Oracle® Endeca Server

Developer's Guide

Version 7.4.0 • September 2012

Table of Contents

Preface

Oracle® Endeca Server is the core search-analytical database. It organizes complex and varied data from disparate source systems into a faceted data model that is extremely flexible and reduces the need for up-front data modeling. This highly-scalable server enables users to explore data in an unconstrained and impromptu manner and to rapidly address new questions that inevitably follow every new insight.

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 Oracle Endeca Server.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Customer Support

Oracle 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 the backbone of all applications running on top of it. The set of server Web services provide the interfaces to the Oracle Endeca Server. You use the Web Services through Integrator to load the data, and through Studio components to query the Oracle Endeca Server and manipulate the query results. You can also use these web services directly with the Oracle Endeca Server.

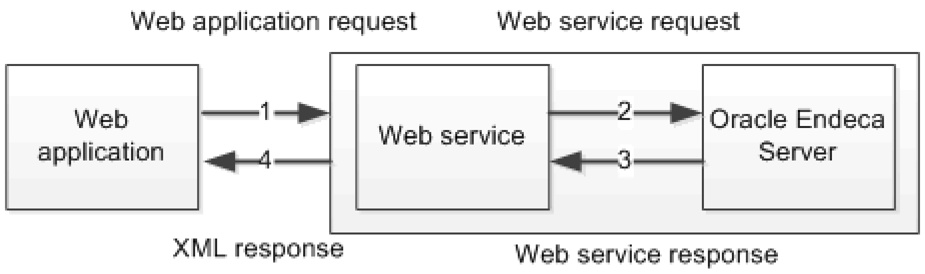
The data flow

The Oracle Endeca Server communicates with the 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 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 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 performs formatting of the query results and returns them to the client browser, via the Web application server.

Note: For security reasons, you should never allow Web browsers to connect directly to the Oracle Endeca Server. Browsers should always connect to your application through an application server.

Oracle Endeca Server overview

The Oracle Endeca Server is the query engine that provides the foundation for all solutions that utilize it.

The Oracle Endeca Server uses proprietary data structures and algorithms that allow it to provide real-time responses to client requests. The Oracle Endeca Server stores the data files that were created after your source data is ingested. After the data files are stored, the Oracle Endeca Server receives client requests via the application tier, queries the data files, and returns the results.

The Dgraph is the name of the process for the data store created in the Oracle Endeca Server. Because the Dgraph is key to every implementation, its performance is critical. A typical implementation includes the Oracle Endeca Server that is running one or more Dgraphs.

The Oracle Endeca Server is designed to be stateless. This design requires that a complete query be sent to it for each request. The stateless design facilitates the addition of Oracle Endeca Servers for load balancing and redundancy — any replica of an Oracle Endeca Server can reply to queries independently of other replicas.

Consequently, configuring additional Dgraph processes as nodes in a cluster increases availability of request processing. If a node in the cluster goes down, at least one of the Dgraphs running in the cluster continues to reply to queries. In addition, total response time is improved by using load balancers to distribute queries among the nodes in the cluster.

Oracle Endeca Server Web services

The Oracle Endeca Server is installed with a set of versioned Web services for loading, configuring, and querying the data. These Web services provide the interface to the Oracle Endeca Server.

List of Web services available for the data store on the Oracle Endeca Server

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

- Data Ingest Web Service, `ingest` (Documented in the *Oracle Endeca Server Data Loading Guide*)
- Configuration Web Service, `config` (Documented in this guide)
- Conversation Web Service, `conversation` (Documented in this guide)
- Transaction Web Service, `transaction` (Documented in this guide)
- Administration Web Service, `admin` (Documented in the *Oracle Endeca Server Administrator's Guide*)

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

- The LQL Parser Web Service, `lql_parser` — a web service for parsing Endeca Query Language queries and filters. For more information on this web service, see the *Oracle Endeca Server API Reference* and the *Oracle Endeca Server Query Language Reference*.
- The Entity Configuration Web Service, `sconfig`, used by Integrator and Studio to create and manage views. For more information on this web service, in the *Oracle Endeca Information Discovery Studio User's Guide*, see the section on creating views.
- The Control Web Service, `control`, used internally by the Oracle Endeca Server commands.

Note: Each web service is assigned its own version, consisting of major and minor version. 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 the section in this guide.

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 API Reference* (if you are planning to use this interface directly), or the *Oracle Endeca Information Discovery Integrator Components Guide* (if you are planning to use components in Integrator 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 Data Ingest Web Service and the Configuration Web Service to load data and configuration into the data store created on the Oracle Endeca Server.
2. Use the Configuration Web Service to configure the record schema and Oracle Endeca Server features.
3. Use the Conversation Web Service and the Configuration Web Service to send requests and obtain results from the Oracle Endeca Server for your data store. These results can subsequently be rendered in the front-end application used by the end users.
4. Use the Administration Web Service to set up monitoring and backups.

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 Service	The Data Ingest Web Service is used to load data into the Oracle Endeca Server. It serves as the basis for various batch processes, and is designed for easy integration with ETL tools.
Configuration Web Service	The Configuration Web Service supports the process of refining the records schema and adjusting your configuration in the development environment.

Web service	Function
Conversation Web Service	The Conversation Web Service is used to query the Oracle Endeca Server and to provide summarizations. Unlike the Data Ingest and Configuration Web Services, it is not used to update the Oracle Endeca Server data files.
Administration Web Service	The Administration Web Service is used by IT engineers and administrators to integrate the Oracle Endeca Server and its reporting with third-party IT tools.
Transaction Web Service	The Transaction Web Service controls outer transactions in the Oracle Endeca Server.

About the Oracle Endeca Server API Reference

The *Oracle Endeca Server API Reference* is a collection of automatically generated reference documentation for each of the Web services and the Bulk Load Interface that are packaged with the Oracle Endeca Server.

The *Oracle Endeca Server API Reference* is the documentation 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 Reference* also includes the documentation for the Bulk Load Interface.

The *Oracle Endeca Server API Reference* is located in the `doc` directory of the Oracle Endeca Server installation. It is 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 Administrator's Guide* discusses how to use the Administrator Web Service. The *Oracle Endeca Server Data Loading Guide* discusses how to use the Data Ingest Web Service.

About the Java client examples

The Oracle Endeca Server includes a collection of Java client examples for sending queries to the Oracle Endeca Server. Treat these examples as one of the possible ways to build your own front-end client code.

The examples are based on Java stubs generated with CXF version 2.4.2, however, you can use any other utility for stub generation.

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. The Java client examples are not intended to be extensible to real tasks. Use the Java client examples 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 the Bulk Load Interface. The automatically-generated documentation for these interfaces, collectively known as the *Oracle Endeca Server API Reference*, 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 the Web service interfaces.

The Java client examples are installed in the `/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 a version of CXF `wsdl2java`.
`src`	Contains sample code for accomplishing basic tasks using these stubs, such as configuring a data store, adding individual records, and making basic queries. 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, supporting JAR files, and a Perl script for starting the Oracle Endeca Server and running the tests. Before running the Perl script, ensure that you have Perl and the Java JDK packages installed on your machine and that your PATH environment variable includes the path to the `bin` directories of both of these packages. 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 test to run. By default, the script assumes that the script's directory is part of the installation, and it will start the Oracle Endeca Server from its sibling path. Alternatively, the script can check the environment variable `ENDECA_SERVER` for the location of your Oracle Endeca Server installation.

Handling secure Web Service requests

Depending on the protocol in the web service request, the Oracle Endeca Server issues an `http` or `https` response.

The front-end clients of the Oracle Endeca Server that do not wish this behavior may override it by adding `?https=true` or `?https=false` to their WSDL query strings, as in this example:

```
http://localhost:<port>/ws/config/DataStore?wsdl?https=true
```

The 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 *Using the Configuration Web Service on page 27*.

Chapter 2

Oracle Endeca Server Concepts

This chapter 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.

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 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. Source records in a variety of formats are supported.
Data records	In most applications, you are primarily concerned with ***data records***. Data records are the business records you want to explore using your application.
System records	***System records*** represent the records schema in the Oracle Endeca Server data files. They are created in the Oracle 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.
Primordial records	***Primordial records*** are created automatically and used internally by the Oracle Endeca Server. They represent the most basic infrastructure of the data model.

Attributes

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

For a data record, an assignment on an attribute provides information about that record. For example, for a list of book records, an assignment on 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 collection of these records that describe attributes forms a schema for your records. This collection 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 standard attribute values. An ***assignment*** indicates that a record has a value for a standard attribute.

A record typically has assignments for 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 only have 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 multi-assign. To make a standard attribute single-assign, you must update the attribute configuration.

Primary keys

In the Oracle Endeca Server data model, primary keys (also known as record specs) are used to uniquely identify records.

In order for an attribute to be used as a ***primary key***:

- The attribute must be unique

This means that no two records in a single Endeca data store may have the same value for a primary key attribute. (Note that by default, a standard attribute is not unique. To make a standard attribute unique, you must update the standard attribute configuration.)

In addition, each record must have an assignment from exactly one primary key, so that the Oracle Endeca Server can uniquely identify it (in order to update it, for example).

Each set of records must have at least one primary key standard attribute, although this primary key attribute could be different for different sets of records. This allows the Oracle Endeca Server to handle different record types, each of which can have a meaningful identifying standard attribute. For example, a store that carries multiple types of items might identify book records by a BookID primary key attribute, and apparel records by an ApparelID 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.
`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 timezone 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.

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>
  <ProductID type="mdex:int">12345</ProductID>
  <BikeType type="mdex:string">Road Bikes</BikeType>
  <Color type="mdex:string">Red</Color>
</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.
- The Name attribute is also unique and single-assign, to avoid duplicated product names across the data set.
- No records have multiple assignments.
- 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

Name	Bike Type	ProductID	Size Range	Color	Number Sold	Price
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>
  <Name type="mdex:string">Road-450</Name>
  <ProductID type="mdex:int">4038</ProductID>
  <BikeType type="mdex:string">Road Bikes</BikeType>
  <SizeRange type="mdex:string">42-46 CM</SizeRange>
  <Color type="mdex:string">Red</Color>
  <NumSold type="mdex:int">171</NumSold>
  <Price type="mdex:double">1457.99</Price>
</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>
  <Name type="mdex:string">Road-450</Name>
  <ProductID type="mdex:int">4038</ProductID>
  <BikeType type="mdex:string">Road Bikes</BikeType>
  <SizeRange type="mdex:string">42-46 CM</SizeRange>
  <Color type="mdex:string">Red</Color>
  <ReviewScore type="mdex:int">35</ReviewScore>
  <ReviewScore type="mdex:int">45</ReviewScore>
  <ReviewScore type="mdex:int">60</ReviewScore>
  <Price type="mdex:double">1457.99</Price>
</Record>
<Record>
  <Name type="mdex:string">Touring-3000</Name>
  <ProductID type="mdex:int">4035</ProductID>
  <BikeType type="mdex:string">Mountain Bikes</BikeType>
  <SizeRange type="mdex:string">48-52 CM</SizeRange>
  <Color type="mdex:string">Yellow</Color>
  <Price type="mdex:double">742.35</Price>
</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 on 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 the Oracle Endeca Server data store, these characteristics become part of the Oracle Endeca Server data files.

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 which 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 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.

System records

The Oracle Endeca Server data store uses ***system records*** to store configuration information.

You can configure the following system records:

- **Property Description Records (PDRs)**, used to define the format and behavior of standard attributes and managed attributes.
- **Dimension Description Records (DDRs)**, used to define managed attributes and thus, among other characteristics, enable the creation of hierarchical standard attribute values.
- The **Global Configuration Record (GCR)**, used to control various aspects of the global configuration.

To avoid naming collisions with customer-created records and attributes, the keys for system records are prefixed with specific namespaces, such as `mdex-property`.

Property Description Record (PDR)

A ***Property Description Record (PDR)*** is a system record that defines a record for a standard and managed attribute in the Oracle Endeca Server data store.

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
- 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 the Oracle Endeca Server data store 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.

Required schema attributes of a PDR

PDRs have the following required attributes:

Schema attribute	Type	Description
`mdex-property_Key`	`string`	The name of the standard attribute. 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*
`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.
`mdex-property_Type`	`string`	The data type of the standard attribute. The possible values are `mdex:string`, `mdex:int`, `mdex:int64`, `mdex:double`, `mdex:boolean`, `mdex:dateTime`, `mdex:time`, `mdex:duration`, and `mdex:geocode`. The default is `mdex:string`.
`mdex-property_IsSingleAssign`	`boolean`	If set to `true`, each record can have at most one value for the standard attribute. If set to `false`, each record may have more than one value for the standard attribute. The default is `false`.
`mdex-property_IsUnique`	`boolean`	If set to `true`, then each record must have a unique value for the standard attribute. If set to `false`, then multiple records can have the same value. The default is `false`.

Schema attribute	Type	Description
`mdex-property_IsTextSearchable`	`boolean`	If set to `true`, then the standard attribute is enabled for text search. If set to `false`, the standard attribute does not support text search. The default is `false`.
`mdex-property_TextSearchAllowsWildcards`	`boolean`	If set to `true`, then wildcard search is enabled for this standard attribute. If set to `false`, then wildcard search is not enabled. If this is set to `true`, then `mdex-property_IsTextSearchable` must be set to `true`. The default is `false`.
`mdex-property_IsPropertyValueSearchable`	`boolean`	If set to `true`, the standard attribute is enabled for value search. If set to `false`, the attribute is not value-searchable. The default is `true`. This schema attribute can be changed only for the standard attributes of type string. 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`	Used to configure the multi-select feature for a standard attribute. The allowed values are: • `single`. Users can select only one refinement from this attribute. • `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). • `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). The default is `single`.
`system-navigation_Sorting`	`string`	The order in which to display refinements in the navigation menu. The allowed values are: • `lexical` sorts refinements alphabetically or by number. • `record-count` sorts refinements in descending order, by the number of records available for each refinement. The default is `record-count`.
`system-navigation_ShowRecordCounts`	`boolean`	Whether to show record counts for a refinement. If set to `true`, the record counts are shown. If set to `false`, the record counts are not shown. The default is `true`.
`system-property_GroupMembership`	`string`	The groups to which the attribute belongs.

User-defined schema attributes of a PDR

You can use the arbitrary, user-defined schema attributes in a Property Description Record to display various aspects of how your data records are organized.

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, 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:

Schema attributes	Type	Description
`mdex-dimension_Key`	`string`	The name of the managed attribute.
`mdex-dimension_EnableRefinements`	`boolean`	If set to `true`, then refinements are displayed. If set to `false`, refinements are not displayed. In other words, the managed attribute is hidden. The default is `true`.
`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. If set to `false`, then the search matches only the assigned values. The default is `false`.
`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. If set to `false`, then the search only matches records with the assigned values. The default is `false`.

Global Configuration Record (GCR)

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

Definition

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

The Global Configuration Record controls the following areas of the Oracle Endeca Server data store 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 Oracle Endeca Server. By default, it is disabled.
Search characters	List which characters you want to identify as search characters to the Oracle Endeca Server.
Merge policy	Optionally, change the policy that the Oracle Endeca Server 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. To do this, for each attribute assignment on a record and for each managed attribute value, you specify the following parameters: • Minimum word occurrence • Minimum word length • Maximum word length **Note:** If you change the spelling settings in the Global Configuration Record, you must run the `admin?op=updateaspell` command in order for them to take effect.

Modifying the settings in the Global Configuration Record

To change the Global Configuration Record settings, use the Data Ingest Web Service, or Integrator.

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`. This attribute is unique and single-assign.
`mdex-config_EnableValueSearchWildcard`	`Boolean`	If set to `true`, then wildcard search is enabled for value search. If set to `false`, then wildcard search is disabled. The default value is `false.`
`mdex-config_MergePolicy`	`String`	The allowed values are `balanced` or `aggressive`. The default is `balanced`.
`mdex-config_SearchChars`	`String`	The characters to use as search characters in the Oracle Endeca Server. The allowed values are strings that are listed sequentially and are not separated by commas or spaces. Each string is a search character.
`mdex-config_SystemRecordVersion`	`String`	The version of the system records in the Oracle Endeca Server. This attribute is used by the Oracle Endeca Server and should not be modified.
`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. The default value is 4.

Attribute	Type	Description
`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. The default value is 3.
`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. The default value is 16.
`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. The default value is 1.
`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. The default value is 3.
`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. The default value is 16.

Validating the GCR settings

During record updates, the Oracle Endeca Server 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.

Part II

Web Services for the Oracle Endeca Server

Chapter 3

About Web Service Versions

This chapter discusses versions used for the web service interfaces in the Oracle 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 1.0, its major version is 1, 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 Information Discovery Installation 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 store in the Oracle Endeca Server. The WSDL document indicates the version of a web service.

You can obtain a WSDL document for a specific web service once the Oracle Endeca Server is installed and your data store is running on it.

Each web service has its own version. If a web service has more than one minor version, for example, 1.0 and 1.1, 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 the Oracle Endeca Server for the WSDL document of the web service, `http://localhost:<port>/ws/<WebServiceName>/<DataStore>?wsdl`, as in the following example for the Configuration Web Service:

   ```
   http://localhost:<port>/ws/config/DataStore?wsdl
   ```

   where the host and port represent the Oracle Endeca Server, `config` is the name of the Configuration Web Service, and the `DataStore` is the name of the data store created on the server.

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

   ```
    xmlns:config-service-v1_0="http://www.endeca.com/MDEX/config/services/config/1/0"
   ```

   In this example, 1 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: `config`, `admin`, `transaction`, `conversation`, `diws` (Data Ingest available at `/ws/ingest`), `control`, `lql-parser`, `sconfig`.

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), an error is issued listing both versions, similar to the following example: `Invalid version for <`*`operation_name`*`>. Request version is 2010.8; supported version is 1.0`.

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 (1.0):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
config:configTransaction xmlns:config="http://www.endeca.com/MDEX/config/services/types/1/0"
xmlns:mdex="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 Administration Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:admin="http://www.endeca.com/MDEX/admin/1/0">
<soapenv:Header/>
<soapenv:Body>
<admin:Request>
...
</admin: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 the Oracle 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 the Oracle 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 Oracle Endeca Server package.
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._1._0.startOuterTransactionDocument;
   ```

   The namespaces for each operation indicate which versions are supported for this operation. In this case, the `startOuterTransaction` operation is supported as of version 1.0.
4. Start using the new stubs in your front-end application to send requests to the data store created in the Oracle Endeca Server.

Chapter 4

Using the Configuration Web Service

This chapter describes the Configuration Web Service.

About the Configuration Web Service

The Configuration Web Service provides an interface that allows ergonomic interaction with both the Oracle Endeca Server configuration and record schema.

Overview

The Configuration Web Service allows you to manipulate schema and configuration. The service is declared in its WSDL document, which you can access at this URL: `http://localhost:<port>/ws/config/DataStore?wsdl`, similar to other packaged Web services. The host and port represent the Oracle Endeca Server, and the `DataStore` is the name of the data store 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 Oracle Endeca Server data store:

- If a record with the specified key already exists in the data store, 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.

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 PDRs are the following:

Operation	Description
`exportProperties`	Return all Property Description Records (PDRs) for the data store.
`listProperties`	Return the key of the PDRs for the standard attributes present in the schema of the Oracle Endeca Server data store.
`getProperties`	Return PDRs for the specified attribute keys. Attribute keys are obtained from `listProperties`.

Operation	Description
`putProperties`	Add the PDRs (specified as an argument) to the schema of the Oracle Endeca Server data store. If an attribute with the same key exists, it is replaced.
`updateProperties`	Lets you add or modify specified assignments on the PDR. As an argument, specify an attribute key associated with an existing PDR and zero or more assignments. The operation replaces the assignment on the PDR with a new assignment if it is provided as an argument. There is no requirement to specify the entire PDR to this operation; there is a requirement to specify the standard attribute key.

Operations for DDRs

The operations on DDRs are the following:

Operation	Description
`exportDimensions`	Return all Dimension Description Records.
`listDimensions`	Return the key of each managed attribute present in the Oracle Endeca Server data store.
`getDimensions`	Return DDRs for specified managed attribute keys. Managed attribute keys are obtained from `listDimensions`.
`putDimensions`	Add the DDRs (specified as arguments) to the Oracle Endeca Server data store. If a managed attribute with the same key exists, it is replaced.
`updateDimensions`	Lets you add or modify specified assignments on the DDR. As an argument, specify a managed attribute key associated with an existing DDR and zero or more assignments. The operation replaces the assignment on the DDR with a new assignment if it is provided as an argument. There is no requirement to specify the entire DDR to this operation; there is a requirement to specify the managed attribute key.

Operations for Attribute Groups

The operations on attribute groups are the following:

Operation	**Description**
`importGroups`	Remove any existing groups and add the specified ones.
`exportGroups`	Return the full representation of each group.
`listGroups`	Return a summary of each group.
`getGroups`	Return the specified groups. 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. 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.
	
	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 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 store:
	
	`<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>`

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 store (created with the Oracle Endeca Server).
`putGlobalConfigRecord`	Add the Global Configuration Record to the data store. If the GCR already exists, it is replaced.

Operations for XML configuration documents

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.

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 store.
`putPrecedenceRules`	Add or replace each of the specified precedence rules.
`deletePrecedenceRules`	Take a list of precedence rule keys and completely delete each rule.

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.

Operation	Description
`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 for your standard and managed attributes.

If you load your attribute schema before loading your source records, you can modify the resulting PDRs and DDRs as needed. After they have been configured as desired, you can then use the Data Ingest Web Service to load your source records.

- The `putProperties` element loads the standard attributes schema.
- The `putDimensions` element loads the managed attributes schema.

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

```
<config-service:configTransaction
   xmlns:config="http://www.endeca.com/MDEX/config/services/types/1/0"
   xmlns:mdex="http://www.endeca.com/MDEX/config/XQuery/2009/09">
<config-service:putDimensions>
    <mdex:record>
      <mdex-dimension_EnableRefinements>true</mdex-dimension_EnableRefinements>
      <mdex-dimension_IsDimensionSearchHierarchical>true<
/mdex-dimension_IsDimensionSearchHierarchical>
      <mdex-dimension_IsRecordSearchHierarchical>true</mdex-dimension_IsRecordSearchHierarchical>
      <mdex-dimension_Key>BikeType</mdex-dimension_Key>
    </mdex:record>
  </config-service:putDimensions>
  <config-service:putProperties>
    <mdex:record>
      <mdex-property_IsSingleAssign>true</mdex-property_IsSingleAssign>
      <mdex-property_IsTextSearchable>false</mdex-property_IsTextSearchable>
      <mdex-property_IsUnique>true</mdex-property_IsUnique>
      <mdex-property_Key>ProductID</mdex-property_Key>
      <mdex-property_TextSearchAllowsWildcards>false</mdex-property_TextSearchAllowsWildcards>
      <mdex-property_IsPropertyValueSearchable>false</mdex-property_IsPropertyValueSearchable>
      <mdex-property_Type>mdex:int</mdex-property_Type>
      <mdex-property_DisplayName>Product ID</mdex-property_DisplayName>
    </mdex:record>
    <mdex:record>
      <mdex-property_IsSingleAssign>true</mdex-property_IsSingleAssign>
      <mdex-property_IsTextSearchable>true</mdex-property_IsTextSearchable>
      <mdex-property_IsUnique>false</mdex-property_IsUnique>
      <mdex-property_Key>BikeType</mdex-property_Key>
      <mdex-property_TextSearchAllowsWildcards>true</mdex-property_TextSearchAllowsWildcards>
      <mdex-property_IsPropertyValueSearchable>true</mdex-property_IsPropertyValueSearchable>
      <mdex-property_Type>mdex:string</mdex-property_Type>
      <mdex-property_DisplayName>Bike Type</mdex-property_DisplayName>
    </mdex:record>
  </config-service:putProperties>
</config-service:configTransaction>
```

The example creates two standard attributes (ProductID and BikeType) and one managed attribute (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.

To load an attribute schema into the data store of the Oracle Endeca Server:

1. Make sure that the Oracle Endeca Server and the data store are running. Access the Configuration Web Service for the data store: `http://localhost:<port>/ws/config/DataStore?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/1/0">
         <config-service:successPutDimensions/>
         <config-service:successPutProperties/>
      </config-types:results>
   </soapenv:Body>
</soapenv:Envelope>
```

Loading configuration documents

You can use the Configuration Web Service to load the XML configuration documents to the data store'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 223*.

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. This request creates three search interfaces. The `RECSEARCH_CONFIG` document specifies the following search interfaces — `Spanish`, `English` and `Essay`:

```
<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="10">Spanish</MEMBER_NAME>
          <MEMBER_NAME RELEVANCE_RANK="9">English</MEMBER_NAME>
          <MEMBER_NAME RELEVANCE_RANK="1">Essay</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 store of the Oracle Endeca Server:

1. Make sure that the Oracle Endeca Server and the data store are running. Access the Configuration Web Service for the data store: `http://localhost:<port>/ws/config/DataStore?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/1
/0">
         <config-service:successPutConfigDocuments/>
     </config-types:results>
   </soapenv:Body>
</soapenv:Envelope>
```

Using the Configuration Web Service in Integrator

The Configuration Web Service lets you perform operations with the schema and configuration documents. All of these operations are supported in Integrator, 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 to load schema records, or records representing precedence rules. You can also use the **WebServiceClient** component of Integrator to load the Global Configuration Record and configuration documents.

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

Chapter 5

Using the Conversation Web Service

This chapter describes the role and operations of the Conversation Web Service in the Oracle Endeca Server and shows how to build basic queries with it.

About the Conversation Web Service

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

Overview

The service is a WS-I compliant SOAP/HTTP Web service that also supports the wrapped-document/literal pattern of binding.

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

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

```
http://localhost:<port>/ws/conversation/DataStore?wsdl
```

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

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_v1_0="http://www.endeca.com/MDEX/conversation/1/0"
```

In this example, 1 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.

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

- Guided navigation
- Record and value searches
- Communication between the front-end application client and the Oracle Endeca Server

- A range of summarizations

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

Conversation Web Service operations

The Conversation Web Service interface provides an operation that queries the Oracle Endeca Server.

This topic provides an overview of the operations in the Conversation Web Service.

Operation description

At a high level, the Conversation Web Service facilitates a dialog with users about data. It manages the filter state, content element configurations, and operators:

- The filter state reflects the currently selected records and the selections that were used to reach them.
- Content element configurations provide information about the currently selected records.
- The operators represent requests to change the filter state or reconfigure content elements, typically as a result of user actions. Operators can be specified for refinements, record and range filters, breadcrumbs, and other aspects of the front-end application available for navigation.

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 (typically, empty, or containing a simple record filter), and a number of content element configurations. These filter state and content element configurations are sent in a Conversation Web Service request.
2. The response to this initial request returns the filter state in a `State` element, describing the records selected, and the resulting content elements.

   The content elements contain a number of operators, represented by the various types of operators available through the `Operator` complex type, each describing a user action that might occur from this state. For instance, a record list content element contains operators to re-sort the records, or to view different pages; a navigation menu content element contains an operator for each refinement.

   Operators describe how a request to the Oracle Endeca Server differs from the previous request. Typically, a `Request` is constructed from the `Results` returned by the previous invocation of the Conversation Web Service request.
3. When the user chooses a particular action, the front-end application submits a new request through the Conversation Web Service, passing in the filter `State` returned by the previous response and passing in the operators corresponding to the user actions.

   The requester may also attach arbitrary XML to the request via the `PassThrough` element, to be returned unchanged in the response.
4. The response returns a transformed query along with new filter `State` and new content element contents (response data).

To summarize, the "conversation" underlying this web service consists of the following stages: The Conversation Web Service offers a list of content elements and a number of operators, the front-end application selects some operators, and the Conversation Web Service offers new content elements and new operators.

Request

The `Request` operation looks like the following:

```
<operation name="Request">
  <input name="request" message="cs:Request"/>
        <output name="response" message="cs:Results"/>
        <fault name="fault" message="cs:Fault"/>
</operation>
```

The `Request` operation takes a `Request` complex type as its input. The schema for the `Request` complex type is:

```
<complexType name="Request">
    <sequence>
    <element name="OuterTransactionId" minOccurs="0" type="cs_v1_0:NonEmptyString"/>
    <element name="State" type="cs_v1_0:State"/>
    <element name="Operator" type="cs_v1_0:Operator" minOccurs="0" maxOccurs="unbounded"/>
    <element name="ContentElementConfig" type="cs_v1_0:ContentElementConfig"
   minOccurs="0" maxOccurs="unbounded"/>
    <element name="PassThrough" type="cs_v1_0:CatchAll" minOccurs="0"/>
    </sequence>
</complexType>
```

A request consists of a filter state, and a list of content element configurations and operators to compute. Each request specifies:

Element	Description
`OuterTransactionId`	Has to be the first element in the request, and is optional. It must be specified only if the request runs within an outer transaction.
`State`	Contains inputs that affect the set of records to operate on. A filter state may contain, for example, selected refinements, search terms, and record filters.
`Operator`	Transform the filter state and configuration. Each request may contain a sequence of operators.
`ContentElementConfig`	Represents a message to the Oracle Endeca Server, asking it to provide certain information relative to a certain filter state. For example, these elements can describe 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.
`PassThrough`	A placeholder element for adding arbitrary XML to the request; it is returned unchanged in the response.

Response

The `Request` operation outputs a `Results` response. The response contains the `Request` element that generated it, as well as any components that were requested. Each component is returned only if its corresponding configuration was supplied in the request.

In other words, a response from the Conversation Web Service contains operators for refinements, breadcrumbs, and other aspects of the front-end application available for navigation.

The schema for the `Results` complex type response is:

```
<complexType name="Results">
    <sequence>
    <element name="Request" type="cs_v1_0:Request"/>
    <element name="ContentElement" type="cs_v1_0:ContentElement"
   minOccurs="0" maxOccurs="unbounded"/>
    <element name="PassThrough" type="cs_v1_0:CatchAll" minOccurs="0"/>
 </sequence>
</complexType>
```

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.

Examples of building query requests

Each request to the Conversation Web Service consists of a filter state and a list of content element configurations and operators to compute. This topic provides examples showing the contents of a typical request.

Example with operator

The following request is used for a record search query. It specifies the search interface that must be used, and also the search terms entered by the user in the front-end application. It utilizes the `Operator` element. This element is the base type for various types of operators, such as `AppplySpellingSuggestionOperator`, `RecordFilterOperator`, `RangeFilterOperator`, and `RefinementOperator`.

In this specific example, state and content element configuration are not shown and only the operator is shown. The `Operator` is used with `SearchOperator` type, which adds a text search component to the filter state.

The `SearchOperator` specifies the options for the `SearchFilter` element, which are `Key` (representing the name of the search interface to use), and the text value of this element (representing the actual user-entered search terms):

```
<Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="SearchOperator" Within="false">
   <SearchFilter Mode="AllPartial" RelevanceRankingStrategy="numfields"
      Key="Description">spice flavors</SearchFilter>
</Operator>
```

In general, for the `SearchFilter` type of `Operator`, you can specify various aspects of your request configuration, such as the name of the configured search interface, the search mode, the relevance ranking strategy, whether to enable snippeting, and the snippet length.

Similarly, for other types of `Operator`, you can specify options of their own. For instance, for the `RangeFilterOperator`, you can specify the range filter type and the lower and upper bound ranges.

To remove an operator from the filter state, you can use `Pop`*`Name_of_operator`*`Operator`, where *Name_of_operator* is the name of the operator that you want to remove, such as `RangeFilter`, `RecordFilter`, or `Search`.

For more information on value search, search modes, and relevance ranking, see the dedicated sections in this guide. For more information on the detailed syntax of various types of operators, see the *Oracle Endeca Server API Reference* for the Conversation Web Service.

Example with state, operator and content element

The following example is used to request breadcrumbs in a search query that also returns spelling correction information. Such a request needs to include all three parts — state, operator, and content element configuration:

1. The initial state that must be passed to the Oracle Endeca Server (it is empty in this example).
2. The `Operator` of type `SearchOperator`, which uses `SearchFilter` to specify the actual user-entered search term that requires spelling correction and the search mode.
3. The content element configuration, represented by `ContentElementConfig`. In this example, it contains a configuration requesting breadcrumbs (via `BreadCrumbConfig`) and a configuration requesting spelling correction (via `SearchAdjustmentConfig`).

   In general, the `ContentElementConfig` complex type can contain many subtypes, such as `AttributeGroupListConfig`, `BreadCrumbConfig` (as in the example below), `LQLConfig`, `RecordListConfig`, `PropertyListConfig`, or `ValueSearchConfig`. Each of these subtypes specifies a particular configuration, such as whether to return breadcrumbs, how to return lists of records or attributes, or which options to use when searching for attribute values.

To return to the example, it contains state, operator, and content element configuration, as follows:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
 <State/>
   <Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchOperator" Within="false">
    <SearchFilter Mode="All" Key="English">
     fife
    </SearchFilter>
   </Operator>
 <ContentElementConfig
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="BreadcrumbConfig" ReturnFullPath="true"
 HandlerFunction="BreadcrumbHandler"
 HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
 Id="Breadcrumbs"/>
 <ContentElementConfig
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="SearchAdjustmentConfig"
 HandlerFunction="SearchAdjustmentHandler"
 HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
 Id="SearchAdjustments"/>
 <PassThrough>...</PassThrough>
</Request>
```

For more information on requesting breadcrumbs, see the section in this guide.

Example of an EQL request

A request made with the Conversation Web Service can include statements in the Endeca Query Language (EQL). To make EQL requests, add EQL statements with the `LQLConfig` type.

Consider the following EQL statement:

```
RETURN statement 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 to the Oracle Endeca Server, use the `LQLConfig` type of `ContentElementConfig`, including the statement inside the `LQLQueryString` element, as in this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
<soapenv:Header/>
<soapenv:Body>
 <ns:Request>
  <ns:State/>
   <ns:ContentElementConfig Id="LQLConfig" xsi:type="ns:LQLConfig"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    HandlerFunction="LQLHandler"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <ns:LQLQueryString>
      RETURN statement 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:LQLQueryString>
    </ns:ContentElementConfig>
   </ns:Request>
  </soapenv:Body>
</soapenv:Envelope>
```

The contents of the `LQLQueryString` in `LQLConfig` must be a valid EQL statement. For detailed information on EQL syntax, see the *Oracle Endeca Server Query Language Reference*.

The `HandlerFunction` that supports processing of the `LQLConfig` is `LQLHandler`.

Note: This example shows only one of the ways for using EQL statements in Conversation Web Service requests. Typically, requests also include State and Operators that define the navigation state. In your EQL statement, you can select from this navigation state using the From clause.

The following abbreviated response returned from the Conversation Web Service contains the calculated results of the EQL statements:

```
<cs:ContentElement xsi:type="cs:LQL" Id="LQLConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <cs:ResultRecords NumRecords="1" Name="statement">
  <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:ContentElement>
```

Chapter 6

Using the Entity Configuration Web Service

This chapter describes the role and operations of the Entity Configuration Web Service in the Oracle Endeca Server. It also contains examples of web service requests for managing entities.

About entities

Entities provide you with intuitive, conceptual views of various categories in your data. They reflect the relational complexity between categories of data that is present in the Oracle 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 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 re-establish the relationship between categories of data once all data is loaded into the Oracle 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 Configuration Web Service. You can only create entities if the underlying attributes are already defined in your schema and exist in your data store.

The `semanticEntity` complex type of the Entity Configuration Web Service defines an entity and all its attributes. The `semanticEntity` type includes the following elements and attributes:

Name of element or attribute	Description
`key`	Required attribute. 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 must be based on the NCName format.
`displayName`	Optional attribute. 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.

`definition`	Required element. 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 name of any of the DEFINE statements should match the name of the entity.
	
	For example:
	`<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`	Optional attribute. Describes an entity.

`attributes`	Optional element. Represents a list of attributes in an entity. 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`.
	
	For each `semanticAttribute` element, specify the following attributes:
	• `name` specifies a unique identifier for the attribute. The identifier must follow the NCName format.
	• `datatype` specifies a valid data type, such as `mdex:string`. Valid types are listed in the `mdex.xsd`.
	• `displayName` is the name of the entity attribute in an easy-to-understand format. The display name can use a non-NCName format.
	• The `isDimension` attribute should be 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 hierarchy, and are candidates for GROUP BY statements in EQL.
	
	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"/>`
	`...</attributes>`

Name of element or attribute	Description
`metrics`	Optional element. Provides a list of one or more suggested metric elements. You create each `metric` by giving it a name and defining its EQL statement in the `definition` element. The `definition` element must contain an arithmetic formula in EQL used for aggregation when querying against the entity's attributes. 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></metric><metric name ="AvgSales" displayName="Average Sale" datatype ="mdex:double"><definition>avg(SalesAmount)</definition></metric></metrics>`

An 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">
  <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>
  <attributes>
    <semanticAttribute name="SalesAmount" displayName="Sales Amount" datatype="mdex:double"
isDimension="false"/>
<semanticAttribute name="ProductLine" displayName="Product Line" datatype="mdex:string"
isDimension="true"/>
<semanticAttribute name="SalesTerritoryCountry" displayName="Sales Territory Country"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="FiscalYear" displayName="Year" datatype="mdex:int"
isDimension="true"/>
<semanticAttribute name="SaleOrderNumber" displayName="Sale Order Number"
datatype="mdex:string" isDimension="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>
</semanticEntity>
```

Notes about the base entity

Once records are loaded into the Endeca data store, one single base entity record is created for each of the attributes in the data store. The base entity record is created based on the PDRs defining each of the attributes on your physical records. The base entity provides a convenient way for you to create your own custom entities.

Note: Because the base entity represents records derived from actual attributes that exist in the system, you cannot create, modify, or delete the base entity.

About the Entity Configuration Web Service

The Entity Configuration Web Service lets you create, replace, delete and update entities.

Entity Configuration Web Service overview

The Entity 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`. The service supports creation and management of entities, as well as validation of statements in them.

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

```
http://localhost:<port>/ws/sconfig/<DataStore>?wsdl
```

where the host and port represent the Oracle Endeca Server, and *DataStore* is the name of the data store created on the server.

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:v1_0="http://www.endeca.com/endeca-server/sconfig/1/0"
```

In this example, 1 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 Entity Configuration Web Service operations and for schema elements, see the *Oracle Endeca Server API Reference*.

Operation description

A request to the Entity Configuration Web Service depends on the operation. The operations perform actions on entities, listing them, adding and removing them, and validating them.

The effect of an Entity Configuration Web Service request that contains `put` operations is to add entities to the corpus of records in the Oracle Endeca Server for this data store:

- If an entity with the specified key already exists in the corpus, it is replaced by the new entity with the same key (if the EQL statements defining the entity are valid).
- If an entity does not exist, and if its EQL definition is valid, it is created.

After creation, each entity is represented as a single logical record in the Endeca data store. The on-disk storage of these records means that they persist across restarts of the Endeca data store, as they are loaded into the Dgraph process at start-up time.

Request

The input to the Entity Configuration Web Service depends on the operation used. 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 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 data files inside the transaction, or against the pre-transaction version of the data 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 Configuration Web Service return data.

If the operation returns data, the response to the Entity 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 Entity Configuration Web Service.

Operations in the Entity Configuration Web Service

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

A request to the Entity Configuration Web Service may be one of the following operations:

Operation	Description
`listEntities`	List entities that already exist in the data store. Note that even if you have not created any custom entities, this operation lists the base entity created automatically on top of PDRs for your record attributes. You can use this operation to export the existing entities, for example during an upgrade procedure, in order to later import them to the Endeca Server with the `putEntities` operation.
`validateEntity`	Validate an entity with the specified key and definition before it has been added.
`validateEntities`	Validate multiple entities with specified definitions.
`putEntity`	Add an entity with the specified key and definition to the data store. 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* For an entity to be created, its building blocks — the physical records and attributes — must already exist in the data store. If an entity with the specified key already exists in the corpus, it is replaced by the new entity with the same key (if the EQL statements defining the entity are valid). If an entity does not exist, and if its EQL definition is valid, the entity is created. You cannot modify or create the base entity using this command. The base entity is created by the Oracle Endeca Server on top of PDRs for existing physical attributes on your records.
`putEntities`	Add multiple entities with the specified keys and definitions to the data store. The keys must be valid according to the NCName format. You cannot use this command to create or replace the base entity.
`deleteEntities`	Delete multiple entities for which keys are specified. You cannot delete the base entity.
`deleteAllEntities`	Delete all custom entities that exist in the corpus without specifying any of their keys. You cannot delete the base entity.

Examples of requests with the Entity Configuration Web Service

This topic includes examples of requests for listing, adding, deleting, and validating entities.

Listing entities example

The following example of the request lists all entities that are present in the corpus:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/endeca-server/sconfig/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:listEntities>
      </ns:listEntities>
   </soapenv:Body>
</soapenv:Envelope>
```

The response to this request returns a list of entities that are already added, including the base entity added by the Oracle Endeca Server for each PDR describing an existing attribute in your data store.

For example, the following abbreviated response lists the `Transactions` entity:

```
<listEntitiesResponse xmlns="http://www.endeca.com/endeca-server/sconfig/1/0">
<semanticEntity key="Transactions" displayName="Transactions">
<definition>/*Calculate Total Sales respecting Navigation*/
DEFINE GlobalSales as select sum(FactSales_SalesAmount) AS TotalSales Group;
DEFINE Transactions AS SELECT FactSales_SalesAmount AS "FactSales_SalesAmount",
DimReseller_ProductLine AS "DimReseller_ProductLine",
DimSalesTerritory_SalesTerritoryCountry AS "DimSalesTerritory_SalesTerritoryCountry",
DimDate_FiscalYear AS "DimDate_FiscalYear",
ProductSubcategoryName AS "ProductSubcategoryName",
DimReseller_ResellerName AS "DimReseller_ResellerName",
DimEmployee_FullName AS "DimEmployee_FullName",
ProductCategoryName AS "ProductCategoryName",
DimDate_FiscalSemester AS DimDate_FiscalSemester,
DimDate_FiscalQuarter AS DimDate_FiscalQuarter,
DimSalesTerritory_SalesTerritoryRegion AS DimSalesTerritory_SalesTerritoryRegion,
DimDate_FiscalYear*100+DimDate_MonthNumberOfYear as "Year-Month"
</definition><description>Sales Transaction Line Items</description>
...
</semanticEntity>
```

The `Transactions` entity contains the following attributes:

```
<attributes>
<semanticAttribute name="DimDate_FiscalQuarter" displayName="Fiscal Quarter"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="DimDate_FiscalSemester" displayName="Fiscal Semester"
datatype="mdex:int" isDimension="true"/>
<semanticAttribute name="DimDate_FiscalYear" displayName="Fiscal Year"
datatype="mdex:int" isDimension="true"/>
<semanticAttribute name="DimEmployee_FullName" displayName="Employee Name"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="DimReseller_ProductLine" displayName="Product Line"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="DimReseller_ResellerName" displayName="Reseller Name"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="DimSalesTerritory_SalesTerritoryCountry"
displayName="Sale Country" datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="DimSalesTerritory_SalesTerritoryRegion"
displayName="Sales Region" datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="FactSales_SalesAmount" displayName="Sales Amount"
datatype="mdex:double" isDimension="false"/>
```

```
<semanticAttribute name="ProductCategoryName" displayName="Product Category"
datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="ProductSubcategoryName"
displayName="Product Subcategory" datatype="mdex:string" isDimension="true"/>
<semanticAttribute name="Year-Month" displayName="Year - Month"
datatype="mdex:long" isDimension="true"/>
</attributes>
```

It also contains the following metrics:

```
<metrics>
  <metric name="SalesShare" displayName="Sales Share"
    datatype="mdex:double"
    description="Ratio of Total Sales of the Group to Total Sales of All Groups">
   <definition>sum(FactSales_SalesAmount)/GlobalSales[].TotalSales
   </definition>
  </metric>
</metrics>
```

Validating an entity example

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`, specifying the entities inside the `semanticEntity` elements. The following abbreviated example illustrates the `ValidateEntity` request:

```
<ns:validateEntity>
   <ns:semanticEntity key="Sales" displayName="Sales">
   ...
   </ns:semanticEntity>
</ns:validateEntity>
```

If the request validates successfully, `validateEntityResponse` does not contain errors.

Adding an entity example

You can add one entity using a `putEntity` operation, or add multiple entities using `putEntities`.

In this abbreviated example, a `putEntities` operation is used in a request to add one entity, `Sales`, with two metrics, `TotalSales` and `AvgSales`:

```
<ns:putEntities><ns:semanticEntity key="Sales" displayName="Sales">
<ns: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
</ns:definition>
<ns:attributes>
    <ns:semanticAttribute name="SalesAmount" displayName="Sales Amount" datatype="mdex:double"
 isDimension="false"/>
    <ns:semanticAttribute name="ProductLine" displayName="Product Line" datatype="mdex:string"
 isDimension="true"/>
    <ns:semanticAttribute name="SalesTerritoryCountry" displayName="Sales Territory Country"
 datatype="mdex:string" isDimension="true"/>
    <ns:semanticAttribute name="FiscalYear" displayName="Year" datatype="mdex:int"
 isDimension="true" />
    <ns:semanticAttribute name="SaleOrderNumber" displayName="Sale Order Number"
 datatype="mdex:string" isDimension="false"/>
</ns:attributes>
  <ns:metrics>
    <ns:metric name="TotalSales" displayName="Total Sale" datatype="mdex:double">
```

```
       <ns:definition>sum(SalesAmount)</ns:definition>
    </ns:metric>
       <ns:metric name="AvgSales" displayName="Average Sale" datatype="mdex:double">
       <ns:definition>avg(SalesAmount)</ns:definition>
    </ns:metric>
  </ns:metrics>
 </ns:semanticEntity>
</ns:putEntities>
```

The abbreviated example response from the Entity Configuration Web Service informs you how many entities were created or replaced:

```
<putEntitiesResponse xmlns="http://www.endeca.com/endeca-server/sconfig/1/0">
  <entityAdditionInformation numEntitiesAdded="1" numEntitiesReplaced="0"/>
</putEntitiesResponse>
```

You cannot add or replace the base entity.

Deleting an entity example

To delete one or more entities, issue a request with the `deleteEntities` operation, specifying keys for one ore more entities you would like to delete, as in this abbreviated example:

```
<ns:deleteEntities>
   <ns:semanticEntityKey key="Sales"/>
</ns:deleteEntities>
```

The response indicates the number of entities deleted:

```
<deleteEntitiesResponse xmlns="http://www.endeca.com/endeca-server/sconfig/1/0">
   <numEntitiesDeleted>1</numEntitiesDeleted>
</deleteEntitiesResponse>
```

You cannot delete the base entity.

Chapter 7

Using the Control Web Service

This chapter describes the role and operations of the Control Web Service in the Oracle Endeca Server.

About the Control Web Service

The Control Web Service provides an interface that lets you create and manage Endeca data store instances.

Overview

The Control Web Service is declared in `control.wsdl`. You can access the Control Web Service at the following URL:

```
http://localhost:<port>/ws/control?wsdl
```

where the `localhost` and *`port`* are the host and port of the running Oracle Endeca Server.

The namespace for the Control Web Service is similar to the following example and reflects the version of the service:

```
http://www.endeca.com/endeca-server/control/1/0
```

This namespace is included in the WSDL document for the Web service.

Operation and request overview

A request to the Control Web Service depends on the operation. The request types and their operations are:

Request Type	Operation	Response Type
`attachDataStoreRequest`	`attachDataStore`	`attachDataStoreResponse`
`createDataStoreRequest`	`createDataStore`	`createDataStoreResponse`
`dataStoreStatusRequest`	`dataStoreStatus`	`dataStoreStatusResponse`
`detachDataStoreRequest`	`detachDataStore`	`detachDataStoreResponse`
`listDataStoresRequest`	`listDataStores`	`listDataStoresResponse`

Request Type	Operation	Response Type
`startDataStoreRequest`	`startDataStore`	`startDataStoreResponse`
`stopDataStoreRequest`	`stopDataStore`	`stopDataStoreResponse`
`usageRequest`	`usage`	`usageResponse`
`versionRequest`	`version`	`versionResponse`

The operations are described in greater detail in *Operations in the Control Web Service on page 52*.

The Control Web Service does not support outer transactions, and therefore does not support the `OuterTransactionId` element that specifies the ID of an outer transaction in a request.

Response

The response types are listed in the table above. Data is returned by the `dataStoreStatusResponse`, `listDataStoresResponse`, `usageResponse`, and `versionResponse` responses.

If any operation does not succeed, the Web service transaction returns a SOAP fault with an appropriate error message and the operation is not applied. The SOAP fault will look like this example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>Cannot perform this operation while the data store is STOPPED</faultstring>
         <detail>
            <controlFault xmlns="http://www.endeca.com/endeca-server/control/1/0"/>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

In this example, the `faultstring` informs the user that the operation (which is a `stopDataStore` in this case) cannot be performed because the Endeca data store is already stopped.

Operations in the Control Web Service

This topic lists the operations of the Control Web Service.

A request to the Control Web Service will contain one of the following operations:

Operation	Description	Equivalent endeca-cmd command
`attachDataStore`	Attach an Endeca data store to existing data files on disk. The data store will be started.	`attach-ds`
`createDataStore`	Create the data files for a new Endeca data store on disk and attach it. The data store will be started.	`create-ds`

Operation	Description	Equivalent endeca-cmd command
`dataStoreStatus`	Get the status of a started or stopped Endeca data store.	`status-ds`
`detachDataStore`	Detach a stopped Endeca data store.	`detach-ds`
`listDataStores`	List the names of all attached Endeca data stores.	`list-ds`
`startDataStore`	Start a stopped Endeca data store.	`start-ds`
`stopDataStore`	Stop a started Endeca data store.	`stop-ds`
`usage`	List the available Dgraph start-up flags.	`usage`
`version`	List the version of the Oracle Endeca Server and the versions of any started Dgraph processes powering each of the data stores.	`version`

Note: The endeca-cmd `list-jobs` command is not listed in the table, as it uses the `listJobsOperation` of the Administration Web Service. For details on this operation, see the *Oracle Endeca Server Administrator's Guide*.

dataStoreConfig complex type

When you are creating or attaching a data store, you do so using the `dataStoreConfig` complex type. It represents an Endeca data store and all its configuration attributes. The `dataStoreConfig` type has the following characteristics:

Element	Data Type	Description
`name`	`string`	Required. The name of the new Endeca data store. The name must be unique among any other Endeca data stores on this node.
`dataFiles`	`string`	Required for `attachDataStore`; optional for `createDataStore`. The absolute path of the Endeca data store's data files directory to be created or attached to. For a `createDataStore`, the default is to create the data files directory in the location configured in the Endeca Server.
`wsPort`	`int`	Optional. Web service port number to be used for the Endeca data store's Dgraph process. The port number cannot be in use by any other process on the machine (including another Endeca data store). Defaults to a port number in the default range specified on the Endeca Server.

Element	Data Type	Description
`bulkLoadPort`	`int`	Optional. The port number for bulk load ingest operations for the Endeca data store's Dgraph process. Defaults to a port number in the default range specified on the Endeca Server.
`startupTimeoutSeconds`	`int`	Optional. The maximum length of time (in seconds) that is allowed for the Endeca data store's Dgraph process to start up. Default is 60 seconds.
`shutdownTimeoutSeconds`	`int`	Optional. The maximum length of time (in seconds) that is allowed for the Endeca data store's Dgraph process to shut down. Default is 60 seconds.
`arg`	`string`	Optional. Specifies one Dgraph flag that will be used for the Endeca data store's Dgraph process. For multiple flags, use an `arg` element per flag.

The `arg` cannot parse spaces. Therefore, for flags that take parameters, use one `arg` for the flag itself and a second `arg` for the parameter. For example, the "--threads 6" flag would be specified as:

```
<arg>--threads</arg>
<arg>6</arg>
```

Examples of Control Web Service requests

This topic includes examples of the various Control Web Service requests.

The examples in this topic were generated using the soapUI utility.

Creating an Endeca data store

The `createDataStore` operation lets you create a new Endeca data store, including a new set of data files, as in this example:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/control/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:createDataStore>
         <ns:dataStoreConfig>
            <ns:name>books</ns:name>
            <ns:dataFiles>c:\endeca\apps\books</ns:dataFiles>
            <ns:wsPort>8000</ns:wsPort>
            <ns:bulkLoadPort>8001</ns:bulkLoadPort>
            <ns:startupTimeoutSeconds>90</ns:startupTimeoutSeconds>
            <ns:shutdownTimeoutSeconds>120</ns:shutdownTimeoutSeconds>
            <ns:arg>--threads</ns:arg>
            <ns:arg>6</ns:arg>
         </ns:dataStoreConfig>
      </ns:createDataStore>
   </soapenv:Body>
</soapenv:Envelope>
```

The `name` attribute specifies the name of the Endeca data store to be created, while the `dataFiles` attribute specifies the location of the data store's data files if they should be created in a non-default location.

Also note that the `arg` attribute cannot parse spaces. Therefore, for flags that take parameters, use one `arg` attribute for the flag itself and a second `arg` attribute for the parameter. It is for this reason that the `--threads 6` flag in the example is specified as:

```
<ns:arg>--threads</ns:arg>
<ns:arg>6</ns:arg>
```

If the request is successful, the `createDataStoreResponse` will be:

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

Attaching an Endeca data store

An `attachDataStore` operation example is not included because its elements and attributes are identical to the `createDataStore` operation. The only difference in implementation is that an `attachDataStore` operation must use the `dataFiles` element to point to an existing data files, while a `createDataStore` operation can omit the `dataFiles` element.

Starting an Endeca data store

The `startDataStore` operation starts a stopped Endeca store, as in this example that starts the **wine** data store:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/control/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:startDataStore>
         <ns:name>wine</ns:name>
      </ns:startDataStore>
   </soapenv:Body>
</soapenv:Envelope>
```

The `name` attribute specifies the name of the Endeca data store to be started.

If the request is successful, the `startDataStoreResponse` will be:

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

Listing all Endeca data stores

The `listDataStores` operation lists the names of all attached Endeca data stores:

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

```
      <ns:listDataStores/>
   </soapenv:Body>
</soapenv:Envelope>
```

If the request is successful, the `listDataStoresResponse` will look similar to this example:

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

Getting the status of an Endeca data store

The `dataStoreStatus` operation retrieves the topped Endeca data store, as in this example:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/control/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:dataStoreStatus>
         <ns:name>wine</ns:name>
      </ns:dataStoreStatus>
   </soapenv:Body>
</soapenv:Envelope>
```

The `name` attribute specifies the name of the Endeca data store whose status is to be retrieved.

If the request is successful, the `dataStoreStatusResponse` will be similar to this example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <dataStoreStatusResponse xmlns="http://www.endeca.com/endeca-server/control/1/0">
         <config>
            <name>wine</name>
            <dataFiles>c:\endeca\apps\wine</dataFiles>
            <wsPort>7771</wsPort>
            <bulkLoadPort>7772</bulkLoadPort>
            <startupTimeoutSeconds>60</startupTimeoutSeconds>
            <shutdownTimeoutSeconds>60</shutdownTimeoutSeconds>
            <arg>--coordinator_port</arg>
            <arg>2181</arg>
            <arg>--coordinator_host</arg>
            <arg>localhost</arg>
         </config>
         <state>Started</state>
      </dataStoreStatusResponse>
   </soap:Body>
</soap:Envelope>
```

In the response, the `state` element shows whether the data store is started or stopped.

Stopping an Endeca data store

The `stopDataStore` operation stops a started Endeca data store, as in this example that stops the **wine** data store:

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

```
   <soapenv:Header/>
   <soapenv:Body>
      <ns:stopDataStore>
         <ns:name>wine</ns:name>
      </ns:stopDataStore>
   </soapenv:Body>
</soapenv:Envelope>
```

The `name` attribute specifies the name of the Endeca data store to be stopped.

If the request is successful, the `stopDataStoreResponse` will be:

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

In the response, the `terminationType` element lists how the stop operation was accomplished:

- `Normal` indicates a normal, graceful shutdown (i.e., the Dgraph was shut down within its configured shutdown timeout period).
- `Killed on Timeout` means the Dgraph could not be shut down within its configured shutdown timeout period and thus the Dgraph process was killed by the Endeca Server.

Detaching an Endeca data store

The `detachDataStore` operation detaches a stopped Endeca data store, as in this example that detaches the **books** data store:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/endeca-server/control/1/0">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:detachDataStore>
         <ns:name>books</ns:name>
      </ns:detachDataStore>
   </soapenv:Body>
</soapenv:Envelope>
```

The `name` attribute specifies the name of the Endeca data store to be detached.

If the request is successful, the `detachDataStoreResponse` will be:

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

Displaying version information

The `version` operation displays the version of the Oracle Endeca Server and the versions of the started Dgraph processes.

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

If the request is successful, the `versionResponse` will be similar to this example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <versionResponse xmlns="http://www.endeca.com/endeca-server/control/1/0">
         <serverVersion>Oracle Endeca Server 7.4.0.701
            hudson-label=endeca-server-project-windows-2960</serverVersion>
         <dataStoreVersions>
            <name>quickstart</name>
            <version>7.4.0</version>
            <build>7.4.0.665981</build>
         </dataStoreVersions>
      </versionResponse>
   </soap:Body>
</soap:Envelope>
```

Displaying Dgraph flag usage information

The `usage` operation displays usage information for the Dgraph flags.

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

If the request is successful, the `usageResponse` will display a list of Dgraph flags.

Chapter 8

Using the Transaction Web Service

This chapter describes the Transaction Web Service.

About outer transactions

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

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

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

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

The Transaction Web Service of an Endeca data store is used for controlling outer transactions.

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

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

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

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

To run an outer transaction, you can either use Integrator, or issue requests to 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 server) 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 Endeca data store.

About the Transaction Web Service

The Transaction Web Service provides a versioned interface for controlling one or more inner transactions on a particular data store.

Each web service request to the Oracle 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://localhost:<port>/ws/transaction/<DataStore>?wsdl
```

where the host and port represent the Oracle Endeca Server, and the `DataStore` is the name of the data store that is created 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 served by the same server against the pre-transaction version of the data files.

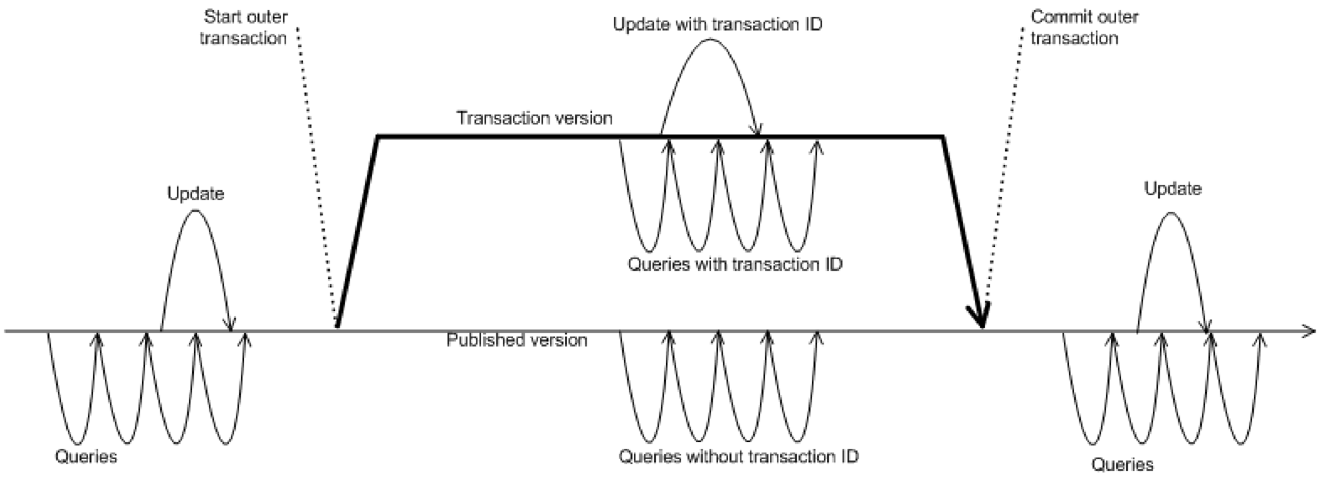
Although you can issue requests to the Transaction Web Service with any web service tool, such as soapUI, you can use the **Transaction RunGraph** in Integrator.

Outer transactions and queries

The Oracle 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 data files, 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 data files and do not attempt to change them.
- The purpose of updating queries is to change the data files or other settings in the Dgraph data store. Updating queries represent "write" requests to the data files.

The following diagram shows how both updating and non-updating queries are processed by the Oracle Endeca Server in view of outer transactions. It illustrates that, to be processed within the 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 data files in the Oracle Endeca Server:



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 data files. 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 data files 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 data files.
- Updating queries wait until the outer transaction is committed (or rolled back), and are computed based on the published version of the data files available at that time.

If the outer transaction has been started and requests that are sent to the Oracle 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 data files.

Transaction Web Service operation description

This topic describes the logic of 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 data files is available; it is known as the last published version of the data files.
- 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 data files 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 don't reference the ID wait until the outer transaction is committed (or rolled back), and are computed based on the published version of the data files. (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 data files are internally updated. Internal versions of the data files 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 server answers updating queries only if they specify the transaction ID. These updating queries are answered against a transaction version of the data files that is not published until the transaction is committed. (This transaction version reflects the most recent "writes" to the data files that occurred within the outer transaction until this point.) All non-updating queries are answered either against the last published version of the data files (if they don't 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 data files is published. Updating requests that were waiting in the queue (and that didn't 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 roll 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 data store.

  Alternatively, 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 a transaction, then updating requests that didn't specify the ID and that were waiting in the queue are processed once the outer transaction is successfully rolled back.

  For information on connectors that support transactions, see the *Oracle Endeca Information Discovery Integrator Components 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 process issues an ID automatically and returns it in the response. In addition, you can use the `listOuterTransaction` operation to obtain the ID from the running Oracle Endeca Server for your data store.

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. If this operation succeeds, it starts the outer transaction, returns a transaction ID, and the Dgraph process enters transaction mode. If this operation does not succeed, the Dgraph process does not start an outer transaction, and does not return a transaction ID. While an outer transaction is in progress, the following actions take place: • 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 data files. • Read-only queries that do not reference the transaction ID are not rejected — they are processed against the published version of the data files. Updates applied within the outer transaction do not become a published version of the data files 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. If an outer transaction with the specified ID is in progress, then if the operation succeeds, the Dgraph process commits the transaction and exits transaction mode. The Dgraph process resumes accepting unqualified queries. The version of the data files that is propagated to all nodes becomes the last published version. If the operation does not succeed, the outer transaction is not committed. The Dgraph process does not apply any updates that referenced the transaction ID. If the operation does not succeed and the transaction is not committed, all queries sent to the Dgraph process continue to use the pre-transaction version of the data files. **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 process, you can manually commit the transaction and roll back any changes using the `rollBackOuterTransaction` operation specifying the ID of the transaction.

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 data files and commit the transaction.

You can run this operation directly using a tool such as soapUI, or in Integrator, using one of the two options: either through the **Transaction RunGraph** component and its **rollback** option, or by using the **WebServiceClient** component in Integrator, by configuring it to access the Transaction Web Service on the particular data store, 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/<DataStore>?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 don't 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, the error message notifies you of the transaction ID that is in progress.
- If you are using this operation directly (such as in the soapUI) and not in the context of Integrator, you can issue it at any point during a running outer transaction on the node on which the transaction is open. Once issued, this operation ensures that inner transactions running within the outer transaction are rolled back to the state of data files prior to when the outer transaction was started.

  If you have updating requests sent to the server that didn't 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 cluster of nodes, issue this operation on the leader node only. This operation is rejected if you attempt to run it on any other node.

  Once the operation completes, it stops the outer transaction, and the leader resumes serving queries on the last version of the data files 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 data files 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.

Treatment of requests from Oracle Endeca Server interfaces

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

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

For read-only requests, if the correct ID of the outer transaction is specified as the first element in the request, they run against the most recent version inside the outer transaction. 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 data files that becomes available at that time.

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

The following statements describe each interface in detail:

- **Configuration Web Service**.Updating and non-updating requests are treated as updating requests. Changes to XML configuration documents made with requests from the Configuration Web Service are no different from updating requests and follow the same rules. See the section below in this topic for more information.
- **Entity Configuration Web Service**.This Web service allows you to run both updating and non-updating requests. These requests are treated as described at the beginning of this topic.
- **Data Ingest Web Service**. All requests from the Data Ingest Web Service are updating requests. If the correct ID is specified, the Data Ingest requests run against the most recent version available within the outer transaction; otherwise, the requests wait until the outer transaction is committed or rolled back and are computed based on the published version of the data files.
- **Conversation Web Service**. All requests from the Conversation Web Service are non-updating. These requests are treated as described at the beginning of this topic.
- **Administration Web Service**. Requests for taking snapshots are not allowed during outer transactions. Listing running jobs requests can be issued while an outer transaction is running. You can specify or omit the transaction ID. An incorrect transaction ID results in a SOAP fault from the Dgraph. Requests for canceling jobs can run regardless of outer transactions — they do not require specifying an outer transaction ID (they do require specifying the ID of the pending job).
- **Administration URL requests**. The following URL administrative request must run within an outer transaction, and thus require specifying the correct transaction ID: `/admin/datastore?op=updateaspell&outerTransactionId="myID"`

  An incorrect transaction ID results in a SOAP fault from the Dgraph. All other URL administrative requests can run alongside outer transactions regardless of whether the outer transaction ID is specified.
- **Bulk Load Interface**. All requests from the Bulk Load Interface are updating requests. If the correct ID is specified, these requests run against the most recent version available within the outer transaction; otherwise, the requests are rejected.

Note: The Control Web Service requests do not interfere with outer transactions. In other words, requests from the Control Web Service do not require specifying an outer transaction ID regardless of whether any outer transactions are running. This is because the Control Web Service runs on the host and port of the Oracle Endeca Server, whereas all other web services run on the host and port of the Dgraph process for the data store.

Treatment of changes to the configuration documents

While an outer transaction is being processed, you can update the configuration, such as the text search configuration.

For example, any changes to the configuration of search interfaces, as well as to `dimsearch_config`, `en_word_forms_collection`, `recsearch_config`, `relrank_strategies`, `stop_words`, and `thesaurus` configuration documents are possible while an outer transaction is in progress. If the correct `OuterTransactionId` is specified in these requests, the requests run within a transaction; otherwise, the requests wait until the outer transaction is committed and are computed based on the post-transaction version of the data files.

Transaction Web Service and Integrator

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

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

Performance impact of transactions

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

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

Part III

Working with Records and Record Filters

Chapter 9

Working with Records

This chapter provides information on handling records in your Web application.

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 store.

Configuring a record list

You use the `RecordListConfig` complex type to configure settings for lists of records returned by the Oracle Endeca Server. For example, 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 shown in this example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
   <soapenv:Header/>
   <soapenv:Body>
<ns:Request>
      <ns:ContentElementConfig xsi:type="ns:RecordListConfig"
          Id="RecordList"
          HandlerFunction="RecordListHandler"
          HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
          MaxPages="2">
         <ns:Column>ModelName</ns:Column>
         <ns:Column>Size</ns:Column>
        <ns:RecordsPerPage>10</ns:RecordsPerPage>
          <ns:Page>1</ns:Page>
        <ns:Sort Key="ModelName" Direction="Ascending"/>
          </ns:ContentElementConfig>
</ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

The elements and attributes that must be specified in the `RecordListConfig` complex type are the following:

Element/Attribute	Description
`HandlerFunction`	Specifies the `RecordListHandler` handler function for this `ContentElementConfig`. Required.
`HandlerNamespace`	Specifies the namespace for the handler function. Required.
`Id`	An arbitrary identifier for this `ContentElementConfig`. Required.
`MaxPages`	Optionally 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.
`Column`	Optionally defines 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`	Optionally 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`	Optionally 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 Key Direction`	Optionally specifies a sort order for the record list. `Key` specifies the standard or managed attribute used for the sort. `Direction` specifies an `Ascending` (the default) or `Descending` sort order.

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. Each record is returned in a `Record` element.

The following sample snippet shows a `RecordList` with one record, one pagination control, and one column:

```
<cs:ContentElement xsi:type="cs:RecordList" Id="RecordList" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">
<cs:NumRecords>61096</cs:NumRecords>
<cs:TotalPages>6110</cs:TotalPages>
 <cs:RecordRange First="11" Last="20"/>
 <cs:RecordListEntry>

<cs:Record>
  <ProductName type="mdex:string">Bike Wash - Dissolver</ProductName>
  <ProductSubcategoryName type="mdex:string">Cleaners</ProductSubcategoryName>
  <Description type="mdex:string">Washes off the toughest road grime; dissolves grease
   </Description>
    ...
</cs:Record>
<cs:ComputedProperties/>
 ...
<cs:PaginationControl Label="First" Active="true">
 <cs:Operator OwnerId="RecordList" Page="0" xsi:type="cs:PageOperator"/>
 </cs:PaginationControl>
  ...
<cs:Column ColumnKey="ProductName" DisplayName="Product Name" SpecColumn="false">
<cs:SortControl Key="ProductName" Direction="Ascending" Active="false" xsi:type="cs:SortControl">
<cs:Operator OwnerId="RecordList" xsi:type="cs:SortOperator" Key="ProductName" Direction="Ascending"
/>
</cs:SortControl>
<cs:SortControl Key="ProductName" Direction="Descending" Active="false" xsi:type="cs:SortControl">
<cs:Operator OwnerId="RecordList" xsi:type="cs:SortOperator" Key="ProductName" Direction="Descending"
/>
</cs:SortControl>
</cs:Column>
 ...
</cs:ContentElement>
```

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.
- `PaginationControl` is a control (a `PageOperator`) for a specific record page.
- `SortControl` identifies the sort order (Ascending or Descending) of the attributes.

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.

Paging through a record list

If many records are returned, you can specify a paging control, using the `PaginationControl` complex type.

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 pagination controls (the `PaginationControl` type) that you can use for paging.

The following is an example of a `RecordList` with a total of seven record pages and three records per page:

```
<cs:ContentElement xsi:type="cs:RecordList" Id="RecordList"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <cs:NumRecords>19</cs:NumRecords>
   <cs:TotalPages>7</cs:TotalPages>
   <cs:RecordRange First="1" Last="3"/>
   <cs:RecordListEntry>
   ...
   </cs:RecordListEntry>
   <cs:PaginationControl Label="First" Active="false">
      <cs:Operator OwnerId="RecordList" Page="0" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="Previous" Active="false">
      <cs:Operator OwnerId="RecordList" Page="-1" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="1" Active="false">
      <cs:Operator OwnerId="RecordList" Page="0" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="2" Active="true">
      <cs:Operator OwnerId="RecordList" Page="1" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="3" Active="true">
      <cs:Operator OwnerId="RecordList" Page="2" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="Next" Active="true">
      <cs:Operator OwnerId="RecordList" Page="1" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   <cs:PaginationControl Label="Last" Active="true">
      <cs:Operator OwnerId="RecordList" Page="6" xsi:type="cs:PageOperator"/>
   </cs:PaginationControl>
   ...
</cs:ContentElement>
```

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 for display:

```
<ContentElementConfig xsi:type="RecordListConfig"
     HandlerFunction="RecordListHandler"
     HandlerNamespace=""http://www.endeca.com/MDEX/conversation/1/0"
     Id="RecordList" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RecordsPerPage>5</RecordsPerPage>
</ContentElementConfig>
```

The `NumRecords` element in the `RecordList` lists the total number of records being returned by the query:

```
<cs:NumRecords>20</cs:NumRecords>
```

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 `PageOperator` type, as in this example that sets the offset to the third page of records:

```
<Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="PageOperator" OwnerId="RecordList" Page="3"/>
```

If the number of total pages is 1:

```
<cs:TotalPages>1</cs:TotalPages>
```

then no paging controls are needed.

If the number of total pages is 2 or greater, you can use the `PaginationControl` elements in the `RecordList` to go to the appropriate page, as indicated in the following table.

Page Label	Result
First	Goes to the first record page (which is page 0).
Previous	Goes to the previous record page.
Next	Goes to the next record page.
Last	Goes to the last record page.
1	Goes to the first record page (which is page 0).
2 or greater	Goes to the Nth record page.

Note that the `Active` attribute in a `PaginationControl` element indicates whether that paging control is relevant within the context of the current state. For example, if you are on the last record page, then neither the **Next** or **Last** paging controls will be active.

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` element 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 settings that limit the number of records for export are configured in Studio. For information on configuring these settings, see the *Oracle Endeca Information Discovery Studio User's Guide*.

Example request

To request a record list with a Conversation Web Service request, use `ContentElementConfig` of type `RecordListConfig`.

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`:

```
<ContentElementConfig xsi:type="RecordListConfig"
      HandlerFunction="RecordListHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      Id="RecordList" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      MaxPages="40">
   <Column>WineType</Column>
   <Column>Price</Column>
      <RecordsPerPage>20</RecordsPerPage>
   <Page>0</Page>
   <Sort Key="Num" Direction="Ascending" />
</ContentElementConfig>
```

Example response

The following abbreviated example shows a returned list:

```
<cs:ContentElement xsi:type="cs:RecordList"
    Id="RecordList" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <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:ContentElement>
```

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:

```
<ProductName type="mdex:string">Bike Wash - Dissolver</ProductName>
```

This example shows a record with five managed attribute values:

```
<cs:Record>
   <Decimal cs:ValueName="53" type="mdex:string">/1-100/51-60/53</Decimal>
   <English type="mdex:string">five three</English>
   <FirstDigit cs:ValueName="5" type="mdex:string">/5</FirstDigit>
   <Hex cs:ValueName="0035" type="mdex:string">/0001-0100/0031-0040/0035</Hex>
   <LastDigit cs:ValueName="3" type="mdex:string">/3</LastDigit>
   <Num type="mdex:int">53</Num>
   <NumberOfDigits cs:ValueName="2" type="mdex:string">/2</NumberOfDigits>
   <Spanish type="mdex:string">cinco tres</Spanish>
</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.

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 10

Sorting Records

Sorting allows you to define the order of records returned with each navigation 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 store, 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 process 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 store 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 `RecordList` configuration that lets you specify a key to sort on and the sort direction. The `Sort` format is:

```
<Sort Key="keyName" Direction="dirOrder"/>
```

where:

- *keyName* is the name of a standard or managed attribute based on which sorting is performed.
- *dirOrder* is either `Ascending` for an ascending order or `Descending` for a descending order.

Note that the attribute name and the sort order are both case sensitive.

The following example shows an Ascending sort order based on the `ModelName` attribute:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
<ns:Request>
      <ns:ContentElementConfig xsi:type="ns:RecordListConfig"
          Id="RecordList"
          HandlerFunction="RecordListHandler"
          HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
          MaxPages="2">
          <ns:RecordsPerPage>10</ns:RecordsPerPage>
          <ns:Page>1</ns:Page>
          <ns:Sort Key="ModelName" Direction="Ascending"/>
          </ns:ContentElementConfig>
</ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

Troubleshooting application sort 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 a numeric when it is actually alphanumeric? Or vice versa? In this case, the Oracle Endeca Server returns a valid response, but the sorting may be incorrect.
- If a record has multiple attribute value assignments from a single attribute, the Oracle 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 used for sorting should only have one value assigned per record.
- If certain records in a data store 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 11

Record Filters

This chapter describes how to implement record filters in your front-end application.

About record filters

Record filters allow a front-end application powered by the Oracle Endeca Server to define arbitrary subsets of the total record set and dynamically restrict search and navigation results to these subsets.

For example, the catalog might be filtered to a subset of records appropriate to the specific end user or user role. The records might be restricted to contain only those visible to the current user based on security policies. Or, an application might allow end users to define their own custom record lists (that is, the set of parts related to a specific project) and then restrict search and navigation based on a selected list. Record filters enable these and many other application features that depend on applying Oracle Endeca Server search and navigation to dynamically defined and selected subsets of the data.

If you specify a record filter, whether for security, custom catalogs, or any other reason, it is applied 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.

Record filters support Boolean syntax using attribute values as base predicates and standard Boolean operators (AND, OR, and NOT) to compose complex expressions. For example, a filter can consist of a list of part number attribute values joined in a multi-way OR expression. Or, a filter might consist of a complex nested expression of ANDs, ORs, and NOTs on managed attribute IDs and attribute values.

Filter expressions are passed directly as part of a query to the Oracle Endeca Server. When a filter is selected, the set of visible records is restricted to those matching the filter expression. For example, record search queries will not return records outside the selected subset, and refinement values are restricted to lead only to records contained within the subset.

Note that all attribute values are automatically enabled for use in record filter expressions.

Finally, it is important to keep in mind that record filters are case-sensitive.

Record filter syntax

Record filters are specified with query-based expressions.

Record filters are specified directly within a query to the Oracle Endeca Server. The query-level syntax supports prefix-oriented Boolean functions (AND, OR, and NOT), colon-separated paths for standard attribute values, and forward-slash-separated paths for managed attribute values.

The following BNF grammar describes the syntax for query-level filter expressions:

```
<filter>      ::= <and-expr>
                | <or-expr>
                | <not-expr>
                | <literal>
<and-expr>    ::= AND(<filter-list>)
<or-expr>     ::= OR(<filter-list>)
<not-expr>    ::= NOT(<filter>)
<filter-list> ::= <filter>
                | <filter>,<filter-list>
<literal>     ::= <pval>
                | <dval-path>
<pval>        ::= <prop-key>:<prop-value>
<prop-key>    ::= <string>
<prop-value>  ::= <string>
<dval-path>   ::= <string>
                | <string>/<dval-path>
<string>      ::= any character string
```

The following six special reserved characters must be prepended with an escape character (\) for inclusion in a string:

```
( ) , : \ /
```

Example of a query-level filter expression

The following example illustrates a basic filter expression that uses nested Boolean operations:

```
OR(AND(Manufacturer:Sony, Product Category/Digital Camera),
   AND(Manufacturer:Aiwa,NOT(Product Category/Television)), Manufacturer:Denon)
```

This expression will match the set of records satisfying any of the following statements:

- Value for the Manufacturer attribute is `Sony` and record assigned managed attribute value is `Product Category/Digital Camera`.
- Value for Manufacturer is `Aiwa` and record is not assigned managed attribute value `Product Category/Television`.
- Value for Manufacturer attribute is `Denon`.

Using Boolean attributes

Filtering by Boolean attribute assignments is supported. You can specify the Boolean value as `true` (or its synonym of `1`), or as `false` (or its synonym of `0`). For example, assuming `isOdd` is a Boolean attribute, both `isOdd:1` and `isOdd:true` will parse properly and yield the same results.

Record filter result caching

The Dgraph process caches the results of record filter evaluations for re-use.

The cached results are used on subsequent queries as part of the global dynamic cache. The cache replacement policy is to discard least recently used (LRU) entries.

Requesting record filters with the API

The `RecordFilterOperator` complex type adds a record filter component to the filter state.

Record filters are not visible in the user interface of the front-end application (so they do not appear in breadcrumbs), and are not cleared by any of the general mechanisms for clearing the state. You can remove them using a `PopRecordFilterOperator`.

When making a query with the Conversation Web Service, you can use a `RecordFilterOperator` to limit the set of returned records, as in this example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
      <State />
      <Operator
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:type="RecordFilterOperator">
           <RecordFilter Name="ProductLine">OR(OR(ProductLine:Road),OR(ProductLine:Mountain))
           </RecordFilter>
      </Operator>
      <ContentElementConfig xsi:type="RecordListConfig"
       HandlerFunction="RecordListHandler"
       HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
       Id="RecordList" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
           <Column>ProductLine</Column>
           <RecordsPerPage>5</RecordsPerPage>
      </ContentElementConfig>
    </Request>
  </soap:Body>
</soap:Envelope>
```

This example, as many other examples in this guide, is based on a sample data store representing bicycles, their models, sellers, and other related information. In this example, a record filter is specified for records that have a `ProductLine` attribute with a value of `Mountain` or `Road`. Only those records are returned as a result of this request.

Examples with managed attribute values

You can use a record filter to perform a record query search so that only results tagged with a specified managed attribute value are returned. For example, say you have a managed attribute hierarchy that looks like this, where Sku is the managed attribute root and 123, 456, and 789 are leaf managed attribute values:

```
Sku
  123
  456
  789
  ...
```

To perform a record query search so that results tagged with any of these managed attribute values are returned, use the following:

```
OR(sku/123,OR(sku/456),OR(sku/789))
```

To perform a record query search so that only results tagged with the managed attribute value 123 are returned, use the following:

```
sku/123
```

Note that the / (forward slash) is used as the delimiter for managed attribute value paths.

Filtering data and non-data records

If you wish to perform some queries only against data records and other queries only against PDRs, or against schema and configuration records, you can use the `RecordKindOperator` type to restrict records to a particular kind.

The `RecordKindOperator` type contains the `RecordKind` element, which allows the following values:

Value	Description
`data`	Restricts records to actual data records that you load to the data store.
`properties`	Restricts records to those records that declare record attributes (PDRs).
`nondata`	Restricts records to those records that do not represent data records, and instead represent all system records (PDRs, DDRs, GCR), records that define groups, records that define precedence rules (if any are created), records that define other configuration, or records that define semantic entity definitions (if any are created).

If you use this operator, you can restrict the `State` from which all other search and filtering operations in the Endeca Server will be performed. It is useful to use the `RecordKindOperator` type 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 example illustrates how to specify the record kind with the `RecordKindOperator`:

```
<ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="ns:RecordKindOperator">
  <ns:RecordKind>data</ns:RecordKind>
</ns:Operator>
```

If a record kind is specified, it is used in the `State` for the subsequent request, as shown in this abbreviated example of the response:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
 <cs:Request>
    <cs:State>
      <ns3:RecordKind xmlns:ns2="http://www.endeca.com/MDEX/lql_parser/types"
        xmlns:ns3="http://www.endeca.com/MDEX/conversation/1/0"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        data
      </ns3:RecordKind>
    </cs:State>
  </cs:Request>
</cs:Results>
```

If a new record kind is specified, it replaces the one specified previously. Record kinds are not visible in the user interface — they do not appear in breadcrumbs, for example. You can remove them from the `State` by using a `PopRecordKindOperator`.

Record filter performance impact

Record filters can have an impact in some areas.

The evaluation of record filter expressions is based on the same technology that supports navigation queries in the Dgraph(s) of the Oracle Endeca Server. Because of this, there is no additional cost associated with using navigation attribute values in record filters.

Because expression evaluation is based on composition of information in the Oracle Endeca Server data files, most expressions of moderate size (that is, tens of terms/operators) do not add significantly to request processing time.

Furthermore, because the Dgraph process caches the results of record filter operations on an LRU (least recently used) basis, the costs of expression evaluation are typically only incurred on the first use of a record filter during a navigation session. However, some expected uses of record filters have known performance bounds, which are described below.

Record filters can impact the following areas:

- Spelling auto-correction and spelling Did You Mean
- Expression evaluation

Interaction with spelling auto-correction and spelling DYM

Record filters impose an extra cost on spelling auto-correction and spelling Did You Mean.

Expression evaluation

Expression evaluation of large OR filters and large scale negation can impose a performance impact on the system.

Because expression evaluation is based on composition of information stored in Oracle Endeca Server data files, most expressions of moderate size (that is, tens of terms and operators) do not add significantly to request processing time. Furthermore, because the Dgraph caches the results of record filter operations, the costs of expression evaluation are typically only incurred on the first use of a filter during a navigation session. However, some expected uses of record filters have known performance bounds, which are described in the following two sections.

Large OR filters

One common use of record filters is the specification of lists of individual records to identify data subsets (for example, custom part lists for individual customers, culled from a superset of parts for all customers).

The total cost of processing records can be broken down into two main parts: the parsing cost and the evaluation cost. For large expressions such as these, XML parsing performance dominates total processing cost.

XML parsing cost is linear in the size of the filter expression, but incurs a much higher unit cost than actual expression evaluation. Though lightweight, expression evaluation exhibits non-linear slowdown as the size of the expression grows.

OR expressions with a small number of operands perform linearly in the number of results, even for large result sets. While the expression evaluation cost is reasonable into the low millions of records for large OR expressions, parsing costs relative to total query execution time can become too large, even for smaller numbers of records.

Part lists beyond approximately one hundred thousand records generally result in unacceptable performance (10 seconds or more load time, depending on hardware platform). Lists with over one million records can take a minute or more to load, depending on hardware. Because results are cached, load time is generally only an issue on the first use of a filter during a session. However, long load times can cause other Dgraph requests to be delayed and should generally be avoided.

Large-scale negation

In most common cases, where the NOT operator is used in conjunction with other positive expressions (that is, AND with a positive attribute value), the cost of negation does not add significantly to the cost of expression evaluation.

However, the costs associated with less typical, large-scale negation operations can be significant. For example, while still sub-second, top-level negation filtering (such as "NOT availability=FALSE") of a record set in the millions does not allow high throughput (generally less than 10 operations per second).

If possible, attempt to rephrase expressions to avoid the top-level use of NOT in Boolean expressions. For example, in the case where you want to list only available products, the expression "availability=TRUE" will yield better performance than "NOT availability=FALSE".

Chapter 12

Using Range Filters

You can use range filters for navigation queries.

About range filters

Range 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 price range, say between $10 and $40.

It is important to remember that, similar to record search, range filters are simply modifiers for a navigation query. The range filter acts in the same manner as an attribute value, even though it is not a specific system-defined attribute value.

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.

Supported attribute types

Range filters can be applied to either standard attributes or managed attributes of the following supported types.

- Standard attributes of type Numeric (Integer, Long, Double, dateTime), type Geocode, or type Boolean
- Managed attributes of type Numeric that contain only Integer or Floating point values

For values of attributes of type Double, you can specify values using both decimal (0.00...68), and scientific notation (6.8e-10).

For standard attributes of type Boolean, `false` is interpreted to be less than `true`. If `isOdd` is a Boolean attribute, a query of `isOdd|LTEQ false` will return all records with assignments of `false` on attribute `isOdd`. Likewise, `isOdd|BTWN false true` will return all records with any assignment on `isOdd`.

No Dgraph process configuration flags are necessary to enable range filters. All numeric standard attributes and managed attributes and all geocode standard attributes are automatically enabled for use in range filters.

Implementing range filters in Studio

To use range filters in a Studio application, you must add a **Range Filters** component and configure at least one attribute.

After adding the **Range Filters** component, you can configure the component to select the attributes to be used for range filter queries.

For detailed information on using Studio, see the *Oracle Endeca Information Discovery Studio User's Guide*.

Implementing range filters with the API

This section describes how to issue range filter queries using the Conversation Web Service.

For additional information on the Conversation Web Service interface, see the *Oracle Endeca Server API Reference*.

Operator for range filters

A range filter search requires a `RangeFilterOperator` with a `Search` type.

The syntax for a `RangeFilterOperator` is shown in this example for a geocode range filter:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
      <State />
     <Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="RangeFilterOperator">
   <RangeFilter AttributeName="Location">
      <LowerBound Inclusive="false">10</LowerBound>
      <UpperBound Inclusive="false">20</UpperBound>
      <GeocodeReferencePoint Latitude="+42.365615"
          Longitude="-71.075647"/>
   </RangeFilter>
</Operator>
    ...
    </Request>
  </soap:Body>
</soap:Envelope>
```

The meanings of the `RangeFilter` elements and attributes are as follows:

RangeFilter Attribute	Meaning
`AttributeName`	The name of a standard attribute of type numeric or geocode, or of a managed attribute of type numeric on which range filtering will be performed. Only a single standard or managed attribute can be specified per range filter.
`LowerBound`	A numeric value that is the lower value of the range to search.
`UpperBound`	A numeric value that is the upper value of the range. This value must be equal to or greater than the lower value.

RangeFilter Attribute	Meaning
`Inclusive`	A Boolean that determines whether the values for the lower and/or upper bounds are excluded (`false`) or included (`true`) in the range. Note that geocode filters support only exclusive bounds.
`GeocodeReferencePoint`	Specifies latitude and longitude values for a geocode range filter.

Note: Multiple range filters can be applied to the same attribute. The results will include the intersection of the result sets of each range filter.

Less-than range filter format

The `UpperBound` element lets you make less-than range filter queries.

To make a less-than query, use only the `UpperBound` element. 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).

In addition, the `Inclusive` attribute determines whether the specified value is included in the range:

- If `Inclusive` is set to `false`, the value for the `UpperBound` element is exclusive. That is, the specified value for the `UpperBound` element is not included in the range.
- If `Inclusive` is set to `true`, the value for the `UpperBound` element is inclusive.

The default for the `Inclusive` attribute is `false` (that is, if you omit the attribute, the query will work as if you had specified `false` for this attribute).

Less-than example

The following is an example of an inclusive less-than query:

```
<cs:Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <cs:State/>
   <cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="RangeFilterOperator">
     <cs:RangeFilter AttributeName="ListPrice">
        <cs:UpperBound Inclusive="true">250</cs:UpperBound>
     </cs:RangeFilter>
   </cs:Operator>
   <cs:ContentElementConfig xsi:type="RecordListConfig"
      HandlerFunction="RecordListHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      Id="RecordList"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
</cs:Request>
```

This example returns all items whose price is up to, and including, $250.

In the example, if `Inclusive` had been set to `false`, the query would return all items whose price is up to, but not including, $250.

Greater-than range filter format

The `LowerBound` element lets you make greater-than range filter queries.

To make a greater-than query, use only the `LowerBound` element. 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).

In addition, the `Inclusive` attribute determines whether the specified value is included in the range:

- If `Inclusive` is set to `false`, the value for the `LowerBound` element is exclusive. That is, the specified value for the `UpperBound` element is not included in the range.
- If `Inclusive` is set to `true`, the value for the `LowerBound` element is inclusive.

The default for the `Inclusive` attribute is `false` (that is, if you omit the attribute, the query will work as if you had specified `false` for this attribute).

Greater-than example

The following is an example of an inclusive greater-than query against a bicycle data set:

```
<cs:Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <cs:State/>
   <cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="RangeFilterOperator">
     <cs:RangeFilter AttributeName="Score">
        <cs:LowerBound Inclusive="true">90</cs:LowerBound>
     </cs:RangeFilter>
   </cs:Operator>
   <cs:ContentElementConfig xsi:type="RecordListConfig"
      HandlerFunction="RecordListHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      Id="RecordList"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
</cs:Request>
```

This example returns all bicycles whose rating score is 90 and above, including those with a score of 90.

In the example, if `Inclusive` had been set to `false`, the query would include all bicycles whose rating score is greater than 90, but would not include bicycles with a score of 90.

Between range filter format

Use both `UpperBound` and `LowerBound` elements to construct between range filter queries.

A between range filter query returns records with a standard or managed attribute value of type numeric that falls between a lower bound (the `LowerBound` element) and an upper bound (the `UpperBound` element).

Between range filters must be inclusive. Therefore, the `Inclusive` attribute for both bound elements must be set to true, as shown in the example below.

Between example

The following is an example of a between range filter query:

```
<cs:Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <cs:State/>
   <cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="RangeFilterOperator">
     <cs:RangeFilter AttributeName="Price">
```

```
        <cs:LowerBound Inclusive="true">100</cs:LowerBound>
        <cs:UpperBound Inclusive="true">250</cs:UpperBound>
     </cs:RangeFilter>
   </cs:Operator>
   <cs:ContentElementConfig xsi:type="RecordListConfig"
      HandlerFunction="RecordListHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      Id="RecordList"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
</cs:Request>
```

This example returns all bicycles whose price is between $100 and $250. Because both bound elements are inclusive, the returned records include bicycles that cost $100 and $250.

Geocode range filter format

When used with a standard attribute of type geocode, the `GeocodeReferencePoint` element indicates a range filter based on the distance of that geocode attribute from a given reference point.

The syntax for a geocode range filter is shown in this snippet sample from the request:

```
<cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="RangeFilterOperator">
   <cs:RangeFilter AttributeName="Location">
       <cs:LowerBound>0</cs:LowerBound>
       <cs:UpperBound>20</cs:UpperBound>
       <cs:GeocodeReferencePoint Latitude="+42.365615"
           Longitude="-71.075647"/>
   </cs:RangeFilter>
   </cs:Operator>
   ...
</cs:Request>
```

The meanings of the `RangeFilter` elements and attributes are described in the following table:

RangeFilter Geocode Attribute	Meaning
`AttributeName`	The name of a standard attribute of type geocode. Only one standard attribute of type geocode can be specified per range filter.
`LowerBound`	Used to specify a greater-than distance (in kilometers) from the geocode attribute to the reference point.
`UpperBound`	Used to specify a less-than distance (in kilometers) from the geocode attribute to the reference point. Note that the range filter must contain at least one of the boundary elements.
`GeocodeReferencePoint`	Uses the `Latitude` and `Longitude` attributes to specify a reference point for range filtering of geocode attributes.
`Latitude`	The latitude of the location in whole and fractional degrees (positive values indicate north latitude and negative values indicate south latitude).

RangeFilter Geocode Attribute	Meaning
`Longitude`	The longitude of the location in whole and fractional degrees (positive values indicate east longitude and negative values indicate west longitude).

Note the following about geocode range filters:

- Distance limits (specified via the `LowerBound` and `UpperBound` elements) are exclusive. This means that you cannot use the `Inclusive` attribute set to `true` for these two elements. Because the `Inclusive` default is `false`, you can omit this attribute for geocode range filters.
- Geocode range filters must contain at least one of the bound elements.
- Distance limits in range filters are always expressed in kilometers.

The records are filtered by the distance from the filter key to the latitude/longitude pair.

Between geocode range filters

Use both `UpperBound` and `LowerBound` elements to indicate that the distance from the geocode attribute to the reference point is between the two bounds. The example at the beginning of this topic returns only records whose location (in the Location property) is between 0 and 20 miles from the reference point.

Less-than geocode range filters

If only the `UpperBound` element is used, the distance from the geocode attribute to the reference point will be less than the given amount. For example:

```
<Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="RangeFilterOperator">
   <RangeFilter AttributeName="Location">
       <UpperBound>20</UpperBound>
       <GeocodeReferencePoint Latitude="+42.365615"
          Longitude="-71.075647"/>
   </RangeFilter>
   </Operator>
```

This sample query returns all records whose location is less than 20 miles from the reference point.

Greater-than geocode range filters

If only the `LowerBound` element is used, the distance from the geocode attribute to the reference point will be greater than the given amount. For example:

```
<Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="RangeFilterOperator">
   <RangeFilter AttributeName="Location">
       <LowerBound>20</LowerBound>
       <GeocodeReferencePoint Latitude="+42.365615"
          Longitude="-71.075647"/>
   </RangeFilter>
</Operator>
```

This sample query returns all records whose location is greater than 20 miles from the reference point.

Removing range filter operators

The `PopRangeFilterOperator` removes a range filter component from the state.

The syntax of this operator is:

```
<Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="PopRangeFilterOperator">
  <RangeFilter AttributeName="ListPrice" />
</Operator>
```

The `AttributeName` attribute is the name of the numeric or geocode attribute for which a range filter component had been used.

If the call is successful, the following `Results` component is returned:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <cs:Request>
      <cs:State/>
   </cs:Request>
</cs:Results>
```

The empty `State` element shows that the range filter component was successfully removed.

Rendering the range filter results

The results of a range filter request can be rendered in the front-end application's user interface similar to any other navigation request.

Because a range filter request is simply a variation of a basic navigation request, rendering the results of a range filter request is identical to rendering the results of a navigation request.

Note, however, that there are no API components to access a list of valid range filter attributes. This is because the attributes do not need to be explicitly identified as valid for range filters in the same way that they need to be explicitly identified as valid for record search. Therefore, specific standard and managed attributes that a user is allowed to filter against must be correctly identified as having the type numeric or geocode in the data store.

Examples of range filter parameters

This topic shows some valid examples of range filter queries.

Consider the following examples that use these four records:

Record	Product line managed attribute value	ListPrice attribute
1	Mountain (Value 101)	200
2	Mountain (Value 101)	275
3	Road (Value 102)	175
4	Other (Value 103)	150

Example 1

Assume that the following query is created:

```
<cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="RangeFilterOperator">
  <cs:RangeFilter AttributeName="ListPrice">
     <cs:LowerBound Inclusive="true">200</cs:LowerBound>
  </cs:RangeFilter>
</cs:Operator>
```

This request has a range filter specifying the ListPrice attribute should be greater than 200 (with no managed attribute values specified). The following objects are returned:

- 2 records (records 1 and 2)

Example 2

This example uses the following query:

```
<cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="RangeFilterOperator">
  <cs:RangeFilter AttributeName="Mountain">
     <cs:UpperBound Inclusive="true">250</cs:UpperBound>
  </cs:RangeFilter>
</cs:Operator>
```

This request specifies the Mountain value (value 101) and a range filter specifying a price less than 250. The following objects are returned:

- 1 record (record 1)
- (No additional refinements)

Example 3

This query:

```
<cs:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="RangeFilterOperator">
  <cs:RangeFilter AttributeName="ListPrice">
     <cs:LowerBound Inclusive="false">140</cs:LowerBound>
     <cs:UpperBound Inclusive="false">180</cs:UpperBound>
  </cs:RangeFilter>
</cs:Operator>
```

would return records 3 and 4 from the sample record set.

Performance impact for range filters

Range filters impact the Dgraph response times, but not memory usage.

This feature does not impact the amount of memory needed by the Dgraph. However, because the feature is evaluated entirely at request time, the Dgraph process response times are directly related to the number of records being evaluated for a given range filter request. You should test your application to ensure that the resulting performance is compatible with the requirements of the deployment.

Part IV

Working with Refinements, Breadcrumbs, and Groups

Chapter 13

Working with Refinements

This chapter provides information on handling and displaying refinements in your Web application.

About refinements

Endeca standard and managed attributes are known as ***refinements***. Refinements can be used for navigation and search — when you select a specific refinement, you refine your result set.

Refinements contain refinement attribute values for the current record set. The attribute values for refinements are derived from both standard and managed attributes.

Standard and managed attributes are similar in that they both:

- Support navigation.
- Are usually generated from a record’s source attributes.
- Consist of key/value pairs (standard attribute name/standard attribute value, managed attribute name/managed attribute value).
- Can be searched and displayed using their display names.
- Can have a multi-level hierarchy.

Displaying refinements in Studio

Each component that is affected by the **Guided Navigation** component uses refinements and information received from refinements computation, such as the order of refinements or a number of refinements for a given attribute.

For additional information on configuring Studio components that use refinement information, including the **Guided Navigation** component, see the *Oracle Endeca Information Discovery Studio User's Guide*.

Configuring managed attributes for query refinement

You must configure a managed attribute for query refinement.

In the managed attribute's DDR (Dimension Description Record), the `mdex-dimension_EnableRefinements` attribute must be set to `true`, so that refinements will be displayed. If the configuration attribute is set to `false`, refinements will not be displayed (i.e., the managed attribute will be hidden).

If a managed attribute is created and used to classify records, but no records are classified with any corresponding values, that managed attribute will not be available as a refinement, because it is not related to the resulting record set in any way.

Notes

No configuration is needed for standard attributes. Assuming that the standard attribute is used to classify records, the corresponding refinement values will be available to create or refine a query.

There are no Dgraph configuration flags necessary to enable the basic displaying of refinements.

Configuring the global order of refinements

You configure the global order for the values of a refinement by sending this configuration request to the Oracle Endeca Server with the Configuration Web Service, or by using Integrator.

The `system-navigation_Sorting` attribute in the PDR for the selected attribute controls the global order of values in a refinement. This attribute can have the following values:

- **record-count** sorts refinement values in descending order, by the number of records available for each refinement. 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).

For information on the Configuration Web Service, see the section in this guide and the *Oracle Endeca Server API Reference*.

For information on how to configure the global order of refinements in Integrator, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

Configuring refinement counts

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.

The `system-navigation_ShowRecordCounts` attribute in the PDR specifies whether to show record counts for a refinement. The valid settings for this attribute 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.

About multi-select attributes

A multi-select attribute is an attribute that can be present on records multiple times, with different values, in a single navigation state.

There are two forms of multi-select attributes:

- If the navigation state contains multiple values from a multi-select-and attribute, then all of the records in that state contain all of the selected values for that attribute.
- If the navigation state contains multiple values from a multi-select-or attribute, then all of the records in that state contain at least one of the selected values.

This has consequences for the refinements that the Dgraph process of the Oracle Endeca Server generates as well as the navigation state: the Dgraph allows you to select additional values for a multi-select attribute.

In the Dgraph, you can configure an attribute as multi-select by changing the values of the `system-navigation_Select` attribute on the Property Description Record for this attribute.

The multi-select feature is only fully supported for those attributes (standard or managed) that do not contain a hierarchy.

Configuring multi-select attributes

You configure whether an attribute is multi-select by changing the value in the `system-navigation_Select` attribute of a PDR, using the Configuration Web Service or Integrator.

The `system-navigation_Select` attribute of a PDR can have the following settings:

- **multi-and** configures the attribute as a multi-select AND attribute. This means that a current navigation state represents the intersection of the records returned from the multiple selections of values on that attribute.
- **multi-or** configures the attribute as a multi-select OR attribute. This means that a current navigation state represents the union of the records returned from the multiple selections of values on that attribute.
- **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.

Handling multi-select attributes in an application

The behavior of multi-select attributes may require changes in the UI.

The fact that an attribute is tagged as multi-select should be transparent to the application developer. There is no special development required to enable multi-select attributes, and there are no query parameters that are specific to multi-select attributes.

However, the semantics of how the Dgraph process of the Oracle Endeca Server interprets navigation queries and returns available refinements changes once an attribute is tagged as multi-select. After tagging an attribute as multi-select, the Dgraph allows multiple attribute values from the same attribute to be added to the navigation state.

The Dgraph process behaves differently for the two types of multi-select managed attributes:

- Multi-select AND managed attributes. The Dgraph treats the list of attribute values selected from a `multi-and` attribute as a Boolean AND operation. That is, it will return all records that satisfy the Boolean AND of all the attribute values selected from a `multi-and` attribute (that is, all records that have been tagged with "Apple" AND "Apricot"). The Dgraph process 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-select OR managed attributes. A `multi-or` managed 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"). The Dgraph process will always return all attribute values for a `multi-or` attribute that have not already been selected – the set of refinements 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.

Avoiding dead-end query results

Be careful when rendering the selected managed attribute values of `multi-or` managed attributes. It is possible to create an interface that might result in dead ends when removing selected attribute values.

Consider this example: Managed 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 UI be designed so that the user must be forced to remove all values from a `multi-or` managed attribute when making changes to the list of selected attribute values.

Performance impact for multi-select managed attributes

Tagging an attribute as multi-select does not affect performance. However, when making decisions about when to tag an attribute as multi-select, keep the following in mind: Users will take longer to refine the list of results, because each selection from a multi-select managed attribute still allows for further refinements within that attribute. Also, refinements for `multi-or` attribute are more expensive.

Refinement counts for multi-or refinements

Refinement counts on a standard refinement that is multi-or indicate how many records in the result set will be tagged with the refinement if you select it. When there are no selections 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 which 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 1that 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).

About externally managed attributes

Web applications powered by the Oracle Endeca Server can use managed attributes created with a taxonomy management tool.

You can also import or otherwise access externally managed attributes. For details on loading these attributes, see the "Adding New Records" chapter in the *Oracle Endeca Server Data Loading Guide*.

Working with refinements using the API

This section provides examples of Conversation Web Service requests and responses that let you retrieve various aspects of the refinement configuration — refinements themselves, their order, counts, and special types of refinements, such as those that are multi-select.

Retrieving refinements with the API

Displaying standard and managed attribute refinement values is the core concept behind Guided Navigation.

The complex type `PropertyListConfig` returns a list of all available attributes in the system. It contains an element `Property`, 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 clients of the Conversation Web Service that prefer to read descriptor records directly.

The following example request illustrates how to obtain a list of attributes:

```
<ns:Request>
  <ns:ContentElementConfig
    Id="AttributeList" xsi:type="ns:PropertyListConfig"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    HandlerFunction="PropertyListHandler"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  </ns:ContentElementConfig>
</ns:Request>
```

Such a request returns information that describes all attributes, and includes all characteristics of the attributes.

The following example shows the characteristics of the managed attribute `ProductCategory`. Using this output, you can view the values of PDRs and DDRs for the attribute `ProductCategory`, indicating whether the attribute is searchable, and specifying other characteristics contained in the PDR and DDR for this managed attribute:

```
<cs:Property Key="ProductCategory" Type="mdex:string" Dimension="true"
     DisplayName="Product Category" Refinable="true">
   <cs:PropertyRecord>
     <mdex-property_DisplayName type="mdex:string">Product Category</mdex-property_DisplayName>
     <mdex-property_IsPropertyValueSearchable type="mdex:boolean">false<
/mdex-property_IsPropertyValueSearchable>
     <mdex-property_IsSingleAssign type="mdex:boolean">false</mdex-property_IsSingleAssign>
     <mdex-property_IsTextSearchable type="mdex:boolean">false</mdex-property_IsTextSearchable>
     <mdex-property_IsUnique type="mdex:boolean">false</mdex-property_IsUnique>
     <mdex-property_Key type="mdex:string">ProductCategory</mdex-property_Key>
     <mdex-property_TextSearchAllowsWildcards type="mdex:boolean">false<
/mdex-property_TextSearchAllowsWildcards>
     <mdex-property_Type type="mdex:string">mdex:string</mdex-property_Type>
     <system-navigation_Select type="mdex:string">single</system-navigation_Select>
     <system-navigation_ShowRecordCounts type="mdex:boolean">true<
/system-navigation_ShowRecordCounts>
     <system-navigation_Sorting type="mdex:string">lexical</system-navigation_Sorting>
   </cs:PropertyRecord>
   <cs:DimensionRecord>
     <mdex-dimension_EnableRefinements type="mdex:boolean">true</mdex-dimension_EnableRefinements>
     <mdex-dimension_IsDimensionSearchHierarchical type="mdex:boolean">false<
/mdex-dimension_IsDimensionSearchHierarchical>
     <mdex-dimension_IsRecordSearchHierarchical type="mdex:boolean">false<
/mdex-dimension_IsRecordSearchHierarchical>
     <mdex-dimension_Key type="mdex:string">ProductCategory</mdex-dimension_Key>
   </cs:DimensionRecord>
</cs:Property>
```

In addition, after a user creates a query using record and/or value search, only valid remaining refinement values are provided to the user to refine that query. This allows the user to reduce the number of matching records without creating an invalid query.

To display refinements (standard and managed attributes), they need to be requested, that is, included in the Conversation Web service request that describes the navigation menu. If the refinements belong to a group, this group needs to be requested.

The following example shows the request in which a group of refinements is requested. This example assumes that the group `Sales-Transaction` exists and includes the refinements (attributes) `CustomerPONumber`, `OrderQuantity`, and `UnitPrice`:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State/>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
    Id="NavigationMenu"
    HandlerFunction="NavigationMenuHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RefinementGroupConfig Name="Sales-Transaction" Expose="true"/>
  </ContentElementConfig>
</Request>
```

These refinements will be returned along with the group `Sales Transaction` in which they are included.

If you would like to retrieve refinements that are not explicitly included in any user-configured groups, you can request a group `system-navigation_InternalGroup`. This group exists in the Oracle Endeca Server and includes all refinements that are not members of any other groups.

Refinements configuration format

Use the `RefinementConfig` element of the Conversation Web service request to expose refinement values for standard and managed attributes.

The `RefinementConfig` element of the Conversation Web service request specifies which managed or standard attribute, out of all valid attributes returned with a navigation query, should return actual refinement values. Note that only the top-level refinement values are returned.

For managed attributes, if a managed attribute value is a parent, you can also use the `RefinementConfig` element with that managed attribute value and return its child attribute values (again, only the top-level child attribute values are returned).

The `RefinementConfig` element is included in the `RefinementGroupConfig`. This parent element returns refinements for groups.

RefinementConfig format

The basic `RefinementConfig` format is shown in this example:

```
<RefinementConfig
   Name="OrderQuantity"
   Spec="/"
   Expose="false"
   OrderByRecordCount="false"
   MaximumCount="100">
</RefinementConfig>
```

The descriptions of the attributes are:

Attribute	Description
`Name`	Required. The name of the standard or managed attribute value. Specifying a root value name of the managed attribute is the same as specifying a name of the managed attribute.
`Spec`	Optional. The standard or managed attribute value spec. For a hierarchical managed attribute, refinements will be returned for any child managed attribute values of this spec.
`Expose`	Optional. Specify `false` (the default) to just show the root refinement, or `true` to expose refinements.
`OrderByRecordCount`	Optional. Specify `true` to use dynamic ranking to order by record count, or `false` (the default) to use the default order from the Oracle Endeca Server.
`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 a value specified in the `MaximumRefinementCount` attribute in the `NavigationMenuConfig` element in the Conversation Web Service request. Further, if that value is not specified, the default is 10.

The `RefinementConfig` element is used in a `NavigationMenuConfig` element, as in this example. It exposes refinement values for the `OrderQuantity` attribute that is part of the group `Sales-Transaction`:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <State/>
   <ContentElementConfig xsi:type="NavigationMenuConfig"
     Id="NavigationMenu"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <RefinementGroupConfig Name="Sales-Transaction" Expose="true">
       <RefinementConfig Name="OrderQuantity" Expose="true" MaximumCount="100" />
     </RefinementGroupConfig>
   </ContentElementConfig>
</Request>
```

Note that you can use multiple `RefinementConfig` elements in a `RefinementGroupConfig`.

Notes on RefinementConfig

Keep in mind that the `RefinementConfig` element is an optional query parameter. However, attributes for which `RefinementConfig` is not included will not return refinements. 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/1/0">
```

```
  <State/>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
     Id="NavigationMenu"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RefinementGroupConfig Name="Sales-Characteristics" Expose="false"/>
  </ContentElementConfig>
  </ContentElementConfig>
</Request>
```

will return all attributes in the group but no refinement values. This is faster for the Oracle Endeca Server to compute, and returns only root values.

However, this query for the `ProductType` managed attribute:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State/>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
     Id="NavigationMenu"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <RefinementGroupConfig Name="Sales-Characteristics" Expose="true"/>
    <RefinementConfig Name="ProductType" Expose="true" />
    </ContentElementConfig>
</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 Oracle Endeca Server 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/1/0">
  <State/>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     Id="NavigationMenu"
     ExposeAllRefinements="true"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <RefinementGroupConfig Name="Sales-Characteristics" Expose="true"/>
  </ContentElementConfig>
</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/1/0">
   <State/>
   <ContentElementConfig xsi:type="NavigationMenuConfig"
      Id="NavigationMenu"
      HandlerFunction="NavigationMenuHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <RefinementGroupConfig Name="Sales-Characteristics" Expose="true">
       <RefinementConfig Name="UnitPrice" Expose="true" />
       <RefinementConfig Name="OrderQuantity" Expose="true" />
       <RefinementConfig Name="CustomerPONumber" Expose="true" />
      </RefinementGroupConfig>
   </ContentElementConfig>
</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.

Retrieving managed attributes that have refinements

The first step in displaying refinements is to retrieve those managed attributes that potentially have refinements.

Types of refinements

Refinement attributes contain attribute values for the current data store, including both *standard refinements* and *implicit refinements*.

- Standard refinements refine the result set when selected.
- Implicit refinements are managed attribute values that are assigned to all records in the current result set and whose selection, therefore, does not narrow the results.

Retrieving standard and implicit refinements

Refinements (both standard and implicit) are returned in a `NavigationMenu` content element, that in turn contains a`NavigationMenuItemGroup` element with `NavigationMenuItem` elements for each managed attribute with refinements.

This example shows the `NavigationMenuItem` element for the managed attribute:

```
<cs:NavigationMenuItem
    Name="WineType"
    Display Name="Wine Type"
    MultiSelect="Or" HasMore="false">
    <cs:ExposureControl Exposed="true">
     <cs:Operator
      OwnerId="NavMenu" xsi:type="cs:RefinementHideOperator"
      Name="Perfect"
      Spec="/"
      Group="Wine Characteristics"/>
    </cs:ExposureControl>
    <cs:Refinement Name="Red" Spec="/Red" Label="Red" Count="40">
      <cs:Operator xsi:type="cs:RefinementOperator" Name="Red" Spec="/Red"/>
    </cs:Refinement>
    <cs:Refinement Name="White" Spec="/White" Label="White" Count="50">
      <cs:Operator xsi:type="cs:RefinementOperator" Name="White" Spec="/White"/>
    </cs:Refinement>
   <cs:RootDimensionValue DimensionName="Sparkling" Spec="/"/>
</cs:NavigationMenuItem>
```

Each refinement is returned in a `Refinement` element, as shown in this example for the Red managed attribute value:

```
<cs:Refinement Name="Red" Spec="/Red" Label="Red" Count="18">
   <cs:Operator xsi:type="cs:RefinementOperator" Name="Red" Spec="/Red"/>
</cs:Refinement>
```

The `Count` element indicates that eighteen records would be in the result set if you were to refine on this attribute value.

Creating a new query from refinement attribute values

Once refinement values have been retrieved, these values typically are used to create additional refinement navigation queries.

First, consider this request in which the `WineType` refinement is requested and exposed:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
 <State/>
 <ContentElementConfig xsi:type="NavigationMenuConfig"
  Id="NavigationMenu"
  HandlerFunction="NavigationMenuHandler"
  HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
    <RefinementConfig Name="WineType" Expose="true"/>
   </RefinementGroupConfig>
  </ContentElementConfig>
</Request>
```

It returns the following query results. Notice that the query results show the `WineType` refinement and the refinement values on it — `White` and `Sparkling`.

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
 <cs:Request>
  <State xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <ContentElementConfig xsi:type="NavigationMenuConfig"
Id="NavigationMenu"
HandlerFunction="NavigationMenuHandler"
HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
xmlns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
    <RefinementConfig Name="WineType" Expose="true"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0">
   </RefinementGroupConfig>
  </ContentElementConfig>
  </cs:Request>
  <cs:ContentElement xsi:type="cs:NavigationMenu" Id="NavigationMenu"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <cs:NavigationMenuItemGroup Name="Wine Characteristics"
  HasRefinablePRoperties="true">
    <cs:NavigationMenuItem Name="WineType" DisplayName="WineType"
     MultiSelect="Or" HasMore="false">
        <cs:ExposureControl Exposed="true">
          <cs:Operator OwnerId="NavigationMenu"
          xsi:type="cs:RefinementHideOperator"
              Name="WineType"
              Spec="/"
              Group="Wine Characteristics"/>
         </cs:ExposureControl>
         <cs:Refinement Name="WineType" Spec="/Red" Label="Red"
          Count="18">
          <cs:Operator xsi:type="cs:RefinementOperator"
          Name="WineType" Spec="/Red"/>
         </cs:Refinement>
         <cs:Refinement Name="WineType" Spec="/White" Label="White"
          Count="40">
          <cs:Operator xsi:type="cs:RefinementOperator"
          Name="WineType" Spec="/Red"/>
         </cs:Refinement>
         <cs:Refinement Name="WineType" Spec="/Sparkling"
         Label="Sparkling" Count="50">
          <cs:Operator xsi:type="cs:RefinementOperator"
          Name="WineType" Spec="/Red"/>
         </cs:Refinement>
```

```
      <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
    </cs:NavigationMenuItem>
   </cs:NavigationMenuItemGroup>
  </cs:ContentElement>
</cs:Results>
```

Based on this result, a follow-on request creates an additional refinement navigation query. It uses the refinement operator to request `Red`, to let you further refine to `WineType Red`.

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <State/>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
Id="NavigationMenu"
HandlerFunction="NavigationMenuHandler"
HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0">
    <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
     <RefinementConfig Name="WineType" Expose="true"/>
    </RefinementGroupConfig>
   </ContentElementConfig>
  <Operator xsi:type="RefinementOperator" Name="WineType" Spec="/Red"/>
</Request>
```

Limiting the number of refinements to be displayed

If there are too many refinements to be returned per refinement, you can limit the number of displayed refinements.

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, the Oracle 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. (In order 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`).

If there are too many refinements to be returned per attribute, you can limit the number of them that are displayed using `NavigationMenuConfig` in the Conversation Web Service request.

You can do this in a global setting or per each refinement value. The following statements describe the logic used by the Conversation Web Service to identify the number of refinements to be displayed:

- Per attribute configuration. You can specify a number of refinements to display per attribute. For each, you can optionally specify the number in the attribute `MaximumCount` in the `RefinementConfig` element.
- Global configuration, for all attributes in a particular navigation menu. If the number for each attribute is not specified, the Conversation Web Service response uses the global setting you can specify in the `MaximumRefinementCount` attribute of `RefinementConfig` in `ContentElementConfig`. The setting is per content element, not per query.
- Further, if neither of the settings is specified, the number defaults to 10.

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.

```
<ContentElementConfig xsi:type="NavigationMenuConfig"
 HandlerFunction="NavigationMenuHandler"
 HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
 Id="NavigationMenu"
 MaximumRefinementCount="15"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
   <RefinementConfig Name="WineType" MaximumCount="40"/>
   <RefinementConfig Name="Year"/>
   <RefinementConfig Name="Score"/>
  </RefinementGroupConfig>
</ContentElementConfig>
```

This request returns up to 40 refinement values for `WineType`. It returns 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/1/0"
   xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <cs:Request>
    <FilterState xmlns="http://www.endeca.com/MDEX/conversation/1/0">
    <ContentElementConfig xsi:type="NavigationMenuConfig" Id="NavigationMenu"
      HandlerFunction="NavigationMenuHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
        xmlns="http://www.endeca.com/MDEX/conversation/1/0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
        <RefinementConfig Name="WineType" MaximumCount="1" Expose="true"
          xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"/>
      </RefinementGroupConfig>
    </ContentElementConfig>
  </cs:Request>
  <cs:ContentElement xsi:type="cs:NavigationMenu" Id="NavigationMenu"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <cs:NavigationMenuItemGroup Name="Wine Characteristics" HasRefinablePRoperties="true">
    <cs:NavigationMenuItem Name="WineType" DisplayName="WineType" MultiSelect="Or" HasMore="true">
         <cs:ExposureControl Exposed="true">
           <cs:Operator OwnerId="NavigationMenu"
              xsi:type="cs:RefinementHideOperator"
              Name="WineType" Spec="/"
              Group="Wine Characteristics"/>
         </cs:ExposureControl>
         <cs:Refinement Name="WineType" Spec="/Red" Label="Red" Count="18">
          <cs:Operator xsi:type="cs:RefinementOperator" Name="WineType" Spec="/Red"/>
         </cs:Refinement>
      <cs:RootDimensionValue DimensionName="WineType" Spec="/"/>
    </cs:NavigationMenuItem>
   </NavigationMenuItemGroup>
  </cs:ContentElement>
</cs:Results>
```

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` attribute. 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.

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:

`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. If set to `true`, the returned refinement contains the full path to its parent refinement values, as in `ProductCategory` > `CAT_COMPONENTS` > `Brakes`. 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 an already create attribute group "Product Categories":

```
<ContentElementConfig xsi:type="NavigationMenuConfig"
     Id="NavigationMenu"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     ReturnFullPath="true">
   <RefinementGroupConfig Name="Product Categories" Expose="true">
     <RefinementConfig Name="CAT_COMPONENTS" Expose="true" MaximumCount="3"/>
   </RefinementGroupConfig>
</ContentElementConfig>
```

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 from the Oracle Endeca Server on the same level within the attribute. For example, the Oracle 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.

The following example request in the Conversation Web Service illustrates how to retrieve a full path of hierarchical refinements for a standard or managed attribute:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State>
    <SelectedRefinementFilter Name="WineType" Spec="/Red"/>
  </State>
  <ContentElementConfig xsi:type="NavigationMenuConfig"
     Id="NavigationMenu"
     HandlerFunction="NavigationMenuHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     ReturnFullPath="true">
    <RefinementGroupConfig Name="Product Categories" Expose="true">
     <RefinementConfig Name="CAT_COMPONENTS" Expose="true" MaximumCount="3"/>
    </RefinementGroupConfig>
  </ContentElementConfig>
```

```
</Request>
```

The response returns a list of hierarchical refinements. It contains the attribute group information, and all the attributes from this group.

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 that is part of `ContentElementConfig`.

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:ContentElementConfig xsi:type="ns:RecordListConfig"
    Id="RecordList"
    HandlerFunction="RecordListHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    MaxPages="60">
   <ns:Column>ProductCategory</ns:Column>
   <ns:RecordsPerPage>200</ns:RecordsPerPage>
   <ns:Page>2</ns:Page>
   <ns:Sort Key="Description" Direction="Ascending"/>
</ns:ContentElementConfig>
```

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>
```

Retrieving the order of refinements with the API

A core capability of the Oracle 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.

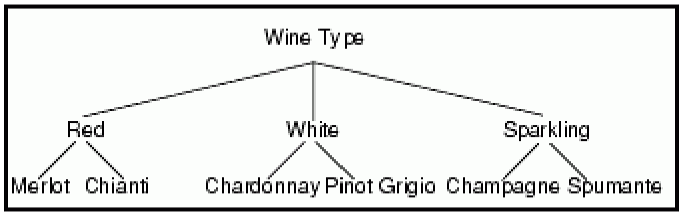
Using query-time control of refinement ordering

You can configure refinement ordering on a per-query basis.

The Oracle Endeca Server lets you 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 from which page 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/1/0">
   <State/>
   <ContentElementConfig xsi:type="NavigationMenuConfig"
      Id="NavigationMenu"
      HandlerFunction="NavigationMenuHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <RefinementGroupConfig Name="Wine Characteristics" Expose="true">
        <RefinementConfig
          Name="WineType"
          Expose="true"
          OrderByRecordCount="true"
          MaximumCount="100" />
      </RefinementGroupConfig>
   </ContentElementConfig>
</Request>
```

This setting overrides the setting for refinement order that you can specify in the `system-navigation_Sorting` in the PDR for a refinement.

Retrieving refinement counts with the API

The application UI can display the number of records returned for refinements.

Refinement counts represent the number of records (in the current navigation state) available beneath a given refinement value. These counts are dynamically computed at run-time by the Oracle Endeca Server and can be displayed in the user interface.

By providing the user with an indication of 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 application.

A *refinement count* is the number of records that would be in the result set if you were to refine on an attribute value.

By default, both standard and managed attributes are enabled for refinement counts. Therefore, no further configuration is needed to display record counts. So long as there are attribute values returned for a given request, refinement value statistics will be returned as an attribute attached to each attribute value.

You can, however, disable refinement statistics for attributes in the `system-navigation_ShowRecordCounts` property on the PDR.

Retrieving refinement counts for records

Record counts are returned in a `Count` attribute.

Each refinement is returned in a `Refinement` element, as shown in this example:

```
<cs:Refinement Name="Red" Spec="/Red" Label="2" Count="18">
   <cs:Operator xsi:type="cs:RefinementOperator" Name="Red" Spec="/Red"/>
</cs:Refinement>
```

In the example, a record count of 18 is returned for the Red attribute value.

About multi-select refinements

The Oracle Endeca Server supports two types of multi-select standard and managed attributes.

The default behavior of the Oracle Endeca Server permits only a single value from an attribute to be added to the navigation state. By default, after a user selects a leaf refinement from any managed attribute, that attribute is removed from the list of 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 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, you can give a user the ability to show wines that have a flavor of "Apple" and "Apricot".

The Oracle Endeca Server provides support for two types of multi-select standard and managed attributes that apply Boolean logic to the values selected:

- **multi-and**
- **multi-or**

You can tag the attribute as **multi-and**, or **multi-or** (or **single**) by changing the values of the `system-navigation_Select` attribute on the PDR that defines a particular attribute.

Performance impact for displaying refinements

Run-time performance of the Dgraph process 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.

Performance impact of refinement ordering

You can use the `--esampmin` option with the Dgraph process, to specify the minimum number of records to sample during refinement computation.

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 `--esampmin` 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.

Performance impact of refinement counts

Dynamic statistics on records are expensive computations for the Dgraph process of the Oracle Endeca Server.

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 that you are not using.

Chapter 14

Using Breadcrumbs

The chapter 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 data files of the particular data store can be selected as a breadcrumb.

Breadcrumbs honor record filters (such as security filters), but do not display them.

Breadcrumbs can reflect spelling correction and DYM information returned by the Dgraph process of the Oracle Endeca Server in response to keyword search queries.

In Studio, the **Breadcrumbs** 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 **Guided Navigation** component are reflected in the **Breadcrumbs** component:

- Once the user selects a refinement in the **Guided Navigation** component, it is reflected as a breadcrumb in the **Breadcrumbs** component.
- The user can select an additional breadcrumb in the **Guided Navigation** component, thereby narrowing down the scope of the record set for the query.
- Alternatively, the user can remove a refinement value from the **Breadcrumbs** 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 Reference*. This reference contains documentation generated from the interface WSDL document.

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.

The request for breadcrumbs is implemented with the `ContentElementConfig` element with the `BreadcrumbHandler`:

```
<ContentElementConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="BreadcrumbConfig"
  ReturnFullPath="false"
  HandlerFunction="BreadcrumbHandler"
  HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
  Id="Breadcrumbs"/>
```

This element includes:

`ReturnFullPath`	Specifies whether to return the full path of hierarchical refinements with the response. This setting is relevant only in navigation queries that request breadcrumbs; it is ignored in search or range filter queries requesting breadcrumbs. If set to `true`, the returned breadcrumb contains the full path to its parent refinement values, as in `Wine` > `Red` > `Merlot`. If set to `false`, returns only the refinement, without the path to its ancestors. The default is `false`.
`BreadcrumbHandler`	Is the function that facilitates breadcrumb generation in the response. This function is required to return breadcrumbs.

If spelling is enabled in the data store configuration, and in addition to breadcrumbs, you want the Conversation Web Service response to contain supplemental information about spelling suggestions and DYM, a second `ContentElementConfig` with `SearchAdjustmentHandler` is required. If this element is included, spelling correction or DYM suggestions are returned with the breadcrumbs in the response.

Note: If spelling is enabled, spelling correction occurs for breadcrumb results even if `ContentElementConfig` with `SearchAdjustmentHandler` 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).

To request breadcrumbs for a navigation query:

1. In the Conversation Web Service request, specify the following:
   - The selection for a specific refinement.
   - The `ContentElementConfig` element for `BreadcrumbHandler`.
   - (Optional). The `ContentElementConfig` element for `SearchAdjustments`.

   In this example, the navigation state includes a selection of the `NumberOfDigits` refinement, and two `ContentElementConfig` elements, one for `BreadcrumbHandler` and one for `SearchAdjustmentHandler`:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State/>
  <Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="RefinementOperator" Spec="/2"
    Name="NumberOfDigits"/>
  <ContentElementConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="BreadcrumbConfig"
    ReturnFullPath="true" HandlerFunction="BreadcrumbHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0" Id="Breadcrumbs"/>
  <ContentElementConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="SearchAdjustmentConfig"
    HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments"/>
  <PassThrough> ...</PassThrough>
</Request>
```

The Conversation Web Service result includes the original request with operators for `BreadcrumbHandler` and `SearchAdjustmentHandler` applied, followed by the `ContentElementConfig` element that lists attribute values identified as breadcrumbs, based on the user-selected navigation state.

Note: The result also includes the `GeneralizationOperator`. It enables the removal of the refinement (and thus the breadcrumb) in the user interface of the front-end application powered by the Oracle Endeca Server (such as Studio), if the user chooses to remove the previously selected refinement from the breadcrumb list.

The first half of the response repeats the request, as follows:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
  <cs:Request>
  <cs:State>
  <cs:SelectedRefinementFilter Name="NumberOfDigits" Spec="/2"/>
  </cs:State>
  <ContentElementConfig xmlns="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="BreadcrumbConfig" ReturnFullPath="true"
    HandlerFunction="BreadcrumbHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0" Id="Breadcrumbs"/>
  <ContentElementConfig xmlns="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchAdjustmentConfig" HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments"/>
  </cs:Request>
```

The second half of the response includes breadcrumbs:

```
<cs:ContentElement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="cs:Breadcrumbs" Id="Breadcrumbs">
  <cs:RefinementBreadcrumb Name="NumberOfDigits"
   DisplayName="Number Of Digits" Spec="/2">
  <cs:DimensionValue>
  <cs:DimensionValue DimensionName="NumberOfDigits" Spec="/">
   NumberOfDigits
  </cs:DimensionValue>
  <cs:Operator xsi:type="cs:GeneralizationOperator" Name="" Spec="/"/>
  </cs:DimensionValue>
  <cs:DimensionValue>
  <cs:DimensionValue DimensionName="NumberOfDigits" Spec="/2">2
  </cs:DimensionValue>
  <cs:Operator xsi:type="cs:GeneralizationOperator" Name="" Spec="/2"/>
  </cs:DimensionValue>
  <cs:Operator xsi:type="cs:GeneralizationOperator"
   Name="NumberOfDigits" Spec="/2"/>
  </cs:RefinementBreadcrumb>
  </cs:ContentElement>
 <cs:ContentElement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="cs:SearchAdjustments" Id="SearchAdjustments"/>
</cs:Results>
```

Retrieving breadcrumbs in a search query

Breadcrumbs returned by the Conversation Web Service in response to a search query can reflect spelling correction and DYM (Did You Mean) information.

The following requirements must be met to implement breadcrumbs that also return spelling correction and DYM information in response to a search query:

- The spelling must be enabled in the data store. To enable spelling, after you install the Oracle Endeca Server and create a data store, run the `admin?op=updateaspell` command.
- The request must include the `ContentElementConfig` element for the `BreadcrumbHandler`. This ensures that breadcrumbs are returned:

  ```
  <ContentElementConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="BreadcrumbConfig"
    ReturnFullPath="false"
    HandlerFunction="BreadcrumbHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="Breadcrumbs" />
  ```

- If you would like to return DYM and spelling correction results with breadcrumbs, the request must include the `ContentElementConfig` element for the `SearchAdjustmentHandler`:

  ```
  <ContentElementConfig
    xmlns="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchAdjustmentConfig"
    HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments" />
  ```

In the response, breadcrumbs are returned in the order in which they were added.

Example of breadcrumbs with spelling correction

Breadcrumbs information returned by the Conversation Web Service can reflect spelling correction. The following example illustrates this case.

To request breadcrumbs in a search query that returns spelling correction, specify the search keyword in the request, and these two types — `BreadcrumbConfig` and `SearchAdjustmentsConfig`. These types are subtypes of the `ContentElementConfig`.

The request in this example specifies a navigation state that includes a search for a user-entered word `fife`. It illustrates a search request with a breadcrumb that needs to be corrected for spelling:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
 <State/>
   <Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchOperator" Within="false">
    <SearchFilter Mode="All" Key="English">
     fife
    </SearchFilter>
   </Operator>
 <ContentElementConfig
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="BreadcrumbConfig" ReturnFullPath="true"
   HandlerFunction="BreadcrumbHandler"
   HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
   Id="Breadcrumbs"/>
 <ContentElementConfig
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="SearchAdjustmentConfig"
   HandlerFunction="SearchAdjustmentHandler"
   HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
   Id="SearchAdjustments"/>
 <PassThrough>...</PassThrough>
</Request>
```

The response from the Conversation Web Service contains the original request with search filter operators applied, the original (not yet spelling-corrected) term `fife`, and the `PopSearchOperator` needed for removing the refinement (if the user decides to remove this breadcrumb). Finally, the response also includes the automatically corrected term `five` in the `ContentElement` for `AppliedAdjustment`.

The first half of the response repeats the request:

```
<cs:Results
  xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <cs:Request>
       <cs:State>
         <SearchFilter
         xmlns="http://www.endeca.com/MDEX/conversation/1/0"
          xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          Mode="All"
          Key="English">
           fife
         </SearchFilter>
       </cs:State>
       <ContentElementConfig
         xmlns="http://www.endeca.com/MDEX/conversation/1/0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="BreadcrumbConfig" ReturnFullPath="true"
         HandlerFunction="BreadcrumbHandler"
         HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0" Id="Breadcrumbs"/>
       <ContentElementConfig
         xmlns="http://www.endeca.com/MDEX/conversation/1/0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="SearchAdjustmentConfig"
         HandlerFunction="SearchAdjustmentHandler"
         HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0" Id="SearchAdjustments"/>
    </cs:Request>
```

The second half of the response returns the automatically corrected term:

```
<cs:ContentElement
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="cs:Breadcrumbs" Id="Breadcrumbs">
       <cs:SearchBreadcrumb DisplayName="English">
         <cs:SearchFilter Key="English" Mode="All">
             fife
         </cs:SearchFilter>
       <cs:Operator xsi:type="cs:PopSearchOperator">
           <cs:SearchFilter Key="English" Mode="All">
             fife
           </cs:SearchFilter>
       </cs:Operator>
      </cs:SearchBreadcrumb>
     </cs:ContentElement>
          <cs:ContentElement
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:type="cs:SearchAdjustments" Id="SearchAdjustments">
         <cs:AppliedAdjustment>
             <cs:SearchFilter Key="English" Mode="All">
                fife
             </cs:SearchFilter>
         <cs:AdjustedTerms>
           five
         </cs:AdjustedTerms>
        </cs:AppliedAdjustment>
     </cs:ContentElement>
</cs:Results>
```

Example of breadcrumbs with DYM

Breadcrumbs information returned by the Conversation Web Service can reflect DYM suggestions. The following example illustrates this case.

To request breadcrumbs in a search query that returns DYM suggestions, specify the keyword search entry, and the `BreadcrumbConfig` and `SearchAdjustmentConfig` types of the `ContentElementConfig` complex type.

The following example of a request contains a keyword search `jane`:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State />
  <Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchOperator" Within="false">
   <SearchFilter Mode="All" Key="Essay">
    jane
   </SearchFilter>
  </Operator>
  <ContentElementConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="BreadcrumbConfig" ReturnFullPath="true"
    HandlerFunction="BreadcrumbHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="Breadcrumbs" />
  <ContentElementConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchAdjustmentConfig"
    HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments" />
</Request>
```

The response reflects DYM results. In this example, the response includes the request with operators applied, followed by a DYM suggested term, can and by the `ApplySpellingSuggestionOperator` that actually replaces the keyword with the term suggested with DYM.

The first half of the response repeats the request:

```
<cs:Results
  xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
 <cs:Request>
   <cs:State>
    <SearchFilter xmlns="http://www.endeca.com/MDEX/conversation/1/0"
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      Mode="All" Key="Essay">
      jane
    </SearchFilter>
   </cs:State>
  <ContentElementConfig
    xmlns="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="BreadcrumbConfig" ReturnFullPath="true"
    HandlerFunction="BreadcrumbHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="Breadcrumbs" />
  <ContentElementConfig
    xmlns="http://www.endeca.com/MDEX/conversation/1/0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="SearchAdjustmentConfig"
    HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments" />
 </cs:Request>
```

The second half of the response includes the information for DYM:

```
<cs:ContentElement
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="cs:Breadcrumbs" Id="Breadcrumbs">
  <cs:SearchBreadcrumb DisplayName="Essay">
     <cs:SearchFilter Key="Essay" Mode="All">
         jane
     </cs:SearchFilter>
     <cs:Operator xsi:type="cs:PopSearchOperator">
       <cs:SearchFilter Key="Essay" Mode="All">
         jane
       </cs:SearchFilter>
     </cs:Operator>
   </cs:SearchBreadcrumb>
  </cs:ContentElement>
  <cs:ContentElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="cs:SearchAdjustments" Id="SearchAdjustments">
    <cs:SuggestedAdjustment RecordCountIfApplied="15">
        <cs:SearchFilter Key="Essay" Mode="All">
          jane
        </cs:SearchFilter>
        <cs:SuggestedTerms>
          can
       </cs:SuggestedTerms>
      <cs:Operator xsi:type="cs:ApplySpellingSuggestionOperator">
              <cs:SearchFilter Key="Essay" Mode="All">
             jane
        </cs:SearchFilter>
             <cs:Replacement>
         can
      </cs:Replacement>
     </cs:Operator>
    </cs:SuggestedAdjustment>
  </cs:ContentElement>
</cs:Results>
```

Chapter 15

Using Attribute Groups

This chapter discusses how to implement attribute groups.

About attribute groups

Attribute groups are ordered collections 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.

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.

Configuring and using attribute groups in Studio

In Studio, lists of attributes are displayed in attribute groups.

This includes:

- For power users, when configuring Studio components
- For end users, when viewing components such as the **Guided Navigation** component

From the **Attribute Settings** component, power users can:

- Create and delete attribute groups

  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. However, the recommended way to create groups is through Studio.

- 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*.

Retrieving attribute group information with the API

You can use Conversation Web Service requests to request groups or lists of groups from the Oracle Endeca Server data store.

The Conversation Web Service returns groups for those types of queries that return refinements (standard or managed attributes). Any attributes returned from the Conversation Web Service as refinements are returned as part of groups.

This section provides examples for how to retrieve groups and lists of groups. For additional information about the Conversation Web Service WSDL, see the *Oracle Endeca Server API Reference*.

Retrieving groups

Any request that asks for refinements is also requesting groups, if the attributes that are going to be returned are configured as part of 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_v1_0:RefinementConfig"/>
 </sequence>
    <attribute name="Name" type="cs_v1_0:NonEmptyString" use="required"/>
    <attribute name="Expose" type="boolean" use="required"/>
    <attribute name="ExposeAllPropertyRefinements" type="boolean"/>
 </complexType>
```

Note: The type `"cs_v1_0:RefinementConfig"` indicates the version of the web service. In this example, the version is 1.0. 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.

Attribute	Description
`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. **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` (if this attribute is not specified). 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_v1_0:NavigationMenuItem"/>
 </sequence>
   <attribute name="HasRefineableProperties" type="boolean"/>
   <attribute name="Name" type="string" use="required"/>
</complexType>
```

Note: The type `"cs_v1_0:RefinementConfig"` indicates the version of the web service. In this example, the version is 1.0. It may or may not correspond to the version of the Conversation Web Service that you are using and that is currently supported.

The 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, `FlavorGroup` and `ProvenanceGroup`, but exposing top-level properties is requested for `FlavorGroup` only:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
  <State/>
  <ContentElementConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="NavigationMenuConfig"
    MaximumRefinementCount="10"
    ReturnFullPath="true"
    ExposeAllRefinements="false"
    HandlerFunction="NavigationMenuHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="Navigation">
    <RefinementGroupConfig Name="FlavorGroup" Expose="true">
       <RefinementConfig Name="Flavors" Expose="true" MaximumCount="2"/>
    </RefinementGroupConfig>
    <RefinementGroupConfig Name="ProvenanceGroup" Expose="false"/>
  </ContentElementConfig>
</Request>
```

The Conversation Web Service result includes results for one group, `FlavorGroup`, for which refinements were requested to be exposed:

```
<cs:ContentElement xsi:type="cs:NavigationMenu" Id="Navigation">
  <cs:NavigationMenuItemGroup Name="FlavorGroup" HasRefineableProperties="true">
    <cs:NavigationMenuItem Name="Flavors" DisplayName="Flavors" MultiSelect="And" HasMore="true">
      <cs:ExposureControl Exposed="true">
        <cs:Operator OwnerId="Navigation" xsi:type="cs:RefinementHideOperator"
           Name="Flavors" Group="FlavorGroup" Spec="/"/>
      </cs:ExposureControl>
      <cs:Refinement Name="Flavors" Spec="Currant" Label="Currant">
        <cs:Operator xsi:type="cs:RefinementOperator" Name="Flavors" Spec="Currant"/>
      </cs:Refinement>
      <cs:Refinement Name="Flavors" Spec="Oak" Label="Oak">
        <cs:Operator xsi:type="cs:RefinementOperator" Name="Flavors" Spec="Oak"/>
      </cs:Refinement>
      <cs:RootDimensionValue DimensionName="Flavors" Spec="/"/>
      <cs:FullPath><!-- path information omitted in this example--></cs:FullPath>
    </cs:NavigationMenuItem>
    <cs:NavigationMenuItem Name="Drinkability" DisplayName="Drinkability"
        MultiSelect="None" HasMore="true">
      <cs:ExposureControl Exposed="false">
        <cs:Operator OwnerId="Navigation" xsi:type="cs:RefinementExposeOperator"
           Name="Drinkability" Group="FlavorGroup" Spec="/"/>
      </cs:ExposureControl>
      <cs:RootDimensionValue DimensionName="Drinkability" Spec="/"/>
      <cs:FullPath><!-- path information omitted in this example --></cs:FullPath>
      </cs:NavigationMenuItem>
    </cs:NavigationMenuItemGroup>
  <cs:NavigationMenuItemGroup Name="ProvenanceGroup" HasRefineableProperties="true"/>
</cs:ContentElement>
```

Retrieving lists of groups

To retrieve a lists of groups, use a request with `AttributeGroupListConfig` which, as an extension of `ContentElementConfig` complex type, provides information about attribute groups.

To retrieve a list of groups:

1. Use the request similar to the following example:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <State/>
   <ContentElementConfig xsi:type="AttributeGroupListConfig"
     Id="AttributeGroupList"
```

```
      HandlerFunction="AttributeGroupListHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    </ContentElementConfig>
</Request>
```

The Conversation Web Service request contains a list of groups that are currently defined, specifying a group's display name and a number of attributes in each within each group. Information about each group is returned inside the `GroupSummary` element of the `ContentElement` response.

In this example, two groups are returned — `Sale-Geography`, and `Sales-Transaction`. The attribute `Cardinality` specifies the number of attributes in each of these groups; the attributes for each group are also listed.

The first half of the response repeats the request:

```
<cs:Results xmlns:cs="http://www.endeca.com/MDEX/conversation/1/0"
  xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <cs:Request>
     <ns3:State xmlns:ns2="http://www.endeca.com/MDEX/lql_parser/types"
        xmlns:ns3="http://www.endeca.com/MDEX/conversation/1/0"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"/>
     <ns3:ContentElementConfig
        xsi:type="ns3:AttributeGroupListConfig" Id="AttributeGroupList"
        HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
        HandlerFunction="AttributeGroupListHandler"
        xmlns:ns3="http://www.endeca.com/MDEX/conversation/1/0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
   </cs:Request>
```

The second half of the response includes information about groups and their contents:

```
<cs:ContentElement
     xsi:type="cs:AttributeGroupList"
     Id="AttributeGroupList" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <cs:GroupSummary Key="Sale-Geography" Cardinality="7">
       <cs:Record>
         <system-group_DisplayName type="mdex:string">
            Sale Geography
         </system-group_DisplayName>
         <system-group_Key type="mdex:string">
            Sale-Geography
         </system-group_Key>
       </cs:Record>
       <cs:GroupMembers>
         <mdex-property_Key>DimGeography_CountryRegionName</mdex-property_Key>
         <mdex-property_Key>DimGeography_StateProvinceName</mdex-property_Key>
         <mdex-property_Key>DimGeography_City</mdex-property_Key>
         <mdex-property_Key>DimGeography_PostalCode</mdex-property_Key>
         <mdex-property_Key>DimSalesTerritory_SalesTerritoryCountry</mdex-property_Key>
         <mdex-property_Key>DimSalesTerritory_SalesTerritoryGroup</mdex-property_Key>
         <mdex-property_Key>DimSalesTerritory_SalesTerritoryRegion</mdex-property_Key>
       </cs:GroupMembers>
    </cs:GroupSummary>
    <cs:GroupSummary Key="Sales-Transaction" Cardinality="15">
       <cs:Record>
         <system-group_DisplayName type="mdex:string">Sales Transaction</system-group_DisplayName>
         <system-group_Key type="mdex:string">Sales-Transaction</system-group_Key>
       </cs:Record>
       <cs:GroupMembers>
         ...
         <mdex-property_Key>FactSales_CarrierTrackingNumber</mdex-property_Key>
         <mdex-property_Key>FactSales_CustomerPONumer</mdex-property_Key>
       </cs:GroupMembers>
    </cs:GroupSummary>
  </cs:ContentElement>
```

```
</cs:Results>
```

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.

Chapter 16

Using Precedence Rules

This chapter 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 are able to 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.

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 store 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 store.

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 managed value specified as the trigger, or any of its descendants, are in the navigation state, 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. For details on `isLeafTrigger` usage, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

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.

Configuring precedence rules

You configure precedence rules in the Oracle Endeca Server by loading them in Integrator. Alternatively, you can create a rule in the data store using the `config-service:putPrecedenceRules` operation of the Configuration Web Service.

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.

To configure precedence rules:

1. Create an input source file (such as a text file or a CSV file) that defines your precedence rules.
2. In Integrator, create a graph that will read the input file, create the precedence rules, and send them to the Oracle Endeca Server using the `config-service:putPrecedenceRules` operation from the Configuration Web Service through **WebClient** component in Integrator.

The *Oracle Endeca Information Discovery Integrator Components Guide* provides details on creating the precedence rules and loading them into the Oracle Endeca Server.

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.

Part V

Using Search Features

Chapter 17

Using Record Search

This chapter 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 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.

Example of record search

For example, consider the following records:

Rec ID	Attribute value (BikeType)	Name of attribute	Description of attribute
1	Road Bikes (Value 2)	Road-450	can do double-duty for racing or long-range mileage...
2	Road Bikes (Value 2)	Road-550-W	its speed comes at the sake of comfort...
3	Touring Bikes (Value 3)	Touring-1000	combines comfort and performance...

Rec ID	Attribute value (BikeType)	Name of attribute	Description of attribute
4	Mountain Bikes (Value 1)	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.

Features for controlling record search

The following statements describe various aspects of record search behavior and how you can control it:

- To configure run-time record search behavior, you must create one or more search interfaces. For more information, see the section on search interfaces.
- 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.
- Multiple Dgraph configuration flags are available to manage different controls for record search, such as spelling support and relevance ranking. See the specific feature sections for details.

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 (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.

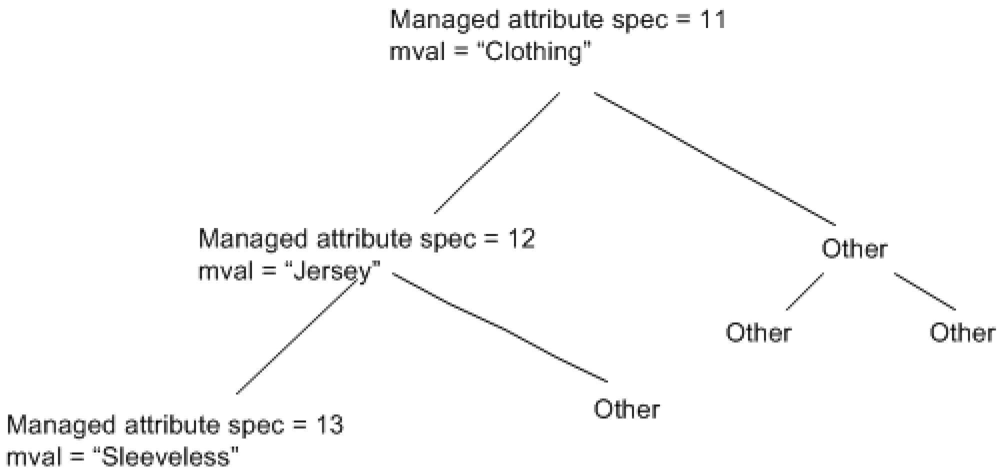
To configure an attribute for record search, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

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. As part of this behavior, record search does not consider implicit ancestor attribute values.

For example, consider the following managed attributes hierarchy:


Managed attribute spec = 11
mval = "Clothing"
Managed attribute spec = 12
mval = "Jersey"
Other
Other
Other
Other
Managed attribute spec = 13
mval = "Sleeveless"

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).

Adding search synonyms to attribute values

You can add synonyms to a managed attribute value so that users can search for other text strings and still get the same records as a search for the original attribute value name.

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.

You can add synonyms to an attribute value so that users can search for other text strings and still get the same records as a search for the original attribute value name. Synonyms can be added only to child attribute values, not to root attribute values.

You can use the Data Ingest Web Service's `ingestDimensionValues` operation to add synonyms when adding attribute values to the data store in the Oracle Endeca Server. For details, see the *Oracle Endeca Server Data Loading Guide*.

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 Reference*. This reference contains documentation generated from the interface WSDL document.

Obtaining the available search keys

The `AvailableSearchKeys` complex type lets you retrieve a list of the searchable attributes and search interfaces available in the data store.

The `AvailableSearchKeys` element contains one or more `AvailableSearchKey` elements. The complex type `AvailableSearchKey` identifies the items that are searchable — search interfaces and searchable properties. This type has the following format:

```
<complexType name="AvailableSearchKey">
  <annotation>
    <documentation>
           A key used to identify searchable properties and search interfaces.
    </documentation>
  </annotation>
  <sequence>
    <element name="Key" type="string"/>
    <element name="DisplayName" type="string" />
  </sequence>
  <attribute name="Interface" type="boolean" use="required" />
</complexType>
```

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`.

Request for available search keys

To make a request for available search keys, use the `AvailableSearchKeysConfig` component as illustrated in this example:

```
<Request xmlns="http://www.endeca.com/MDEX/conversation/1/0">
   <State/>
   <ContentElementConfig xsi:type="AvailableSearchKeysConfig"
      HandlerFunction="AvailableSearchKeysHandler"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      Id="MySearchKeys" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
```

```
</Request>
```

The `Id` attribute is an identifier for the configuration.

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/1/0"
     xmlns:mdex="http://www.endeca.com/MDEX/XQuery/2009/09">
   <cs:Request>
      <State xmlns="http://www.endeca.com/MDEX/conversation/1/0"
         xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"/>
      <ContentElementConfig xsi:type="AvailableSearchKeysConfig"
        HandlerFunction="AvailableSearchKeysHandler"
        HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
        Id="MySearchKeys" xmlns="http://www.endeca.com/MDEX/conversation/1/0"
         xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
   </cs:Request>
   <cs:ContentElement xsi:type="cs:AvailableSearchKeys"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <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:ContentElement>
</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). If the search key is a search interface, the `Interface` attribute is set to `true`.

In this sample response, one search interface, `AllWineSearch`, and two attributes, `Description` and `WineType`, are listed as available search keys.

Record search operator

A basic record search requires an `Operator` with a `SearchOperator` type.

The syntax for a search request is shown in this example:

```
<ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0" xsi:type="ns:SearchOperator"
   Within="false">
  <ns:SearchFilter Mode="AllPartial" RelevanceRankingStrategy="numfields"
      Key="Description">Mountain</ns:SearchFilter>
</ns:Operator>
```

The text content of the `SearchFilter` element contains the search term(s). In the example, a record search is being made for the "spice flavors" keywords. The meanings of attributes for the `SearchOperator` and the `SearchFilter` element are as follows:

Search attribute	Description
`Within`	Optional. If set to `false` on the `SearchOperator` element, then the visible parts of the filter state are cleared first. The default value is `false`.
`Key`	Required to be specified for the `SearchFilter`. 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` on the `SearchFilter` element, enables snippeting. If set to `false`, disables snippeting.
`SnippetLength`	Optional. Specifies the length of the snippet for the `SearchFilter`.
`Mode`	Optionally specifies a match mode for the `SearchFilter`. Cannot use `Boolean` match mode.
`RelevanceRankingStrategy`	Optionally specifies a relevance ranking strategy for the `SearchFilter`.

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. Auto correction (spelling correction and automatic phrasing)
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, custom catalogs, or any other reason, the Oracle 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: Auto correction (spelling correction and automatic phrasing)

If spelling correction and automatic phrasing are enabled and triggered, the Dgraph process of the Oracle Endeca Server 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 the section about it.

If automatic phrasing is enabled, then the Dgraph automatically combines distinct query terms that match a phrase in the phrase dictionary into a search phrase.

Once distinct terms are grouped as an automatic phrase, the phrase is not subject to additional thesaurus expansion and stemming.

For example, suppose the phrase dictionary contains two phrases `Kenneth Cole` and also `blue jeans`. If the query is `Kenneth Cole blue jeans`, the alternative query might be `"Kenneth Cole" "blue jeans"`.

For details on automatic phrasing, see the section about it.

Step 4: Thesaurus expansion

The tokenized query, as well as each query alternative generated by spelling suggestion, is expanded by the Oracle Endeca Server based on thesaurus matches. This topic describes the behavior of the thesaurus expansion feature.

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)
```

assuming 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 the section on this feature.

Step 5: Stemming

Query terms, unless they are delimited with quotation marks to be treated as exact phrases, are expanded by the Oracle Endeca Server 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 the section on this feature.

Step 6: Primitive term and phrase lookup

Primitive term and phrase lookup is the lowest level of search processing performed by the Oracle Endeca Server.

The Oracle Endeca 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 Oracle Endeca Server performs the "Did you mean" processing as part of the record search processing.

"Did you mean?" processing is analogous to the spelling correction and automatic phrasing processing, only that the results are not included, but rather the spelling suggestions and automatic phrases themselves are returned.

For details on the "Did you mean?" feature, see the section about it.

Step 8: Navigation filtering

The Oracle Endeca Server 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 the Oracle 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 Query Language Reference*.

Step 10: Relevance ranking

Relevance ranking is the last step in the Oracle Endeca Server 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 about it.

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 the Oracle 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, the Oracle 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 `-v` flag to the Dgraph causes the Dgraph process to output detailed information about its record search configuration. If you are unsure whether the Dgraph is recognizing a particular parameter, start it with the `-v` flag and check the output.

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 process.

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 18

Working with 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.

In Integrator, you can use the **WebClient** component to send a request to the Oracle Endeca Server using the Configuration Web Service. This request sends the `RECSEARCH_CONFIG` document to the Oracle Endeca

Server, thus creating a search interface. For information on how to configure a search interface using Integrator, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

If you are not using Integrator, you can create a request with the `putConfigDocuments` operation of the Configuration Web Service and send the `RECSEARCH_CONFIG` XML document to the Oracle Endeca Server. For information, see *Configuration Web Service operations on page 28*.

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 process of the Oracle Endeca Server 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. 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 process.

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.

Attribute	Purpose
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.
STRICT_PHRASE_MATCH	Specifies that the Dgraph process 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.

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.

Tips for troubleshooting search interfaces

All the tips for troubleshooting basic record search are also useful for troubleshooting record search that uses search interfaces. To get the most out of the search interfaces feature, make sure to set your search interfaces to contain the relevant searchable fields.

Chapter 19

Using Value Search

This chapter discusses how the Oracle 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 the Oracle 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 to the Oracle Endeca Server.

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. 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. 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.

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 GCR using the Configuration Web Service, or you can use Integrator.

For information on how to use the Configuration Web Service, see the section in this guide and the *Oracle Endeca Server API Reference* (which contains documentation for the WSDL).

For information on how to use Integrator to enable a standard attribute for value search, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

Utilizing value search in Studio

Value search supports the refinement search available in the **Guided Navigation** 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.

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.

`ValueSearchConfig` is a type of `ContentElementConfig` complex type element. `ValueSearchConfig` controls the behavior of a single value search query.

The `SearchTerm` element specifies search term(s) used by the Oracle 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**
`HandlerFunction`	Required attribute. Specifies the `ValueSearchHandler` handler function for `ValueSearchConfig`, which is a type of `ContentElementConfig`.
`HandlerNamespace`	Required attribute. Specifies the namespace for the handler function.
`Id`	Optional. An identifier for this query configuration.
`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.
`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 Oracle Endeca Server uses the value for the strategy provided in the `DIMSEARCH_CONFIG` XML configuration document. Further, if the document does not specify a strategy, the Oracle Endeca Server ranks the results using the following three strategies in this order (to break ties): `interp`, `exact`, and `static`.
`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) the Oracle Endeca Server uses to conduct value search.

Example of a value search query

The following example illustrates the format of a typical value search request in the Conversation Web Service. In this request, a search is conducted for the term `Wash` within the `ModelName` attribute. The number of requested results to return per attribute is set to 5:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
<soapenv:Header/>
<soapenv:Body>
<ns:Request>
 <ns:State/>
 <ns:Operator/>
```

```
  <ns:ContentElementConfig
   Id="ValueSearchConfig"
   xsi:type="ns:ValueSearchConfig"
   HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
   HandlerFunction="ValueSearchHandler"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   MaxPerProperty="5"
   RelevanceRankingStrategy="static (nbins,descending)"
   Mode="Any">
   <ns:SearchTerm>Wash</ns:SearchTerm>
   <ns:RestrictToProperties>
   <ns:Property>ModelName</ns:Property>
   </ns:RestrictToProperties>
   </ns:ContentElementConfig>
   <ns:PassThrough/>
  </ns:Request>
 </soapenv:Body>
</soapenv:Envelope>
```

Constructing a value search query

You create a value search query by issuing a request that uses the `ValueSearchConfig` type.

Use the parameters for `ValueSearchConfig` specified in its format.

As a rule of thumb, for any record attribute in the data store that could contain more than 100 possible results, use `<RestrictToProperties>` and `MaxPerProperty` attributes 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 the Oracle Endeca Server.

To create a value search query:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:Request>
      <ns:State>
      </ns:State>
      <ns:Operator/>
      <ns:ContentElementConfig Id="ValueSearchConfig"
        xsi:type="ns:ValueSearchConfig"
        HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
        HandlerFunction="ValueSearchHandler"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        MaxPerProperty="5"
        RelevanceRankingStrategy="static (nbins,descending)"
        Mode="Any">
        <ns:SearchTerm>26</ns:SearchTerm>
        <ns:RestrictToProperties>
          <ns:Property>ProductCategory</ns:Property>
          <ns:Property>Bike Racks</ns:Property>
        </ns:RestrictToProperties>
      </ns:ContentElementConfig>
      <ns:PassThrough />
    </ns:Request>
  </soapenv:Body>
</soapenv:Envelope>
```

The results of a value search query are returned in the `ValueSearch` type of `ContentElementConfig`. In the 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.

The `HasMore` attribute specifies whether any results are cut off because of a limit specified in the request.

Restricting value search to specific attributes

Value search queries could potentially contain many results. You can use `RestrictToProperties` attribute to limit the number of returned results to a list of one or more specified attributes.

If a managed attribute is searched, using `RestrictToProperties` you can 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` element inside `ContentElementConfig`, as shown in this abbreviated example:

```
<ns:ContentElementConfig
...
<ns:RestrictToProperties>
<ns:Property>ProductCategory</ns:Property>
<ns:Property>Bike Racks</ns:Property>
</ns:RestrictToProperties>
</ns:ContentElementConfig>
```

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` attribute 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:

```
<ns:ContentElementConfig
...
MaxPerProperty="2">
<ns:SearchTerm>Handlebars</ns:SearchTerm>
<ns:RestrictToProperties>
...
</ns:ContentElementConfig>
```

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:

`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`.

Additionally, if the returned match is a managed attribute, then in the response, you will see the `Match` complex type. `Match` lists details of a single value within a particular attribute that matched a value search. If the matched value belongs to a managed attribute, then the `FullPath` is present in the response too, as in the following abbreviated example of the response:

```
<cs:PropertyMatches Name="ProductCategory" DisplayName="Product Category" TotalValuesCount
="1" HasMore="false">
 <cs:Match>
  <cs:MatchingValue DisplayName="Gloves">20</cs:MatchingValue>
    <cs:FullPath>
      <cs:DimensionValue>
        <cs:DimensionValue DimensionName="ProductCategory"
         Spec="/">ProductCategory
        </cs:DimensionValue>
        <cs:Operator xsi:type="cs:RefinementOperator"
         Name="ProductCategory" Spec="/"/>
        </cs:DimensionValue>
                <cs:DimensionValue>
         <cs:DimensionValue DimensionName="ProductCategory"
          Spec="CAT_CLOTHING">Clothing
        </cs:DimensionValue>
        <cs:Operator xsi:type="cs:RefinementOperator"
         Name="ProductCategory" Spec="CAT_CLOTHING"/>
        </cs:DimensionValue>
        <cs:DimensionValue>
            <cs:DimensionValue DimensionName="ProductCategory"
              Spec="20">Gloves</cs:DimensionValue>
              <cs:Operator xsi:type="cs:RefinementOperator"
               Name="ProductCategory" Spec="20"/>
             </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 an ascending alphabetical order.

The ordering of values, within each attribute, uses an order described 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 process of the Endeca Server uses the value for this strategy provided in the DIMSEARCH_CONFIG XML configuration document (you can send an updated version of this document to the Endeca Server by using the Configuration Web Service).
- Further, if the document does not provide a strategy, the Endeca Server 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:

```
<ns:ContentElementConfig
...
RelevanceRankingStrategy="static (nbins,descending)">
...
</ns:ContentElementConfig>
```

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 need to 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 Oracle Endeca Server treats wildcards in value searches:

- A wildcard search at the end of the search term, such as `gua*`, is conducted by the Oracle Endeca Server for all standard attributes for which value search is enabled.
- Wildcard searches of type `*uara` and `g*ara` are conducted by the Oracle Endeca Server 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 Oracle 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 of 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 `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 20

Using Search Modes

By default, search operations built on top of the Oracle Endeca Server 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 chapter 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. The example in this topic explains the interaction between stop words and `Partial` mode.

The Oracle 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, the Oracle Endeca Server first uses `All` mode to return results matching all search terms, if any are available.

If no such `All` results are available, the Oracle 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, spell correction can also alter the results. A search that matches any or partial spell-corrected in a 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, the Oracle Endeca Server first uses `All` mode to return results matching all search terms, if any are available.

If no such `All` results are available, the Oracle 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 that is part of `RECSEARCH_CONFIG`. For information, see *Recsearch_config elements on page 227*.

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 for `Mode` attribute in the `SearchFilter` element of your request, and in the `ValueSearchConfig` element.

The following example uses the `AllPartial` search mode in `SearchFilter`:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:Request>
      <ns:State/>
      <ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
        xsi:type="ns:SearchOperator">
        <ns:SearchFilter Mode="AllPartial" Key="MySearchInterface">mountain</ns:SearchFilter>
      </ns:Operator>
      <ns:ContentElementConfig xsi:type="ns:RecordListConfig"
          Id="RecordList"
          HandlerFunction="RecordListHandler"
          HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
          MaxPages="2">
          <ns:Column>ProductCategory</ns:Column>
          <ns:RecordsPerPage>50</ns:RecordsPerPage>
          <ns:Page>15</ns:Page>
          <ns:Sort Key="ProductCategory" Direction="Ascending"/>
          </ns:ContentElementConfig>
    </ns:Request>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 21

Using Boolean Search

This chapter describes how to enable Boolean search for record search and attribute 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 store 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 run Boolean search queries against search interfaces also; 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:    ':' ;
```

Examples of using the key restrict operator

This topic uses examples to explain how to use the key restrict operator (`:`) in queries that contain Boolean search.

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 execute the desired result:

```
Actor: Deniro AND NOT Director: Pacino
```

This 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. For example, the query:

```
car maker : aston martin
```

will search for the word `car` against the specified search interface, the word `aston` against the attribute named `maker`, and `martin` against the specified search interface.

If the intention was to search against the attribute named "car maker", you must alter the query to one of the following:

-
  ```
  "car maker" : aston martin
  ```
  This query searches for the word `aston` against the attribute `car maker`, while it searches for `martin` against the specified search interface.

-
  ```
  "car maker" : (aston martin)
  ```
  This query does a conjunctive (All) search for the words `aston martin` against the attribute `car maker`.

-
  ```
  "car maker" : "aston martin"
  ```
  This query searches for the phrase `aston martin` against the attribute `car maker`.

About proximity search

The proximity operators, `NEAR` and `ONEAR`, let users 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

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" NEAR/8 Hartford
```

The following sentence:

```
"Tour the Hartford,
Connecticut home where Mark Twain lived and
 worked from 1874 to 1891."
```

would not be considered 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 expression:

```
(chardonnay OR merlot) NEAR/5 California
```

is not a valid expression.

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 query:

```
("car maker" : aston) NEAR/3 martin
```

is not valid.

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 either 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	
Range filters	Yes for the `AND` operator only.	
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.`	

Sample query	Error message	Comments
`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 managed attribute 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 types of 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 149*.

Boolean search in value search

In this example, Boolean search mode is used for a value search made with the `ValueSearchConfig` type:

```
<ns:ContentElementConfig
   Id="ValueSearchConfig"
   xsi:type="ns:ValueSearchConfig"
   HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
   HandlerFunction="ValueSearchHandler"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   Mode="Boolean">
  <ns:SearchTerm>"Bike Racks" AND "Handlebars"</ns:SearchTerm>
</ns:ContentElementConfig>
```

Boolean search in record filters

You can use strings that specify Boolean search as input for the `RecordFilter` operator:

```
<ns:Operator
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="ns:RecordFilterOperator">
   <ns:RecordFilter
     Name
="OmitSystemRecords">AND(NOT(mdex-config_Key),NOT(mdex-property_Key),NOT(mdex-dimension_Key))
```

```
   </ns:RecordFilter>
</ns:Operator>
```

Boolean search in attribute filters

When you set up the attributes for which to search using the `SearchFilter` operator, 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 `SearchFilter` that uses Boolean search:

```
<ns:SearchFilter Mode="Boolean" Key="Description">Mountain</ns:SearchFilter>
```

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 *on page ???*.

Now you can create a single Boolean search request against the defined search interfaces:

```
<ns:Operator
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="ns:SearchOperator" Within="false">
  <ns:SearchFilter Key="All" Mode="Boolean">English : one AND Spanish : dos</ns:SearchFilter>
</ns:Operator>
```

Troubleshooting Boolean search

If you encounter unexpected behavior while using Boolean search, use the Dgraph `-v` flag when starting the data store in the Oracle Endeca Server. This flag prints detailed output to standard error describing its execution of the Boolean query.

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 22

Using 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 Oracle 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, phase 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 the Oracle Endeca Server at all times. For phrase search query processing, the Oracle 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)

The Oracle 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 store.

How punctuation is handled 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:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:typ="http://www.endeca.com/MDEX/lql_parser/types">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:Request>
      <ns:State/>
      <ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0"
        xsi:type="ns:SearchOperator">
        <ns:SearchFilter Key="MySearchInterface">"Mountain Bikes"</ns:SearchFilter>
      </ns:Operator>
     </ns:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

- A value search for values containing the phrase `"Touring Bikes"` is similar to the following abbreviated example:

```
<ns:Request>
  <ns:State>
  </ns:State>
  <ns:Operator/>
  <ns:ContentElementConfig Id="ValueSearchConfig" xsi:type="ns:ValueSearchConfig"
      HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
      HandlerFunction="ValueSearchHandler"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      MaxPerProperty="5" RelevanceRankingStrategy="static (nbins,descending)"
      Mode="Any">
     <ns:SearchTerm>"Bike Racks"</ns:SearchTerm>
     <ns:RestrictToProperties>
       <ns:Property>ProductCategory</ns:Property>
     </ns:RestrictToProperties>
  </ns:ContentElementConfig>
</ns: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 the Oracle 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 23

Using 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 the Oracle 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 the Oracle Endeca Server via the `RECSEARCH_CONFIG` configuration document.

The Oracle 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 the Oracle Endeca Server using the Configuration Web Service or Integrator.

The `RECSEARCH_CONFIG` document allows inclusion of `SEARCH_INTERFACE`, which in turn lets you specify 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, snippet size is set to 10 for an attribute "Description".
2. Use the Configuration Web Service or Integrator to send the `RECSEARCH_CONFIG` document to the data store in the Oracle Endeca Server.
   For information on the Configuration Web Service, see the section in this guide, or the *Oracle Endeca API Reference*. For information on Integrator, see the *Oracle Endeca Information Discovery Integrator Components 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 the Oracle 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 `--snip_cutoff` flag to the Dgraph to restrict the number of words that the Oracle Endeca Server evaluates in an attribute. For example, `--snip_cutoff 300` evaluates the first 300 words of the attribute to identify the snippet.

If the `--snip_cutoff` Dgraph flag is not specified, 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 the topic for enabling snippeting for the detailed format of the configuration.

Request examples for retrieving snippets

To request snippets with the Conversation Web Service, use the `SearchFilter` with the specified search interface. The Conversation Web Service returns snippets as part of the `RecordListEntry` element (which also returns the records themselves).

Specifying the name of the search interface in the `SearchFilter` retrieves snippeting information:

```
 <ns:SearchFilter Key="My search interface">
</ns:SearchFilter>
```

where `My search interface` is the name of the search interface for which snippeting is enabled for its members in the `RECSEARCH_CONFIG` XML document.

Example request

The following request illustrates how to request a snippet with the Conversation Web Service for a search interface `Description`:

```
<ns:Request>
  <ns:State/>
  <ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0" xsi:type="ns:SearchOperator">
    <ns:SearchFilter Key="Description">gearing</ns:SearchFilter>
  </ns:Operator>
  <ns:ContentElementConfig xsi:type="ns:RecordListConfig"
     Id="RecordList"
     HandlerFunction="RecordListHandler"
     HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
     MaxPages="2">
    <ns:Column>Description</ns:Column>
    <ns:RecordsPerPage>50</ns:RecordsPerPage>
    <ns:Page>15</ns:Page>
    <ns:Sort Key="Description" Direction="Ascending"/>
  </ns:ContentElementConfig>
</ns:Request>
```

Example 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>
...
```

Enabling snippets per query with the API

You can enable snippets for a particular attribute on a per query basis using the `SearchFilter` element in the Conversation Web Service.

Setting the `EnableSnippeting` attribute to `true` in the `SearchFilter` enables snippeting per query, for the specified property. The `SnippetLength` attribute sets the length of the snippet; the search term specifies the snippet term:

```
<ns:Request>
  <ns:State/>
  <ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0" xsi:type="ns:SearchOperator">
    <ns:SearchFilter Key="Description" EnableSnippeting="true"
      SnippetLength="4">gearing</ns:SearchFilter>
  </ns:Operator>
</ns:Request>
```

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.

Chapter 24

Using Wildcard Search

Wildcard search allows users to match query terms to fragments of words in text processed by an Endeca data store.

About wildcard search

Wildcard search is the ability to match user query terms to fragments of words in indexed text.

Normally, Oracle 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*
```

which 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 the Oracle Endeca Server, you can use the Configuration Web Service directly or use Integrator.

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 Components Guide*.

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 the Oracle Endeca Server through a connector in Integrator.

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 it is an error if `mdex-property_TextSearchAllowsWildcards` is set to `true` but `mdex-property_IsTextSearchable` is set to `false`. In other words, an attribute must be record searchable in order for it to allow wildcard search in record searches.

Note: Enabling wildcard search for an attribute which 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/1/0"
     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.

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 Oracle Endeca Server data store. 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`.

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 to send this document to the Oracle Endeca Server. For information, see the section about the Configuration Web Service in this guide, or the *Oracle Endeca Information Discovery Integrator Components 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 `--wildcard_max` flag for the Dgraph.

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.

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 to 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 the Oracle 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 process of the Oracle Endeca Server performs wildcard indexing for words that are shorter than 1024 characters. Words that are longer than 1024 characters are not indexed for wildcard search.

- **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 process generates lists of words that match each of the punctuation-separated wildcard expressions. Only in this case, Dgraph uses the `--wildcard_max <count>` setting to optimize its 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.

Chapter 25

Search Characters

This chapter describes the semantics of matching search queries to result text.

About search characters

The Oracle 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 the Oracle Endeca Server.

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 the Oracle Endeca Server, use the Configuration Web Service or Integrator.

Query matching semantics

The semantics of matching search queries to text containing special non-alphanumeric characters in the Oracle 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

The Oracle Endeca Server treats characters in indexed text based on three categories:

- 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 Oracle 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

The way 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 the chapter "About phrase search.")
  - 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 does 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 process flags necessary to enable the search characters feature. The Oracle Endeca Server automatically detects the additional search characters.

The Oracle Endeca Server supports an important closely related feature: automatic mapping of ISO-Latin1 international characters to ASCII equivalents in text search queries. You can specify this mapping with the `--latin1` flag for the Dgraph. This option allows search queries containing international characters such as Spätlese to match against Anglicized result text such as Spatlese. Using the `--latin1` flag causes the Latin1 mappings to be applied to search queries.

Chapter 26

Working with Spelling Correction and Did You Mean

This chapter describes the behavior of the Spelling Correction and Did You Mean features.

About Spelling Correction and Did You Mean

The Oracle Endeca Server supports two complementary forms of Spelling Correction — automatic spelling correction for record search and value search, and explicit spelling suggestions for record search ("Did You Mean?").

The Automatic Spelling Correction and Did You Mean features of the Oracle Endeca Server enable search queries to return expected results when the spelling used in query terms does not match the spelling used in the result text (that is, when the user misspells search terms).

Either or both features can be used in a single application, and all 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, the Oracle 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. The Oracle 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*.

Enabling Spelling Correction and Did You Mean

To enable spelling correction and spelling suggestions, run the `admin?op=updateaspell` command on the Dgraph process.

Logic used for spelling correction

At a high level, the spelling engine in Oracle 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 process uses, 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*.

To change the Dgraph process configuration for Automatic Spelling Correction and DYM, you can rebuild the spelling dictionary with the `admin?op=updateaspell` command at any time. During the data ingest process, you can periodically run this command to update the spelling dictionary in the Dgraph data files.

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 Aspell indexing parameters such as minimum word occurrences, maximum and minimum word length. These parameters let you set the boundaries indicating to the Dgraph process of the Oracle Endeca Server which words should be included in the spelling dictionary.

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 results computed by spelling suggestion queries according to the number of records than these values would lead to if selected in a navigation query. This alternate definition of number or results allows consistent behavior between spelling corrections computed for value and record search operations when given the same query terms.

updateaspell

The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus while continuing to issue queries and updates to the Oracle Endeca Server and without stopping and restarting the Dgraph process.

Run this command after you have added data records to the data store, to enable spelling correction in the Dgraph process for this data store.

During the data ingest process, you can run the `admin?op=updateaspell`command periodically to update the spelling dictionary used by the Dgraph for Automatic Spelling Correction and DYM.

The `admin?op=updateaspell` 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.
- Compiles a temporary text version of the `aspell` word list, `<db_prefix>.worddat`.
- Converts this word list to the binary format required by `aspell`.
- Writes the generated binary file into the current data files representation in the Oracle Endeca Server, for a particular data store.
- Makes the updated `aspell` spelling dictionary available in the Dgraph for processing of all queries arriving after this index update. The Dgraph process uses this updated dictionary when processing all future queries.

  Note: Because of the nature of continuous query, once the Dgraph processes this administrative request, it will start using the updated spelling dictionary after a certain point in its processing, and all newly incoming queries will be answered against the updated spelling dictionary. However, it is not possible to identify after which particular partial update or after which query the Dgraph will start using the newly updated spelling dictionary.

The Dgraph applies the updated settings while continuing to run queries and without needing to restart.

Note: If `admin?op=updateaspell` is started within a transaction, it must reference a transaction ID, as in the following example:

```
admin?op=updateaspell&outerTransactionId=42
```

Only one `admin?op=updateaspell` operation can be processed at a time.

The `admin?op=updateaspell` operation returns output similar to the following in the Dgraph error log:

```
...
spellengine aspell ran successfully.
```

If you start the Dgraph with the `-v` flag, the output also contains a line similar to the following:

```
Time taken for updateaspell, including wait time on any
previous updateaspell, was 290.378174 ms.
```

Spelling mode (Aspell)

Spelling features of the Oracle 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 process 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

The Oracle 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 suggestions and DYM in query results

You can retrieve spelling suggestion and did you mean (DYM) information in a query using the `SearchAdjustmentConfig` type of the `ContentElementConfig` complex type of your Conversation Web Service request.

If spelling is enabled in the Oracle Endeca Server, and, in addition to breadcrumbs, you would like that the Conversation Web Service response contains supplemental information about spelling suggestions and DYM, a second `SearchAdjustmentConfig` type in `ContentElementConfig` is required. If it is included, spelling correction 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 additional `ContentElementConfig` with `SearchAdjustmentConfig` type is not included; however, while spelling correction takes place, the spelling correction and DYM suggestions are not returned in the response.

For example, the following abbreviated section of a query request contains `ContentElementConfig` with `SearchAdjustmentConfig` type, to ensure that spelling correction and DYM suggestions are returned in the response:

```
<ns:ContentElementConfig
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="ns:SearchAdjustmentConfig"
    HandlerFunction="SearchAdjustmentHandler"
    HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
    Id="SearchAdjustments"/>
```

The response would then be similar to the following. It contains suggested terms for DYM:

```
<cs:ContentElement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="cs:SearchAdjustments" Id="SearchAdjustments">
  <cs:SuggestedAdjustment RecordCountIfApplied="15">
    <cs:SearchFilter Key="Essay" Mode="All">jane</cs:SearchFilter>
    <cs:SuggestedTerms>can</cs:SuggestedTerms>
    <cs:Operator xsi:type="cs:ApplySpellingSuggestionOperator">
      <cs:SearchFilter Key="Essay" Mode="All">jane</cs:SearchFilter>
      <cs:Replacement>can</cs:Replacement>
    </cs:Operator>
  </cs:SuggestedAdjustment>
</cs:ContentElement>
```

Configuring constraints for spelling dictionaries

The Oracle Endeca Server selects words for the spelling dictionary based on predefined 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 the Oracle 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 process of the Oracle Endeca Server 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.

Attribute	Type	Description
`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.
`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 Integrator. For information, see either the section on Configuration Web Service in this guide, or, if you are using Integrator, see the *Oracle Endeca Information Discovery Integrator Components Guide*.
3. Run the `admin/<DataStore>?op=updateaspell` command on the data store 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 the Oracle Endeca Server to consider the alternate `Backstreet Boys`.

The following statements describe how word-break analysis works in the Dgraph process of the Oracle Endeca Server:

- It is enabled by default.
- As part of the word-break analysis, the Dgraph process 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, at which point 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 Oracle 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 prior to production application deployment.

Chapter 27

Using Stemming and Thesaurus

This chapter describes the tasks involved in implementing the Stemming and Thesaurus features.

Overview of stemming and thesaurus

The Oracle 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 in an Endeca data store by default.

The default configuration for stemming is recorded in the `en_word_forms_collection` configuration file. This file lists all word forms used for stemming dictionaries in an Endeca data store. This file is created in the Oracle Endeca Server data files once the data store is provisioned, and is typically not modified. However, you can overwrite the default English stemming word forms with those from another stemming file in another language, as explained below.

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 default language for the stemming feature is English. However, stemming files for these other languages are available:

- Dutch
- French
- German
- Italian
- Portuguese
- Spanish

The stemming files for these languages are shipped with Integrator. You use Integrator to overwrite the default stemming file with another one. For details, see the *Oracle Endeca Information Discovery Integrator Components Guide*.

Note: An Endeca data store supports only one stemming language at a time.

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.

The thesaurus feature is 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. As a rule, the Oracle Endeca Server will expand the user query into the maximum possible set of alternate queries based on the available thesaurus entries.

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, the Oracle Endeca Server returns hits for all of the queries: *red wine opener*, *vino rosso opener*, and *red corkscrew*.

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 the Oracle Endeca Server using the Configuration Web Service or Integrator, thus specifying the configuration your 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 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 Oracle Endeca Server configuration necessary to configure thesaurus and stemming information.

The Dgraph `--thesaurus_cutoff` flag can be used to tune performance associated with thesaurus expansion. By default, the value of this flag is set to 3, meaning that if a search query contains more terms that match thesaurus entries than the number set by this flag, none of the terms are thesaurus expanded.

Interactions with other search features

As core features of the Oracle 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.

The Oracle 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 the chapter in this guide.

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 Oracle Endeca Server's Dgraph process, 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 Oracle Endeca Server 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 28

Relevance Ranking

This chapter describes the tasks involved in implementing the Relevance Ranking feature.

About the relevance ranking feature

Relevance ranking lets you control the order in which search results are displayed to the end user of a front-end application powered by the Oracle 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.

About relevance ranking modules

Relevance ranking modules are the building blocks from which you build the relevance ranking strategies that you actually 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 the "Using Search Modes" chapter of this guide.

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 the "Using Search Modes" chapter in this guide.
2. Within the above strata, all single-field matches are ranked ahead of all cross-field matches. For more information, see the "Working with Search Interfaces" chapter in this guide.
3. Within the above strata, all non-spelling-corrected matches are ranked above all spelling-corrected matches. See the "Working with Spelling Correction and Did You Mean" chapter in this guide for more information.

4. Within the above strata, all thesaurus matches are ranked below all non-thesaurus matches. See the "Using Stemming and Thesaurus" chapter in this guide for more information.
5. Within the above strata, all stemming matches are ranked below all non-stemming matches. See the "Using Stemming and Thesaurus" chapter 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 the *Using Search Modes on page 156* section in this guide.

- 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 which 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.

Details on the three options are explained in the following topic.

Phrase module options

The Phrase module has a variety of options that you use to customize its behavior.

The Phrase module has three options, which 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.

Subphrase	Approximate	Expansion	Description
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. Look 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. Look only at the first possible phrase match within each record.
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. Extend subphrases to facilitate matching but rank based on the length of the original subphrase (before extension). 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. Look 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. Expand the query to facilitate matching but rank based on the length of the original subphrase (before extension). Look 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 can be specified as `nbins`, `depth`, or `rank`:

- Specifying `nbins` causes the static module to sort result values by the number of associated records in the full data set.
- Specifying `depth` causes the static module to sort result values by their depth in the managed attributes hierarchy.

- Specifying `rank` causes values to be sorted by the ranks assigned to them for the application.

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 the Oracle 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 the Oracle Endeca Server:

- You can configure relevance ranking to a search interface in the `RECSEARCH_CONFIG` configuration document, and send this document to the Oracle Endeca Server using the Configuration Web Service or Integrator.
- 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 the "Using standalone relevance ranking at the query level" topic.

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 store 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, thus specifying the configuration your 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.

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 like 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. For information about configuring relevance ranking in search interfaces, see the "Working with Search Interfaces" section in this guide.

For value search, the "Implementing relevance ranking for value search" topic in this section describes the configuration procedure.

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 the Oracle Endeca Server, using the Configuration Web Service or Integrator.

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 search and value search in query requests

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 `SearchFilter` element lets you specify a relevance ranking strategy for the query, as in this example:

```
<ns:Operator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns="http://www.endeca.com/MDEX/conversation/1/0" xsi:type="ns:SearchOperator"
Within="false">
<ns:SearchFilter Mode="AllPartial" RelevanceRankingStrategy="exact"
      Key="Description">Mountain</ns:SearchFilter>
</ns:Operator>
```

For more information on the `SearchFilter`, see *Record search operator on page 137*.

Value search

For value search, the `RelevanceRankingStrategy` attribute of the `ValueSearchConfig` type lets you specify a relevance ranking strategy for the query.

```
<ns:ContentElementConfig
Id="ValueSearchConfig"
xsi:type="ns:ValueSearchConfig"
HandlerNamespace="http://www.endeca.com/MDEX/conversation/1/0"
HandlerFunction="ValueSearchHandler"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
MaxPerProperty="5"
RelevanceRankingStrategy="static (nbins,descending)"
Mode="Any">
        <ns:SearchTerm>"Bike Racks"</ns:SearchTerm>
        <ns:RestrictToProperties>
                <ns:Property>ProductCategory</ns:Property>
        </ns:RestrictToProperties>
</ns:ContentElementConfig>
```

For more information on the `ValueSearchConfig` type, see *Value search query format on page 149*.

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 the "Recommended strategies" topic.

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 that 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 as well as the ones in your data. In this way you will see the full range of relevance ranking effects.

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 the "Working with Search Interfaces" chapter in this guide.

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 VI

References

Chapter 29

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

Format

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

Attributes

The COMMENT element has no attributes.

Sub-elements

The COMMENT element has no sub-elements.

Example

This example includes an informational comment.

```
<DIMSEARCH_CONFIG FILTER_FOR_ANCESTORS="FALSE"
```

```
  <COMMENT>Displays ancestor managed values.</COMMENT>
/DIMSEARCH_CONFIG>
```

DIMNAME

The DIMNAME element specifies the name of a managed attribute.

Format

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

Attributes

The DIMNAME element has no attributes.

Sub-elements

The DIMNAME element has no sub-elements.

Example

This example shows the name of a managed attribute.

```
<RECORD>
   <DIMNAME="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 search matching, spelling correction, filtering, and relevance ranking for value search. These options are configured in the file's `DIMSEARCH_CONFIG` root element.

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

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 the "Relevance Ranking" chapter of this guide.

RELRANK_APPROXPHRASE

The RELRANK_APPROXPHRASE element implements the Approximate Phrase relevance ranking module.

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

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

Format

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

Attributes

The RELRANK_APPROXPHRASE element has no attributes.

Sub-elements

The RELRANK_APPROXPHRASE element has no sub-elements.

RELRANK_EXACT

The RELRANK_EXACT element implements the Exact relevance ranking module.

This module groups results into strata based on how well they match a query string, with the highest stratum containing results that match the user's query exactly.

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 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 result, then the order of the sub-elements, which represent the strategies, is significant. The order of the sub-elements defines the order in which the strategies are applied to the search results.

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.
RELRANK_INTERP	A general-purpose strategy that assigns a score to each result document based on the query processing techniques used to obtain the match. Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching.
RELRANK_FREQ	Provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text.

Sub-element	Brief description
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 the "Working with Search Interfaces" chapter in this guide.

Format

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

Attributes

The MEMBER_NAME element has the following attributes.

RELEVANCE_RANK

RELEVANCE_RANK is an unsigned integer that specifies the relevance rank of a match on the specified Endeca standard or managed attribute. 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 "Using Snippeting in Record Searches" in this guide.

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 chapters in the "Search Features" part of 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 one.

MAX_WORDS_OMITTED

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

Sub-elements

The PARTIAL_MATCH element has no sub-elements.

Example

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

```
<SEARCH_INTERFACE CROSS_FIELD_BOUNDARY="ALWAYS"
      CROSS_FIELD_RELEVANCE_RANK="0"
      DEFAULT_RELRANK_STRATEGY="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 the "Using Stemming and Thesaurus" chapter in this guide.

THESAURUS

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

THESAURUS is the root element for all thesaurus entries.

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

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

Format

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

Attributes

The THESAURUS element has no attributes.

Sub-elements

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

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

Example

This example shows the thesaurus entries for an application.

```
<THESAURUS>
   <THESAURUS_ENTRY>
     <THESAURUS_FORM>france</THESAURUS_FORM>
     <THESAURUS_FORM>french</THESAURUS_FORM>
   </THESAURUS_ENTRY>
   <THESAURUS_ENTRY_ONEWAY>
     <THESAURUS_FORM_FROM>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_ENTRY	Indicates a set of word forms that are equivalent.

Example

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

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

THESAURUS_ENTRY_ONEWAY

A THESAURUS_ENTRY_ONEWAY element specifies a single-direction mapping.

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

Format

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

Attributes

The THESAURUS_ENTRY_ONEWAY element has no attributes.

Sub-elements

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

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

Example

In this example, searches for `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 30

Suggested Stop Words

Stop words are words that are set to be ignored by the Oracle Endeca Server.

About stop words

List of suggested stop words

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.

Stop words are counted in any search mode that calculates results based on number of matching terms. However, the Oracle 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 the Oracle Endeca Server with the Configuration Web Service, or by using Integrator.

a	do	me	when
about	find	not	where
above	for	or	why

an	from	over	with
and	have	show	you
any	how	the	your
are	I	under	
can	is	what	

Index