies the maximum number of words a snippet can contain. Omitting this attribute or setting its value equal to zero disables snippeting. For more information, see *Snippeting in Record Searches on page 281*.

### Sub-elements

The MEMBER_NAME element has no sub-elements.

### Example

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

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="NEVER"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="ProductRelRank" NAME="ProductSearch">
   <MEMBER_NAME RELEVANCE_RANK="4">ProductType</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="3">Name</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="2">SalesRegion</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="1" SNIPPET_SIZE="10">Description</MEMBER_NAME>
</SEARCH_INTERFACE>
```

## PARTIAL_MATCH

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

For details about searching and search modes, see the information on search features in this guide.

### Format

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

### Attributes

The PARTIAL_MATCH element has the following attributes.

**MIN_WORDS_INCLUDED**

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

**MAX_WORDS_OMITTED**

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

### Sub-elements

The PARTIAL_MATCH element has no sub-elements.

### Example

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

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

## SEARCH_INTERFACE

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

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

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

### Format

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

### Attributes

The SEARCH_INTERFACE element has the following attributes.

**NAME**

A unique name for this search interface.

**DEFAULT_RELRANK_STRATEGY**

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

**CROSS_FIELD_RELEVANCE_RANK**

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

**CROSS_FIELD_BOUNDARY**

Specifies when the search engine should try to match search queries across standard attribute/managed attribute boundaries, but within the members of the SEARCH_INTERFACE:

- If its value is set to ON_FAILURE, the search engine will only try to match queries across standard attribute/managed attribute boundaries if it fails to find any match within a single standard attribute/managed attribute.
- If its value is set to ALWAYS, the engine will always look for matches across standard attribute/managed attribute boundaries, in addition to matches within a standard attribute/managed attribute.
- If its value is set to NEVER, the engine will not look across boundaries for matches. This is the default.

**STRICT_PHRASE_MATCH**

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

## Sub-elements

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

| Sub-element | Brief description |
|---|---|
| MEMBER_NAME | Specifies the name of an attribute that is part of a SEARCH_INTERFACE. |
| PARTIAL_MATCH | Specifies if partial query matches should be supported for the SEARCH_INTERFACE that contains this element. |

## Example

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

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="ALWAYS"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="All" NAME="AllFields">
   <MEMBER_NAME RELEVANCE_RANK="4">ProductType</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="3">ProductName</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="2">SalesRegion</MEMBER_NAME>
   <MEMBER_NAME RELEVANCE_RANK="1">Description</MEMBER_NAME>
</SEARCH_INTERFACE>
```

# Stop_words elements

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

Each stop is specified in a STOP_WORD element.



## STOP_WORD

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

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

### Format

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

### Attributes

The STOP_WORD element has no attributes.

### Sub-elements

The STOP_WORD element has no sub-elements.

### Example

This example shows a common set of stop words.

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

## STOP_WORDS

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

Each stop word is represented by a STOP_WORD element.

### Format

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

### Attributes

The STOP_WORDS element has no attributes.

### Sub-elements

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

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

### Example

This example shows a common set of stop words.

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

# Thesaurus elements

The Thesaurus elements contain thesaurus entries for your application.

Thesaurus entries provide a means to account for alternate forms of a user's query. These entries provide concept-level mappings between words and phrases. For details, see *Stemming and Thesaurus on page 303*.

*THESAURUS*

*THESAURUS_ENTRY*

*THESAURUS_ENTRY_ONEWAY*

*THESAURUS_FORM*

*THESAURUS_FORM_FROM*

*THESAURUS_FORM_TO*

## THESAURUS

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

THESAURUS is the root element for all thesaurus entries.

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

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

## Format

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

## Attributes

The THESAURUS element has no attributes.

## Sub-elements

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

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

## Example

This example shows the thesaurus entries for an application.

```
<THESAURUS>
   <THESAURUS_ENTRY>
     <THESAURUS_FORM>france</THESAURUS_FORM>
     <THESAURUS_FORM>french</THESAURUS_FORM>
   </THESAURUS_ENTRY>
   <THESAURUS_ENTRY_ONEWAY>
     <THESAURUS_FORM_FROM>bike accessory</THESAURUS_FORM_FROM>
     <THESAURUS_FORM_TO>helmet</THESAURUS_FORM_TO>
     <THESAURUS_FORM_TO>pannier</THESAURUS_FORM_TO>
     <THESAURUS_FORM_TO>tire</THESAURUS_FORM_TO>
   </THESAURUS_ENTRY_ONEWAY>
</THESAURUS>
```

# THESAURUS_ENTRY

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

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

## Format

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

## Attributes

The THESAURUS_ENTRY element has no attributes.

## Sub-elements

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

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

## Example

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

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

# THESAURUS_ENTRY_ONEWAY

A THESAURUS_ENTRY_ONEWAY element specifies a single-direction mapping.

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

## Format

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

### Attributes

The THESAURUS_ENTRY_ONEWAY element has no attributes.

### Sub-elements

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

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

### Example

In this example, searches for `bike accessory` would return hits for `bike accessory` as well as for `helmet`, `pannier`, and `tire`. Since the equivalence is one-way, more specific searches such as `helmet` or `pannier` would not return results for the more general concept `bike accessory`.

```
<THESAURUS_ENTRY_ONEWAY>
  <THESAURUS_FORM_FROM>bike accessory</THESAURUS_FORM_FROM>
  <THESAURUS_FORM_TO>helmet</THESAURUS_FORM_TO>
  <THESAURUS_FORM_TO>pannier</THESAURUS_FORM_TO>
  <THESAURUS_FORM_TO>tire</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

## THESAURUS_FORM

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

### Format

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

### Attributes

The THESAURUS_FORM element has no attributes.

### Sub-elements

The THESAURUS_FORM element has no sub-elements.

### Example

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

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

```
   </THESAURUS_ENTRY>
</THESAURUS>
```

## THESAURUS_FORM_FROM

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

### Format

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

### Attributes

The THESAURUS_FORM_FROM element has no attributes.

### Sub-elements

The THESAURUS_FORM_FROM element has no sub-elements.

### Example

In this example, searches for `bike part` would return hits for `bike part` as well as for `handlebar` and `derailleur`. Because the equivalence is one-way, more specific searches such as `handlebar` or `derailleur` would not return results for the more general concept `bike part`.

```
<THESAURUS_ENTRY_ONEWAY>
   <THESAURUS_FORM_FROM>bike part</THESAURUS_FORM_FROM>
   <THESAURUS_FORM_TO>handlebar</THESAURUS_FORM_TO>
   <THESAURUS_FORM_TO>derailleur</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

## THESAURUS_FORM_TO

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

### Format

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

### Attributes

The THESAURUS_FORM_TO element has no attributes.

### Sub-elements

The THESAURUS_FORM_TO element has no sub-elements.

## Example

In this example, searches for `bike part` would return hits for `bike part` as well as for `handlebar` and `derailleur`. Because the equivalence is one-way, more specific searches such as `handlebar` or `derailleur` would not return results for the more general concept `bike part`.

```
<THESAURUS_ENTRY_ONEWAY>
   <THESAURUS_FORM_FROM>bike part</THESAURUS_FORM_FROM>
   <THESAURUS_FORM_TO>handlebar</THESAURUS_FORM_TO>
   <THESAURUS_FORM_TO>derailleur</THESAURUS_FORM_TO>
</THESAURUS_ENTRY_ONEWAY>
```

Chapter 33

# Stop Words

Stop words are words that are set to be ignored by the Oracle Endeca Server.



## About stop words

Typically, common words (like "the") are included in the stop word list. In addition, the stop word list can include the extraneous words contained in a typical question, allowing the query to focus on what the user is really searching for.

Stop words must be single words only, and cannot contain any non-searchable characters. If more than one word is entered as a stop word, neither the individual words nor the combined phrase will act as a stop word. Non-searchable characters within a stop word will also cause this behavior. Entering "full-bodied" as a stop word acts just as if you had entered "full bodied", and does not have any effect on searches.

**Note:** Stop words are supported only for searches that are marked with the `unknown` language identifier.

Stop words are counted in any search mode that calculates results based on number of matching terms. However, Endeca Server reduces the minimum term match and maximum word omit requirement by the number of stop words contained in the query.

**Note:** Did You Mean can in some cases correct a word to one on the stop words list.

## List of suggested stop words

The following table provides a list of words that are commonly added to the stop word list; you may find it useful as a point of departure when you configure a list for your application.

In addition to some or all of the words listed below, you might want to add terms that are prevalent in your data set. For example, if your data consists of lists of books, you might want to add the word book itself to the stop word list, since a search on that word would return an impracticably large set of records.

You can add stop words to Endeca Server using the Configuration Web Service, or using Integrator ETL.

| | | | | | |
|---|---|---|---|---|---|
| a | any | for | is | show | where |
| about | are | from | me | the | why |

| | | | | | |
|---|---|---|---|---|---|
| above | can | have | not | under | with |
| an | do | how | or | what | you |
| and | find | I | over | when | your |

# Index

## E



## F



## G



## H



## I



## J

## K



## L



## M



## N



## O



## P

## S

## T



## U



## V



## W

## X