Endeca® MDEX Engine

Basic Development Guide

Version 6.2.1 • December 2011

Contents

Copyright and disclaimer

Preface

Endeca® InFront enables businesses to deliver targeted experiences for any customer, every time, in any channel. Utilizing all underlying product data and content, businesses are able to influence customer behavior regardless of where or how customers choose to engage — online, in-store, or on-the-go. And with integrated analytics and agile business-user tools, InFront solutions help businesses adapt to changing market needs, influence customer behavior across channels, and dynamically manage a relevant and targeted experience for every customer, every time.

InFront Workbench with Experience Manager provides a single, flexible platform to create, deliver, and manage content-rich, multichannel customer experiences. Experience Manager allows non-technical users to control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

At the core of InFront is the Endeca MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. InFront Integrator provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. InFront Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Endeca InFront, a customer experience management platform focused on delivering the most relevant, targeted, and optimized experience for every customer, at every step, across all customer touch points.

About this guide

This guide describes the basic tasks involved in developing an Endeca application.

It assumes that you have read the *Endeca Concepts Guide* and the *Endeca Getting Started Guide* and are familiar with the Endeca terminology and basic concepts.

Who should use this guide

This guide is intended for developers who are building applications using the Endeca Information Access Platform.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Endeca Customer Support

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

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

Part 1

Presentation API Basics

Chapter 1

Endeca Presentation API Overview

The Endeca Presentation API provides the interface to the Endeca MDEX Engine. You use the API to query the MDEX Engine and manipulate the query results.

List of Endeca APIs

Depending on the packages you installed, your Endeca installation may include one or more sets of Endeca APIs. This topic lists APIs and provides a brief overview of each API set.

The Endeca software packages contain the following API sets:

- The Endeca Presentation API. You can use this API to communicate with the MDEX Engine.

  Note: In addition to the Presentation API, the MDEX Engine also provides a Web service interface that is designed to communicate with standards-compliant Web service clients, using standard protocols and syntax such as HTTP and XML. For more information, see the *Web Services and XQuery Developer's Guide*. You can use both Presentation API and Web services features in the same application.

- The Logging API that is used by the Endeca Logging and Reporting System. For information, see the *Log Server and Report Generator Guide*.
- Security-related methods that are used to implement secure Endeca implementations. For information, see the *Security Guide*.

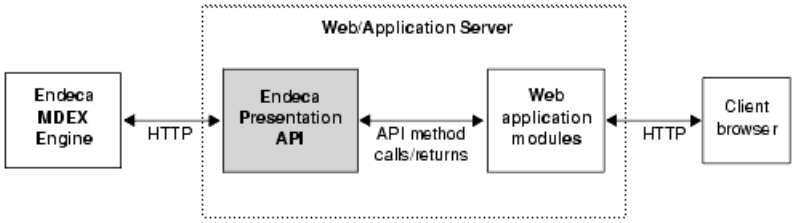
Architecture of the Presentation API

In a typical Endeca-based application that uses the Presentation API, the MDEX Engine communicates with the web application using the Presentation API.

The online portion of a typical Endeca implementation has the following components:

- The MDEX Engine, which receives and processes query requests.
- The Endeca Presentation API, which you use to query the MDEX Engine and manipulate the query results.
- A Web application in the form of a set of application modules, which receive client requests and pass them to the MDEX Engine through the Presentation API.

The following diagram illustrates the data flow between these components for a typical Endeca-based application that uses the Endeca Presentation API:



In this diagram, the following actions take place:

1. A client browser makes a request.
2. The Web application server receives the request and passes it to the application modules.
3. The application modules pass the request to the Endeca MDEX Engine, via the Presentation API.
4. The MDEX Engine executes the query and returns its results.
5. The application modules use Presentation API method calls to retrieve and manipulate the query results.
6. The application modules format the query results and return them to the client browser, via the Web application server.

Note: For security reasons, you should never allow Web browsers to connect directly to your MDEX Engine. Browsers should always connect to your application through an application server.

About Web application modules

The Web application modules are responsible for receiving client requests, and passing those requests to the MDEX Engine, via the Endeca Presentation API.

You build custom application modules for each Endeca application. This step is a key part of building an Endeca implementation. These modules can take many forms, depending on your application's requirements.

The Endeca distribution includes a set of sample UI reference implementations that you can refer to when building your own application modules.

Regardless of how you choose to build them, the application modules should perform the following functions:

- Receive requests from client browsers from the Web application server.
- Pass the client request to the MDEX Engine via the Endeca Presentation API.
- Retrieve the MDEX Engine query results via the Presentation API.
- Format the query results and return them to the client browser.

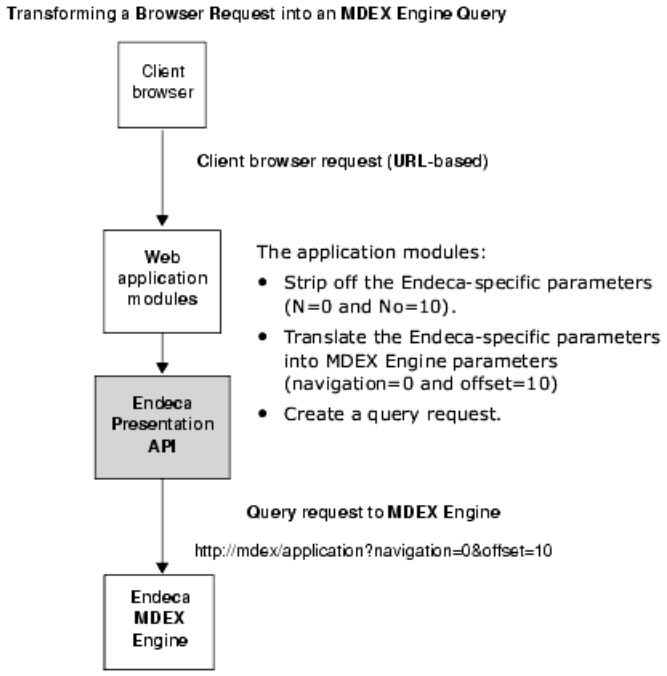
Methods for transforming requests into queries

A diagram in this topic illustrates how application modules transform a client browser request into an MDEX Engine query.

Before the Web application modules can send a client browser request to the MDEX Engine, the request must be transformed into an MDEX Engine query.

Typically, to make this transformation, the application modules extract the MDEX Engine-specific parameters from the original client request. In some cases, the modules may also edit the extracted parameters or add additional parameters, as necessary.

The following diagram illustrates the logic of transforming a client browser request into an MDEX Engine query:



Methods for passing request parameters

Several methods exist for passing the query request parameters from the client browser request to the application modules.

You can use one of the following methods:

- Embed parameters in the URL that the client browser sends.
- Send parameters in a cookie along with the client request.
- Include parameters in a server-side session object.

For example, in the UI reference implementations that are included with the Endeca Platform Services package, client request parameters are embedded directly in the URL. This method eases development and ensures load balancing, redundancy and statelessness.

Related Links

Creating the query with UrlENEQuery on page 24

You use the `UrlENEQuery` class to parse MDEX Engine-specific parameters from the browser request query string into MDEX Engine query parameters.

The Endeca Presentation API for Java and .NET

The Endeca Presentation API exists in the form of Java classes or .NET objects.

The Endeca Presentation API is managed by a Web application server of your choice. Depending on the environment you are working in, the Presentation API can take several different forms:

- For Java, the Presentation API is a collection of Java classes in a single `.jar` file.
- For .NET, the Presentation API is a set of .NET objects in a single assembly.

One query, one page

The data that the MDEX Engine returns in response to a query includes all of the information that the application modules would need to build an entire page for a typical application.

The MDEX Engine returns the following objects in response to a query request:

- Endeca records
- Follow-on query information
- Supplemental information, such as merchandising information, or information that enables the "Did You Mean" functionality

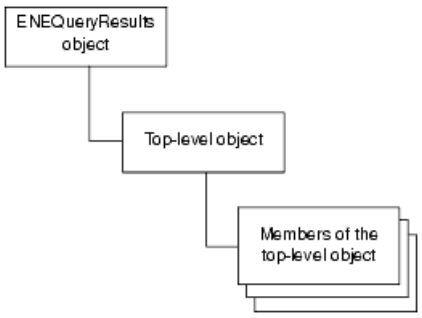
This enables the MDEX Engine to reduce the number of queries required to build an entire page, thereby improving performance. The performance improvement is gained by leveraging the processing for one section of a page to build the rest of the page.

For example, separate requests for record search information and navigation control information can be redundant. (Of course, you can make as many queries to the MDEX Engine as you want to build your pages, if the application design warrants it.)

About query result objects returned by the MDEX Engine

The MDEX Engine returns its results for all query types—navigation, record search, dimension search, and so on—in the form of a top-level object that is contained in an `ENEQueryResults` object. These top-level objects are complex objects that contain additional member objects.

The following diagram illustrates the relationship between an `ENEQueryResults`object, top-level object, and members of the top-level object:



Related Links

ENEQueryResults on page 26
An `ENEQueryResults` object contains the results returned by the MDEX Engine.

About top-level object types

The parameters in the MDEX Engine query determine the type of top-level object that is returned for the query.

You use Endeca Presentation API method calls to retrieve and manipulate data from a top-level object, and any of its members.

Top-level object types include the following:

- *Navigation objects* contain information about the user's current location in the dimension hierarchy, and the records that are associated with that location. Navigation objects also contain the information required to build any follow-on queries.

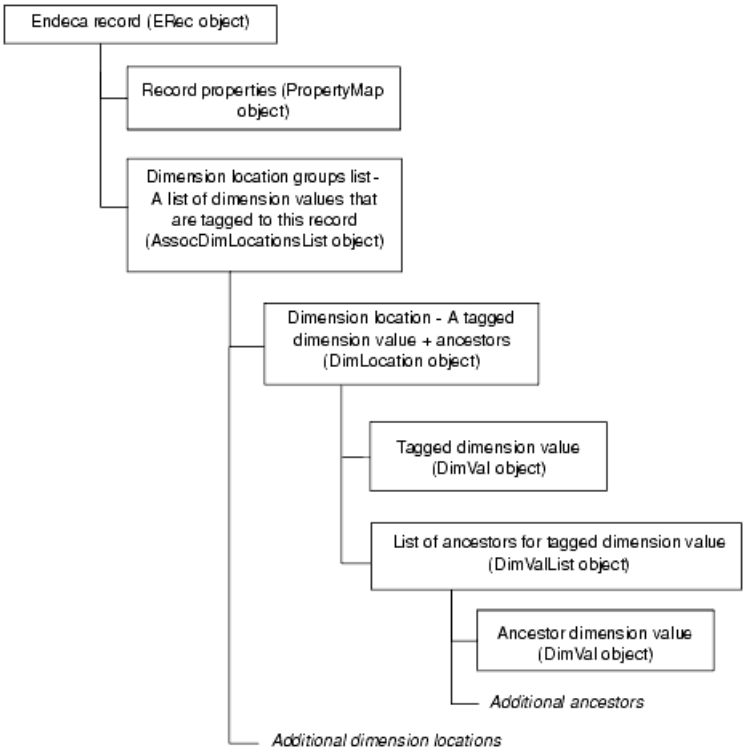
  Note: Both navigation queries and record search queries return Navigation objects.

- *Endeca record objects* contain full information about individual Endeca records in the data set. This information includes the record's Endeca properties, as well as its tagged dimension values.
- *Aggregated Endeca record objects* contain information about aggregated Endeca records. An aggregated Endeca record is a collection of individual records that have been rolled up based on a rollup key (an Endeca property or dimension name).
- *Dimension search objects* contain the results of a dimension search.

Example of a top-level object

To better understand an Endeca record object returned by the MDEX Engine, we can look at a diagram.

The following diagram shows the structure of a generic Endeca record object:



This diagram illustrates that Endeca record objects contain all the information associated with an Endeca record, including:

- A list of the dimensions that contain dimension values that have been tagged to the record.

- Information about each individual dimension, including:
  - Dimension root.
  - Tagged dimension value(s).
  - Ancestors for the tagged dimension value(s), if any exist.

Note: The combination of a tagged dimension value and its ancestors is called a dimension location.

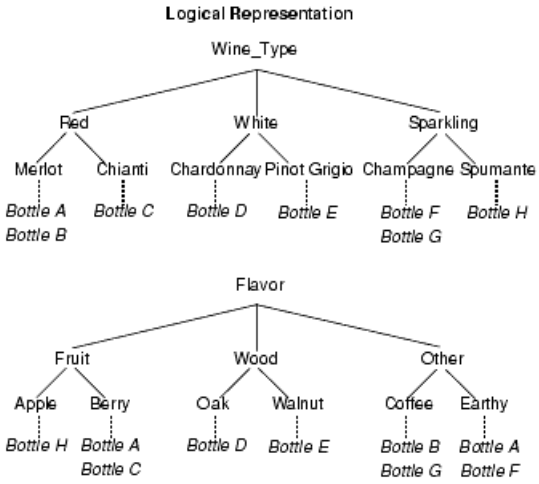
You can use the dimension hierarchy information in an Endeca record object to build follow-on navigation queries. For example, you can incorporate Find Similar functionality into your application by building a navigation query from the tagged dimension values for the current record.

Example of an Endeca record object for the wine data

To better understand an Endeca record object returned by the MDEX Engine, we can look at an example of an Endeca record object for Bottle A from a wine store.

In this example, our wine store data consists of two dimensions, one for Wine Type and another for Flavor.

The *logical* representation of the wine data can be presented as follows:



The *physical* representation of the wine data can be presented as follows:

Physical Representation

Bottle A
Wine_Type: Merlot
Flavor: Berry
Flavor: Earthy

Bottle B
Wine_Type: Merlot
Flavor: Coffee

Bottle C
Wine_Type: Chianti
Flavor: Berry

Bottle D
Wine_Type: Chardonnay
Flavor: Oak

Bottle E
Wine_Type: Pinot Grigio
Flavor: Walnut

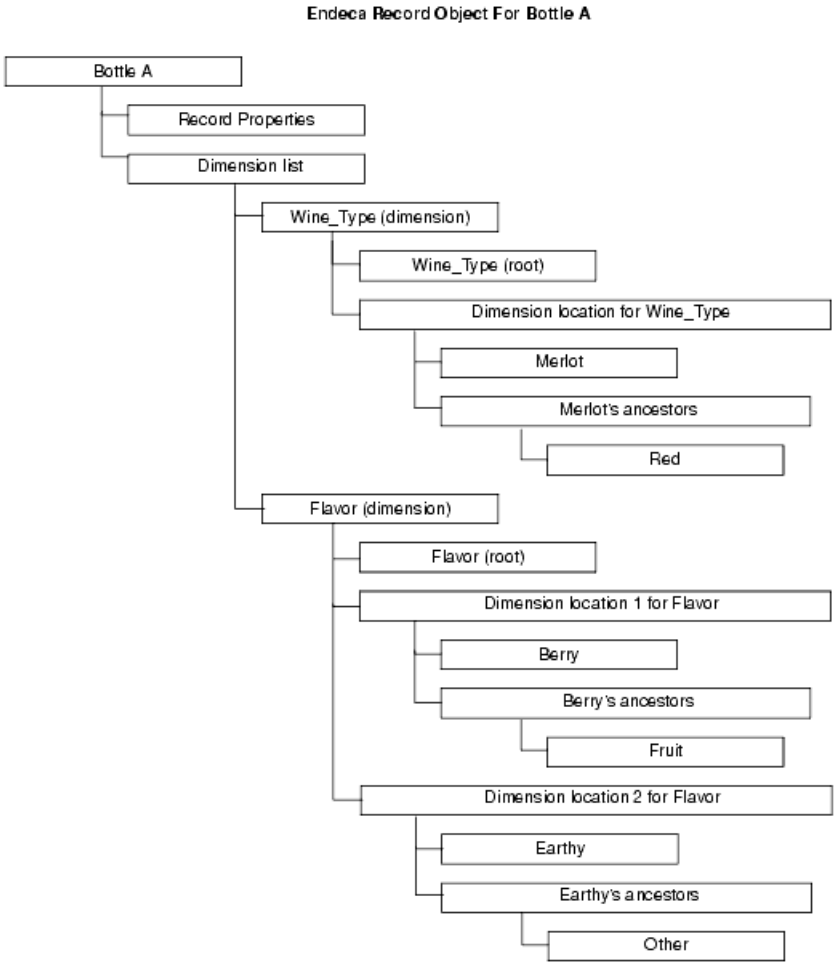
Bottle F
Wine_Type: Champagne
Flavor: Earthy

Bottle G
Wine_Type: Champagne
Flavor: Coffee

Bottle H
Wine_Type: Spumante
Flavor: Apple

In this example, you can see that Bottle A has been tagged with two dimension values from the Flavor dimension. This means that Bottle A has two dimension locations within the Flavor dimension.

The following illustration shows the Endeca record object for Bottle A:



Obtaining additional object information

Understanding the contents of Endeca's top-level objects is crucial to using and manipulating the MDEX Engine query results.

Refer to one of the following, depending on your platform, for detailed information on the top-level objects, and all of their members:

- Endeca API Javadocs
- Endeca API Guide for .NET

Chapter 2

Working with the Endeca Presentation API

This section provides information on working with the Endeca Presentation API classes.

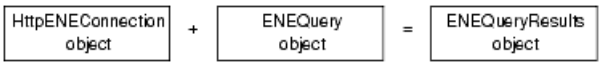
Core classes of the Presentation API

To query the MDEX Engine and access the resulting data, you use three core classes of the Endeca Presentation API together— `HttpENEConnection`, `ENEQuery`, and `ENEQueryResults`.

The Endeca Presentation API is based on three core classes:

- The `HttpENEConnection` class enables connections with the MDEX Engine.
- The `ENEQuery` class builds the query to be sent to the MDEX Engine.
- The `ENEQueryResults` class contains the results of the MDEX Engine query.

This diagram illustrates the relationship between three core classes:



HttpENEConnection

The `HttpENEConnection` class functions as a repository for the hostname and port configuration for the MDEX Engine you want to query.

The signature for an `HttpENEConnection` constructor looks like this:

```
//Create an ENEConnection
ENEConnection nec = new HttpENEConnection(eneHost, enePort);
```

`HttpENEConnection` is one of two implementations of the `ENEConnection` interface for Java and `IENEConnection` for .NET. This interface defines a `query()` method in Java, and a `Query()` method in .NET for all implementing classes.

Note: The other implementation of this interface is `AuthHttpENEConnection`.

In Java, you call the `query()` method on an `ENEConnection` object to establish a connection with an MDEX Engine and send it a query.

In .NET, you call the `Query()` method on an `HttpENEConnection` object to establish a connection with an MDEX Engine and send it a query.

Note: The instantiation of an `HttpENEConnection` object does not open a persistent connection to the MDEX Engine, nor does it initiate an HTTP socket connection. Instead, each issuance of the `HttpENEConnection` object's `query()` method in Java or `Query()` method in .NET opens an HTTP socket connection. This connection is closed after the query results have been returned. For some queries, multiple connections are opened for multiple MDEX Engine requests.

Changing the timeout setting for HttpENEConnection

If a connection to the MDEX Engine experiences a timeout, the default timeout period is 90 seconds. You can change the timeout setting for the `HttpWebRequest` objects (used by `HttpENEConnection`) to return.

By default, it takes 90 seconds for the `HttpWebRequest` objects (used by `HttpENEConnection`) to return, after an MDEX Engine connection timeout.

To change this default timeout for all `HttpWebRequest` objects inside `web.config`:

Modify the `httpRuntime` section as shown in the following example:

```
<system.web>
  <httpRuntime executionTimeout="00:00:30"/>
</system.web>
```

This change sets up a timeout of 30 seconds for a query request to time out.

ENEQuery and UrlENEQuery

You use the `ENEQuery` class, or its subclass `UrlENEQuery`, to create an MDEX Engine query.

Creating the query with UrlENEQuery

You use the `UrlENEQuery` class to parse MDEX Engine-specific parameters from the browser request query string into MDEX Engine query parameters.

The code to accomplish this task looks like the following:

- Java:

```
//Create a query from the browser request query string
 ENEQuery nequery = new UrlENEQuery(request.getQueryString(), "UTF-8");
```

  The browser request query string resides in the `HTTPServletRequest` object from the `javax.servlet.http` package.

- .NET:

```
//Create a query from the browser request query string
 ENEQuery nequery = new UrlENEQuery(Request.QueryString.ToString(), "UTF-
8");
```

Note: The browser request query string resides in the `HttpRequest` object from the `System.Web` namespace in ASP.NET. ASP .NET exposes the `HttpRequest` object as the intrinsic request object.

The `UrlENEQuery` class ignores non-MDEX Engine-specific parameters, so this class is still safe to use when additional application-specific parameters are needed (as long as they don't conflict with the MDEX Engine URL parameter namespace).

Creating an empty ENEQuery object and populating it

Alternatively, you can use the `ENEQuery` class to instantiate an empty `ENEQuery` object, and then populate it with MDEX Engine query parameters using a variety of `setter` methods in Java, or `ENEQuery` properties in .NET.

The code to accomplish this task is similar to the example below:

- Java:

```
//Create an empty ENEQuery object and populate it using setter methods
ENEQuery nequery = new ENEQuery();
nequery.setNavDescriptors(dimensionValueIDs);
nequery.setERec(recordID);
...
```

- .NET:

```
//Create an empty ENEQuery object and populate it using properties
ENEQuery nequery = new ENEQuery();
nequery.NavDescriptors = dimensionValueIDs
nequery.ERec = recordID
...
```

Creating MDEX Engine queries from state information

You can use the `ENEQuery` class to construct a query from any source of state information, including non-Endeca URL parameters, cookies, server-side session objects, and so forth. These are all application design decisions and have no impact on the final MDEX Engine query or its results.

The following are all valid ways of creating an MDEX Engine query:

- Java:

```
ENEQuery nequery = new UrlENEQuery("N=123", "UTF-8");

ENEQuery nequery = new ENEQuery();
DimValIdList descriptors = new DimValIdList("123");
nequery.setNavDescriptors(descriptors);

ENEQuery nequery = new ENEQuery();
DimValIdList descriptors =
 new DimValIdList((String)session.getAttribute("<variableName>");
nequery.setNavDescriptors(descriptors);

ENEQuery nequery = new ENEQuery();
DimValIdList descriptors = new DimValIdList(request.getParameter("N"));
nequery.setNavDescriptors(descriptors);
```

- .NET:

```
ENEQuery nequery = new UrlENEQuery("N=123", "UTF-8");

ENEQuery nequery = new ENEQuery();
DimValIdList descriptors = new DimValIdList("123");
nequery.NavDescriptors = descriptors;

ENEQuery nequery = new ENEQuery();
DimValIdList descriptors = new DimValIdList(Request.QueryString["N"]);
nequery.NavDescriptors = descriptors;
```

Executing MDEX Engine queries

The `ENEConnection` `query()` method in Java, and the `HttpENEConnection` `Query()` method in .NET use an `ENEQuery` object as its argument when they query the MDEX Engine.

The code to execute an MDEX Engine query looks like this:

Java Example

```
//Execute the MDEX Engine query
ENEQueryResults qr = eneConnectionObject.query(eneQueryObject);
```

.NET Example

```
//Execute the Navigation Engine query
ENEQueryResults qr = eneConnectionObject.Query(eneQueryObject);
```

ENEQueryResults

An `ENEQueryResults` object contains the results returned by the MDEX Engine.

An `ENEQueryResults` object can contain any type of object returned by the MDEX Engine. The type of object that is returned corresponds to the type of query that was sent to the MDEX Engine. See "Four basic queries" for more information.

Related Links

Four basic queries on page 28
While the queries you send to an Endeca MDEX Engine can become quite complex, there are four basic queries that you should be familiar with.

Using the core objects to query the MDEX Engine

To build an MDEX Engine query and execute it, you use the three core classes of the Endeca Presentation API. Code examples in this topic show you how to build and execute a query.

The code to build and execute a query would look similar to the following:

Java Example

```
//Create an ENEConnection
ENEConnection nec = new HttpENEConnection(eneHost, enePort);

//Create a query from the browser request query string
ENEQuery nequery = new UrlENEQuery(request.getQueryString(),
 "UTF-8");

//Execute the MDEX Engine query
ENEQueryResults results = nec.query(nequery);

//Additional Presentation API calls to retrieve query results
...
```

.NET Example

```
//Create an ENEConnection
HttpENEConnection nec = new HttpENEConnection(eneHost, enePort);

//Create a query from the browser request query string
ENEQuery nequery = new
UrlENEQuery(Request.QueryString.ToString(), "UTF-8");

//Execute the Navigation Engine query
ENEQueryResults results = nec.Query(nequery);

//Additional Presentation API calls to retrieve query results
...
```

List of query exceptions

The ENEConnection query() method in Java and the HttpENEConnection Query() method in .NET throw an exception if they encounter an error while attempting to query the MDEX Engine.

The following table describes the exceptions that can be thrown:

Exception	Description
`ENEException`	Indicates an exception from the MDEX Engine. This means that `ENEConnection` was able to contact the MDEX Engine but the MDEX Engine responded with an error.
`ENEAuthenticationException`	Indicates an authentication exception from the MDEX Engine. This means that `ENEConnection` was able to contact the MDEX Engine but the MDEX Engine responded with an authentication error.
`ENEQueryException`	Indicates any connection problems in this method.
`ENEConnectionException`	Indicates a communication error in the `ENEConnection` with the MDEX Engine.
`EmptyENEQueryException`	Indicates that the `query()` method in Java, and the `Query()` method in .NET were called using an empty `ENEQuery` object. This exception occurs because the `ENEQuery` object did not express any requests to the MDEX Engine.
`PartialENEQueryException`	Indicates that the `ENEQuery` object does not contain all the necessary query parameters.

Exception	Description
`UrlENEQueryParseException`	Indicates an error while parsing a browser request query string into individual MDEX Engine query parameters.
`VersionMismatchException`	Indicates the presence of incompatible modules in the Endeca application (discovered while attempting to process a query). Most often this exception signals a version mismatch between the Presentation API and the MDEX Engine itself.

Four basic queries

While the queries you send to an Endeca MDEX Engine can become quite complex, there are four basic queries that you should be familiar with.

These queries, and the type of objects they return, are listed below. Keep in mind that all of the returned objects are contained in the `ENEQueryResults` object:

Basic query	Returned object (type)
Navigation query	`Navigation`
Endeca record query	`ERec`
Dimension search query	`DimensionSearchResult`
Aggregated Endeca record query	`AggrERec`

You create the four basic queries using both `UrlENEQuery` and `ENEQuery` classes.

Building a basic query with the UrlENEQuery class

In order to create an MDEX Engine query based on a client browser request, the request URL must contain MDEX Engine-specific query parameters. While the number of parameters that the `UrlENEQuery` class can interpret is large, only a few of these parameters are required for the four basic queries.

The parameters that the `UrlENEQuery` class needs for the four basic queries are listed in this table:

Note: `Controller.jsp` or `Controller.aspx` in the examples below refer to the point of entry into the UI reference implementation.

Basic query type	URL param	Parameter definition	URL query string example
Navigation	`N`	The IDs of the dimension values to be used for a navigation query, or `N=0` for the root navigation request.	Java: `controller.jsp?N=0 con¬ troller.jsp?N=123+456` .NET: `controller.aspx?N=0 con¬ troller.aspx?N=123+456`
Endeca record	`R`	The specifier (string-based ID) of the Endeca record to be returned.	Java: `controller.jsp?R=12345` .NET: `controller.aspx?R=12345`
Dimension search	`D`	The dimension search terms.	Java: `controller.jsp?D=red+wine` .NET: `controller.aspx?D=red+wine`
Aggregated Endeca record	A,An ,Au	A: The specifier (string-based ID) of the aggregated Endeca record to be returned. An: The navigation descriptors that describe the record set from which the aggregated record is created. Au: The rollup key used to create the aggregated Endeca record.	Java: `con¬ troller.jsp?A=123&An=456+789&Au=Name` .NET: `controller.aspx?A=123& An=456+789&Au=Name`

You can combine the four basic queries in one URL, with the restriction that each type of query can appear only once per URL. Each basic query, however, has no impact on the other queries. Combining queries in the URL is used exclusively for performance improvement because it reduces the number of independent queries that are queued up waiting for the MDEX Engine.

Building a basic query with the ENEQuery class

To create a query manually, you instantiate an empty `ENEQuery` object and then use the `ENEQuery setter` methods (Java), or properties (.NET) to specify query parameters.

The number of `setter` methods (Java), or properties (.NET) available is large, but only a few are required to create a basic query with `ENEQuery`.

The methods and properties required for `ENEQuery`are listed in the table below:

Basic query type	Required methods (Java) or properties (.NET)
Navigation	Java: `setNavDescriptors(DimValIdList descriptors)` .NET: `NavDescriptors`
Endeca record	Java: `setERecSpec(String recordSpec)`

Basic query type	Required methods (Java) or properties (.NET)
	.NET: `ERecSpec`  **Note:** A `recordSpec`, or record specifier, is a string-based identifier.
Dimension search	Java: `setDimSearchTerms(String terms)` .NET: `DimSearchTerms`
Aggregated Endeca record	Java: `setAggrERecSpec(String aggregatedRecordSpec)`, `setAggrERecNavDescriptors(DimValIdList descriptors)`, `setAggrERecRollupKey(String key)` .NET: `AggrERecSpec`, `AggrERecNavDescriptors`, `AggrERecRollupKey`

ENEQuery naming convention

Each `ENEQuery` `setter` and `getter` method in Java, and property in .NET follow a naming convention that provides a quick way to determine the type of results the `ENEQuery` object will yield.

For example, `setNavRecordFilter()` in Java and `NavRecordFilter` in .NET are modifiers for a navigation request, and navigation requests return `Navigation` objects.

The table describes methods an properties, their corresponding returned object types and examples of usage in Java and .NET.

 **Note:** See the *Endeca API Javadocs* and *Endeca API Guide for .NET* for complete information on all Presentation API classes, method (Java), and properties (.NET).

Method (Java) or property (.NET) convention	Returned object (type)	Examples
Java: `setERec...()` .NET: `ERec...`	`ERec`	Java: `setERecs()`, `setERecSpec()` .NET: `ERecs`, `ERecSpec`
Java: `setNav...()` .NET: `Nav...`	`Navigation`	Java: `setNavNumERecs()` .NET: `NavNumERecs`
Java: `setDimSearch...()` .NET: `DimSearch...`	`DimensionSearchResult`  **Note:** This object has been deprecated.	Java: `setDimSearchTerms()` .NET: `DimSearchTerms`

Method (Java) or property (.NET) convention	Returned object (type)	Examples
Java: `setAggrERec...()` .NET: `AggrERec...`	`AggrERec`	Java: `setAggrERecRollupKey()` .NET: `AggrERecRollupKey`

Methods of accessing data in basic query results

To access data in query results, you can use `ENEQueryResults` methods in Java and properties in .NET.

There is a distinct correlation between the MDEX Engine parameters passed in the URL (or the `setter` methods (Java) and `ENEQuery` properties (.NET) used), and the methods or properties you can use to access data in the `ENEQueryResults` object.

For example, by including an `N` parameter in your query, a `Navigation` object is returned as part of the `ENEQueryResults`, and you use the `getNavigation()` method in Java on the `ENEQueryResults` object, or the `ENEQueryResults` object's `Navigation` property in .NET to access that `Navigation` object.

If you used this to create your query:	You can use these ENEQueryResults methods or properties:
`N` or Java: `setNavDescriptors()` .NET: `NavDescriptors`	Java: `getNavigation()` .NET: `Navigation`
`R` or Java: `setERecSpec()` .NET: `ERecSpec`	Java: `getERecSpec()` .NET: `ERecSpec`
`D` or Java: `setDimSearchTerms()` .NET: `DimSearchTerms`	Java: `getDimensionSearch()` .NET: `DimensionSearch`

`A`, `An`, `Au` or
Java:
`setAggrERecSpec()`
`setAggrERecNavDescriptors()`
`setAggrERecRollupKey()`
.NET:
`AggrERecSpec`
	Java: `getAggrERecSpec()`
	.NET: `AggrERecSpec`

If you used this to create your query:	You can use these ENEQueryResults methods or properties:
`AggrERecNavDescriptors` `AggrERecRollupKey`	

Methods of determining types of queries passed to the MDEX Engine

To determine what type of query is being passed or has been passed to the MDEX Engine, you can use `contains` methods on both the `ENEQuery` and `ENEQueryResults` objects.

If these methods evaluate to true:	Your query uses:

Java:
`ENEQuery` object: `containsNavQuery()`
`ENEQueryResults` object: `containsNav¬igation()`
.NET:
`ENEQuery` object: `containsNavQuery()`
`ENEQueryResults` object: `ENEQueryRe¬sults object:ContainsNaviga¬tion()`
	N or
	Java: `setNavDescriptors()`
	.NET: `NavDescriptors`

Java:
`ENEQuery` object: `containsERec¬Query()`
`ENEQueryResults` object: `contain¬sERec()`
.NET:
`ENEQuery` object: `ContainsERec¬Query()`
`ENEQueryResults` object: `Contain¬sERec()`
	R or
	Java: `setERecSpec()`
	.NET: `ERecSpec`

Java:
`ENEQuery` object: `ENEQuery ob¬ject:containsDimSearchQuery()`
`ENEQueryResults` object: `ENEQueryRe¬sults object:containsDimension¬Search()`
.NET:
	D or
	Java: `setDimSearchTerms()`
	.NET: `DimSearchTerms`

If these methods evaluate to true:	Your query uses:
ENEQuery object: ENEQuery ob¬ject:ContainsDimSearchQuery() ENEQueryResults object: ENEQueryRe¬sults object:ContainsDimension¬Search()	
Java: ENEQuery object: containsAggrERec¬Query() ENEQueryResults object: containsAg¬grERec() .NET: ENEQuery object: ContainsAggrERec¬Query() ENEQueryResults object: ContainsAg¬grERec()	A, An, Au or Java: setAggrERecSpec() setAggrERecNavDescriptors() setAggrERecRollupKey() .NET: AggrERecSpec AggrERecNavDescriptors AggrERecRollupKey

Getting started with your own Web application

Now that you have a deeper understanding of the Endeca Presentation API, you can begin building your own Endeca application. This topic gives you some pointers on how to approach building your first application.

This section refers to the UI reference implementation, which is a sample Web application included with the Endeca Platform Services package.

To start building your own application:

1. Define your architecture.

   Without relying on the UI reference implementation, define what your application's architecture requirements are.

   In Java, if you need to create JavaBeans or command classes, have a good definition of those requirements independent of the current structure and architecture of the reference implementation.

2. Determine your page and page element definitions.

   Again, this should be done without relying on the UI reference implementation. Most applications have a navigation page and a record page, but each application has its own requirements. A typical navigation page includes some sort of results section and query controls section, but this is also entirely dependent on the application design. Whatever the resulting design is, produce a list of all required elements and the pages they are associated with.

3. Evaluate each page element and decide which UI reference implementation module, if any, is the closest match to the functionality required.
   For example, if you have a dimension search results section, the misc_dimsearch_results module may be a good starting point. Keep in mind that the UI reference implementation does not use all of the Presentation API objects. You may need a component that has no closely

corresponding reference module. In this case, you need to develop this component from scratch or based on significant adjustments to an existing module. See the appropriate Endeca API Guide for complete information on the Presentation API.

4. Create a new application framework (that is, an "empty" application) and begin building each required element.

   Refer to the corresponding UI reference implementation modules as necessary. If a new element is very similar to an existing module, you may be able to start from that module's framework and simply add supporting HTML. If the new element is significantly different, however, you may want to use the existing module as a guide only and construct the new code from scratch.

Related Links

Using the UI Reference Implementation on page 35

This section describes the UI reference implementation, its components, and information you should know when using it.

Chapter 3

Using the UI Reference Implementation

This section describes the UI reference implementation, its components, and information you should know when using it.

UI reference implementation overview

The Endeca distribution includes a UI reference implementation that provides skeleton examples of typical navigation, record, and aggregated record pages and the components that make up these pages.

The UI reference implementation provides examples of modules, such as navigation controls, navigation descriptors, and a record set. It is intended as a guide for creating MDEX Engine queries and building pages from the query results. As such, you should feel free to use modules that are appropriate for your application's requirements and ignore those that aren't.

Each UI reference implementation module has a banner with the module name located prominently at the top.

In Java, the banner is orange.

In .NET, the banner is red.

All modules that have dependencies are named in such a way as to indicate the dependency. For example, the `nav_records_header` module is dependent on the `nav_records` module, which is dependent on the `nav` module.

Dependencies exist only between modules that have a parent-child relationship. Modules that have no parent-child relationship have no dependencies on each other and you can remove or modify them independently of each other. See "Module maps" for a visual representation of the parent-child dependencies.

Related Links

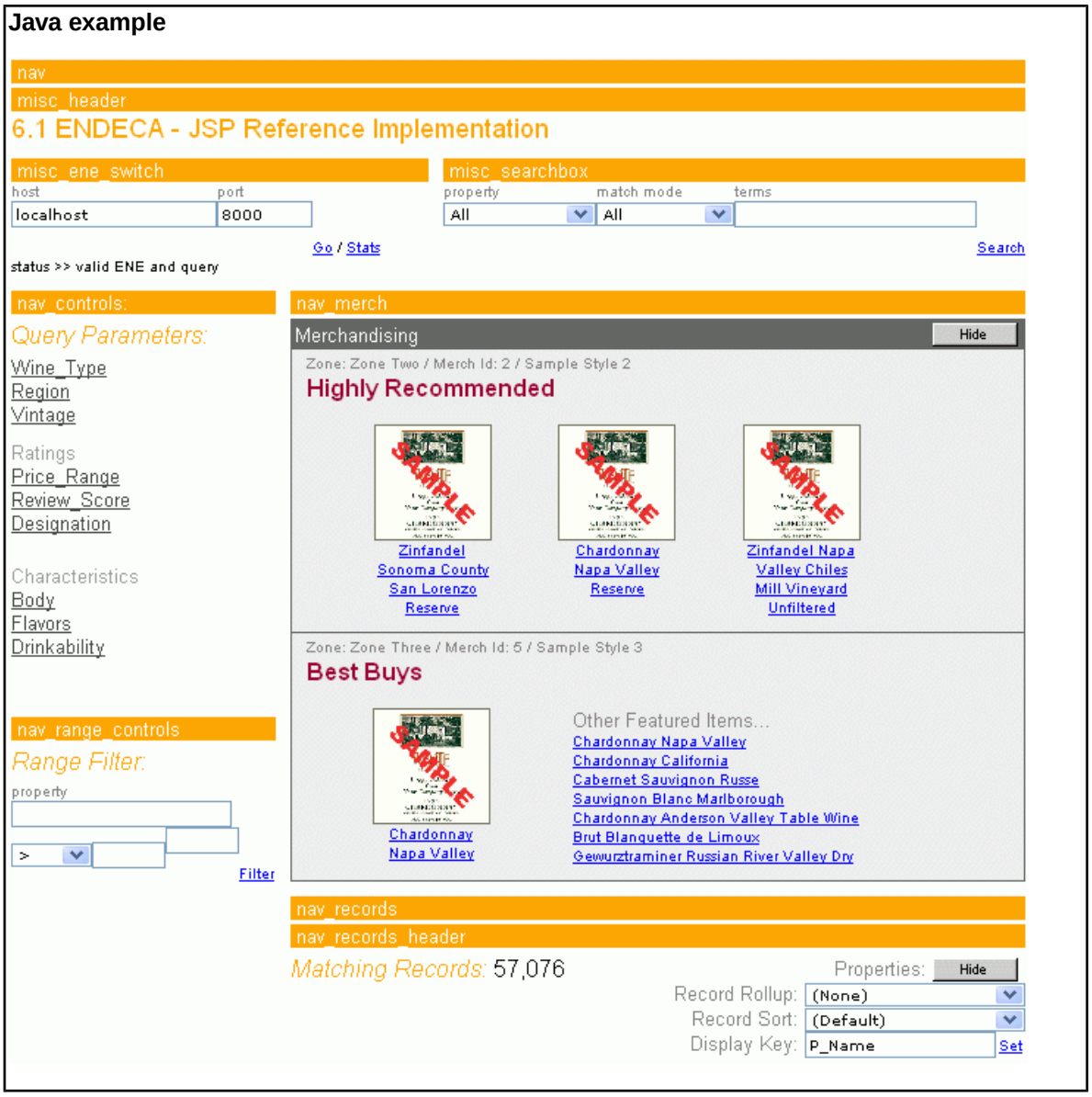
Module maps on page 39

The following diagrams show the relationship between the various UI reference implementation modules. The diagrams are broken into the four primary modules for Java and .NET.

The UI reference implementation screenshots

The diagrams in this topic show the UI reference implementation's primary page.

Java example



.NET example

The purpose of the UI reference implementation

In order to use the UI reference implementation appropriately, it is important to understand what the reference implementation is and is not.

The UI reference implementation is:

- A good code base for copying snippets of Presentation API calls.
- An excellent data inspection and data debugging application.
- A good template from which to build a rapid Endeca prototype.

The Java version

The Java version of the UI reference implementation is not:

- A good web application architecture example.
- A good place for copying snippets of HTML.

The UI reference implementation is built using a significantly different architecture than that you would use for a production-ready implementation. It does not use Java beans or classes, it has a heavy amount of in-line Java, and a relatively small amount of HTML. We chose this architecture in an effort to help you better visualize the `ENEQueryResults` object and its nested member objects. By merging in the Java code normally reserved for classes and using a small amount of HTML in each module, we hoped to create a streamlined, easier-to-read example of how the `ENEQueryResults` object is manipulated.

The .NET version

The .NET version of the UI reference implementation is not:

- A good web application architecture example.
- A good place for copying snippets of HTML.

The .NET version of the UI reference implementation is built using the ASP .NET architecture.

Four primary modules

The UI reference implementation has four primary modules.

These modules are:

- `controller`
- `nav`
- `rec`
- `agg_rec`

The `controller` module

The `controller.jsp` (Java) and `controller.aspx`(.NET) module is the entry point into the UI reference implementation. It receives the browser request from the application server, formulates the MDEX Engine query, establishes a connection with the MDEX Engine and sends the query. Based on the contents of the query results, the `controller` module determines whether the request was a navigation, a record, or an aggregated record request. For navigation requests, `controller` forwards the request to the `nav` module.

The `nav` module

The `nav.jsp` (Java) and `nav.aspx` (.NET) module, using other included `nav` modules, renders the main navigation page, including the navigation controls, navigation descriptors, and a record set.

The `rec` module

For record requests, `controller` forwards the request to the `rec.jsp` (Java) and `rec.aspx` (.NET) module which, along with its child `rec_*` modules, is responsible for rendering a record page for a single record.

The agg_rec module

For aggregated record requests, `controller` forwards the request to the `agg_rec.jsp` (Java) and `agg_rec.aspx` (.NET) module which, again, along with its child `agg_rec_*` modules, renders a page for an aggregated Endeca record.

About JavaScript files

The UI reference implementation includes several JavaScript files to support modules that use forms.

These JavaScript files contain functions that combine the URL from the current browser request with form data to create the new browser requests. The JavaScript was written to avoid the use of complicated forms that use hidden elements to maintain the MDEX Engine parameters from the current browser request.

The two modules that use JavaScript are:

- `Java: misc_ene_switch.jsp`

  .NET:`misc_ene_switch.aspx`

- Java: `misc_searchbox.jsp`

  .NET: `misc_searchbox.aspx`

The JavaScript files that support these modules are `misc_ene_switch.js` and `misc_searchbox.js`, respectively.

In addition, both JavaScript files use standard functions contained in a utility JavaScript file called `util.js`.

The use of JavaScript is completely optional. Using the `ENEQuery` alternatives, you can create a form-posting solution that avoids the use of JavaScript altogether. You must remember, however, that if you create your query using one of these alternatives, you are potentially left in a state where the browser request URL no longer reflects the `ENEQuery`. In this instance, the JavaScript returned with the page will not be useful, because it references a browser request that has since been modified. Given this caveat, Endeca recommends that you only use the JavaScript files when:

- You use the `UrlENEQuery` class to build your query.
- You use redirect calls in the `controller` module to redirect the modified request back to the `controller` module using the new parameters. See comments in the `controller.jsp` (Java), and `controller.aspx` (.NET) files for more details.

Module maps

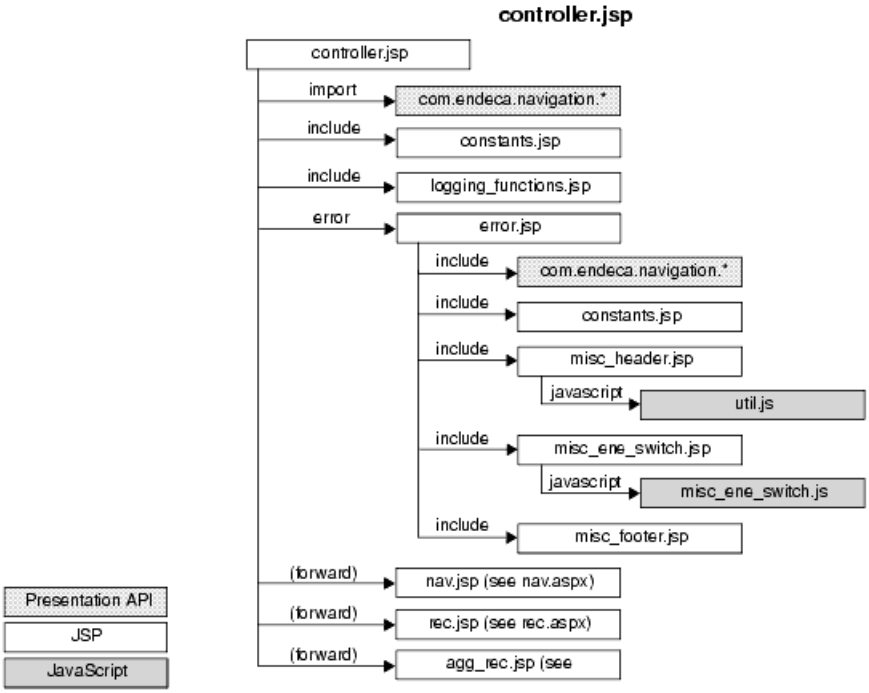
The following diagrams show the relationship between the various UI reference implementation modules. The diagrams are broken into the four primary modules for Java and .NET.

Java module maps

The `controller.jsp` (Java) module is the entry point into the UI reference implementation. It receives the browser request from the application server, formulates the MDEX Engine query, establishes a connection with the MDEX Engine and sends the query. Based on the contents of the query results, the `controller` module determines whether the request was a navigation, a record,

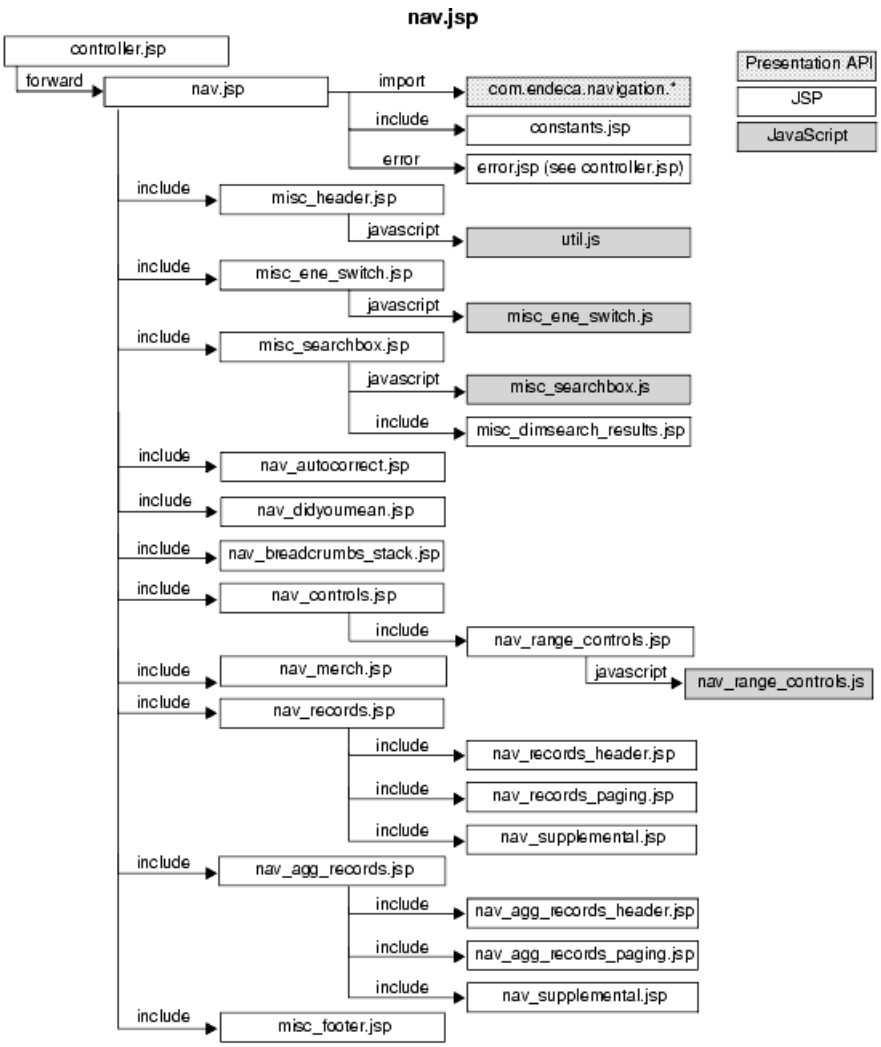
or an aggregated record request. For navigation requests, `controller` forwards the request to the `nav` module.

The following diagram shows the `controller` module map:



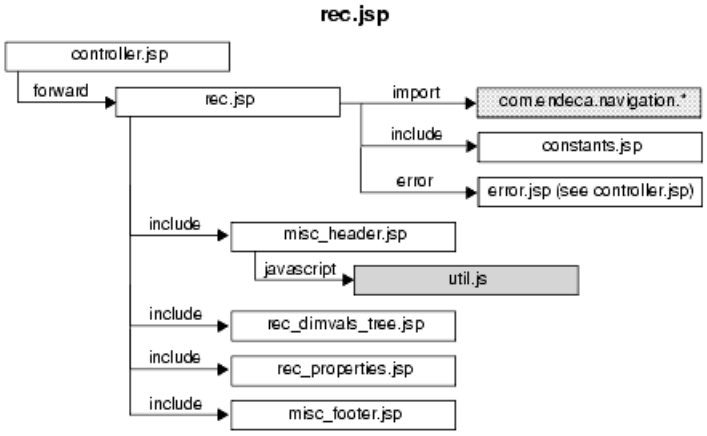
The `nav.jsp` (Java), using other included `nav` modules, renders the main navigation page, including the navigation controls, navigation descriptors, and a record set.

The following diagram shows the `nav` module map:



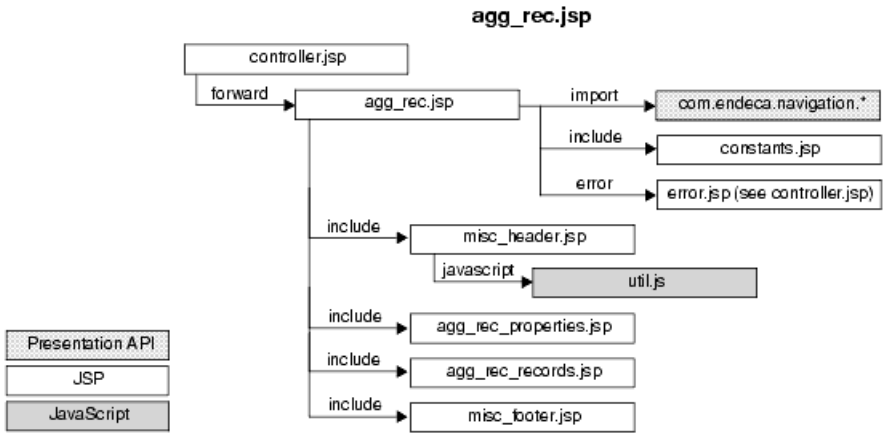
For record requests, `controller` forwards the request to the rec.jsp (Java) module which, along with its child `rec_*` modules, is responsible for rendering a record page for a single record.

The following diagram shows the `rec` module map:



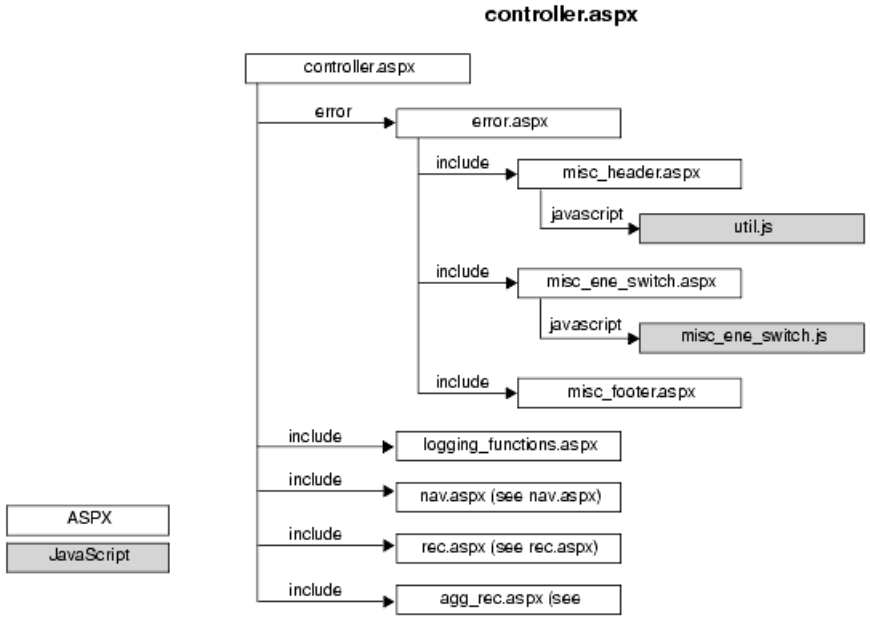
For aggregated record requests, `controller` forwards the request to the `agg_rec.jsp` (Java) module which, again, along with its child `agg_rec_*` modules, renders a page for an aggregated Endeca record.

The following diagram shows the `agg_rec` module map:

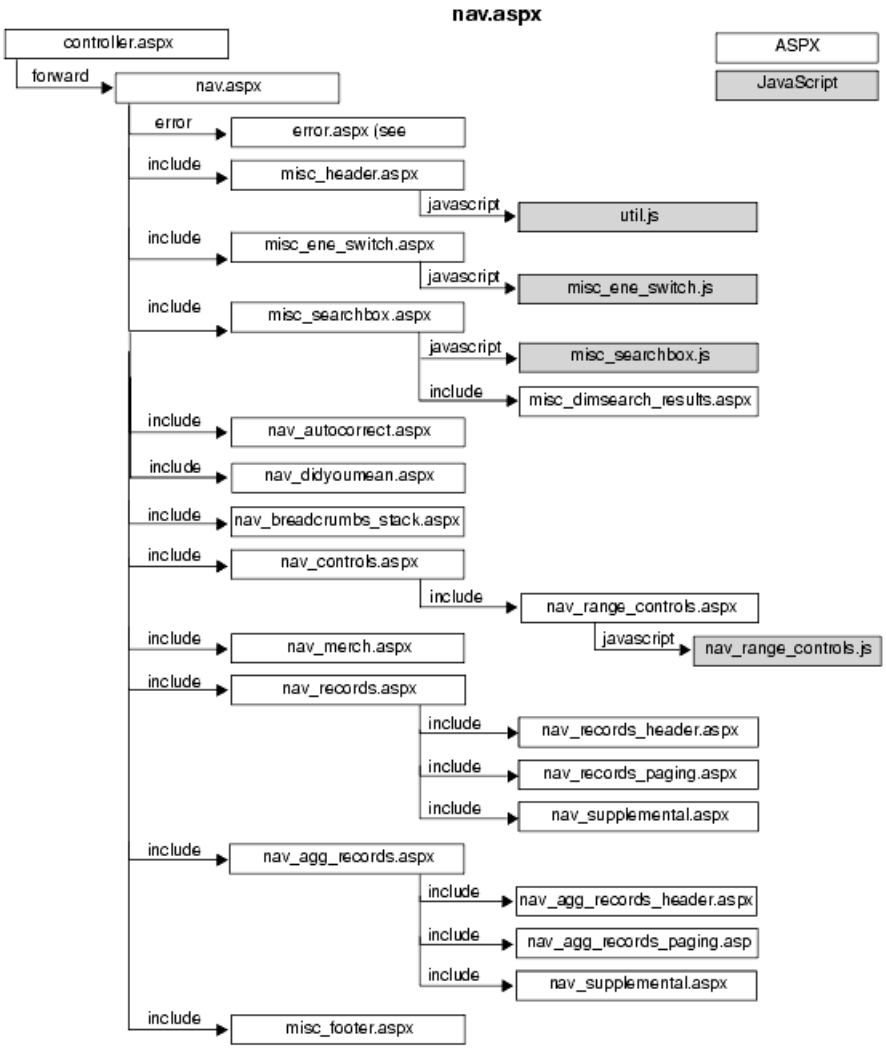


.NET module maps

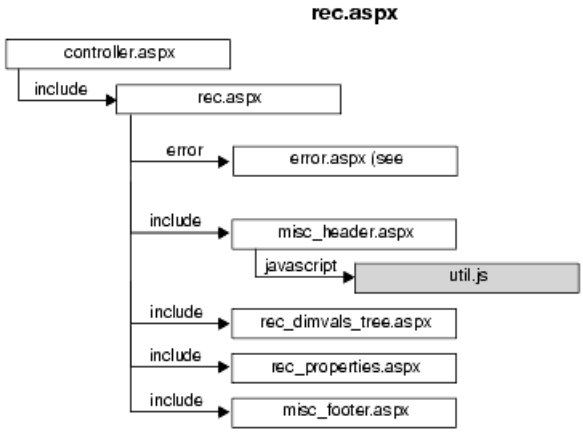
The following diagram shows the `controller` module map:



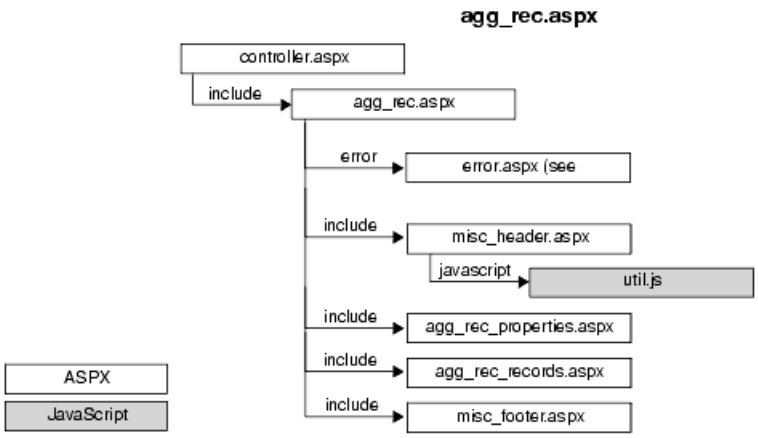
The following diagram shows the `nav` module map:



The following diagram shows the `rec` module map:



The following diagram shows the `agg_rec` module map:



Module descriptions

The table in this topic provides brief descriptions of the UI reference implementation modules.

Refer to the comments in the individual module files for more detailed information. Reference implementation module files are located in:

- Java:
  - `$ENDECA_REFERENCE_DIR/endeca_jspref` on UNIX
  - `%ENDECA_REFERENCE_DIR%\endeca_jspref` on Windows
- .NET:

  `ENDECA_REFERENCE_DIR\endeca_ASP.NETref`

Note: In the following table, the module names do not contain file extensions. Unless otherwise noted, it is assumed that the modules are present in both Java and .NET environments, and that the file extensions are `.jsp` for Java and `.aspx` for .NET. Some modules have specific file extensions; this in indicated in the module name. Similarly, some modules are specific to Java or .NET environments only; this is indicated in the module description.

Module	Description
`controller`	Initiates the primary MDEX Engine query and determines which type of page to render (navigation, record, or aggregated record).
`constants.jsp`	In Java only: Functions as a repository for variables that do not change across requests.
`global.asax`	Functions as a repository for special event handlers that are run automatically when certain ASP events occur.

Module	Description
error	Handles error conditions.
misc_header	A general-use page header used by all page types (navigation, record, aggregated record, and error).
misc_footer	A general-use page footer used by all page types (navigation, record, aggregated record, and error).
util.js	A collection of utility routines used by various JavaScript functions to create new queries from browser request URLs.
logging_functions	Adds logging and reporting capability to your application. This module contains the key/value pairs required by each Endeca report element.
misc_ene_switch and misc_ene_switch.js	Render the MDEX Engine switching widget that allows you to dynamically change the MDEX Engine hostname and port.
nav	Creates the main navigation page, including navigation controls, navigation descriptors, and a record set.
nav_autocorrect	Displays autocorrection for the user's search terms.
nav_didyoumean	Displays alternative suggestions for the user's search terms.
nav_controls	Displays basic navigation controls. This module should be used in conjunction with nav_breadcrumbs_stack
nav_range_controls and nav_range_controls.js	Renders a set of controls that allow you to filter record results according to a specified range. Works with numeric properties only.
nav_breadcrumbs_stack	Display the navigation descriptors for the current query.
nav_merch	Displays merchandising-specific supplemental objects, if any exist, that accompany the results of a navigation query.
nav_supplemental	Displays supplemental objects, if any exist, that accompany the results of a navigation query.
nav_records	Renders the record set results for the current query in a non-formatted display.

Module	Description
`nav_records_header`	Displays a record count and other controls to handle the record set display. Also displays an aggregated record count when records have been aggregated.
`nav_records_paging`	Displays controls for paging through the record set, when applicable.
`nav_agg_records`	Renders a list of records that have been aggregated based on a rollup key.
`nav_agg_records_header`	Displays a record count and other controls to handle the record set display along with an aggregated record count.
`nav_agg_records_paging`	Displays controls for paging through a list of aggregated records, when applicable.
`misc_searchbox` and `misc_searchbox.js`	Render a basic searchbox widget.
`misc_dimsearch_results`	Displays the results of a dimension search.
`rec`	Displays a record page for an individual record.
`rec_dimvals_trees`	Displays the dimension values that have been tagged to the current record.
`rec_properties`	Displays the properties for the current record.
`agg_rec`	Displays an aggregated record page for one aggregated record.
`agg_rec_properties`	Displays the properties associated with an aggregated record's representative record. Displays properties derived from performing calculations on the aggregated record's constituent records.
`agg_rec_records`	Displays the constituent records associated with the current aggregated record.
`coremetrics`	Implements integration of the Coremetrics Online Analytics product.

Tips on using the UI reference implementation modules

This topic contains notes to keep in mind as you are working with the reference modules.

Consider the following characteristics:

- The page components produced by each module are wrapped in `<table>` tags.
- Some of the child modules have dependencies on their parents (for example, the `nav_records` module relies on the `nav` module to retrieve a Navigation object). The module maps provide visual representation of module dependencies.
- There are no dependencies across unrelated features (for example, there are no dependencies between the `nav_controls` and `nav_records` modules).
- All modules reside in the same directory.
- JavaScript routines are provided on a per module basis for those modules with form elements (`misc_ene_switch`, `misc_searchbox`, and `nav_range_controls`).
- There are no cascading stylesheets.

Related Links

Module maps on page 39

The following diagrams show the relationship between the various UI reference implementation modules. The diagrams are broken into the four primary modules for Java and .NET.

Non-MDEX Engine URL parameters

Although we have attempted to keep the UI reference implementation as pure as possible, it is still necessary to use some non-MDEX Engine URL parameters to maintain application state independent of the MDEX Engine query.

It is important, when building your own application, that you remove these parameters (unless they are required by your application). For example, if the MDEX Engine location is specified in a configuration file, it is no longer necessary to maintain or support the `eneHost` and `enePort` parameters.

The non-MDEX Engine URL parameters that are used in the UI reference implementation are described in the following table:

Parameter	Description
`eneHost`	Used by the `misc_ene_switch` module to dynamically set the MDEX Engine hostname with each request. This parameter is particularly useful during development, but should be removed from a production deployment.
`enePort`	Used by the `misc_ene_switch` module to dynamically set the MDEX Engine port with each request.

Parameter	Description
	As with `eneHost`, this parameter is particularly useful during development, but should be removed from a production deployment.
`displayKey`	Used by `nav_records` and `nav_supplemental` to identify the property key that should be used to represent the name of a record. This parameter is useful for data inspection where different data sets may require different property keys to name the records. You should remove the `displayKey` parameter from a production deployment as the record names should never change.
`hideProps`	Provides a simple means of hiding properties for each record in the `nav_records` module.
`hideSups`	Provides a simple means of hiding the data that is returned with each supplemental object in the `nav_supplemental` module.
`hideMerch`	Provides a simple means of hiding the data that is returned with each supplemental merchandising object in the `nav_merch` module.

Chapter 4

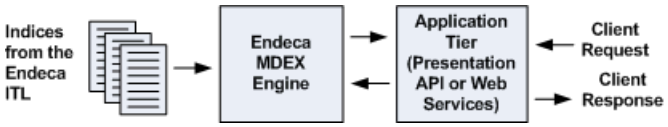
About the Endeca MDEX Engine

Before you begin building your Endeca implementation, it is useful to understand some basics about the Endeca MDEX Engine. This section provides an overview of the MDEX Engine, what it is, and how you work with it.

MDEX Engine overview

The Endeca MDEX Engine is the indexing and query engine that provides the backbone for all Endeca solutions.

The MDEX Engine uses proprietary data structures and algorithms that allow it to provide real-time responses to client requests. The MDEX Engine stores the indices that were created by the Endeca Information Transformation Layer (ITL). After the indices are stored, the MDEX Engine receives client requests via the application tier, queries the indices, and then returns the results.



The MDEX Engine is designed to be stateless. This design requires that a complete query be sent to the MDEX Engine for each request. The stateless design of the MDEX Engine facilitates the addition of MDEX Engine servers for load balancing and redundancy. Because the MDEX Engine is stateless, any replica of an MDEX Engine on one server can reply to queries independently of a replica on other MDEX Engine servers.

Consequently, adding replicas of MDEX Engines on additional servers provides redundancy and improved query response time. That is, if any one particular server goes down, a replica of an MDEX Engine provides redundancy by allowing other servers in the implementation to continue to reply to queries. In addition, total response time is improved by using load balancers to distribute queries to a replica MDEX Engine on any of the additional servers.

The MDEX Engine package contains the following components:

MDEX Engine Component	Description
Dgraph	The Dgraph is the name of the process for the MDEX Engine. A typical Endeca implementation includes one or more Dgraphs. Optionally, it can include an Agraph that manages a number of Dgraphs.

MDEX Engine Component	Description
Agraph	The Agraph is the name of the program that runs in a distributed configuration in addition to the Dgraph. The Agraph typically resides on a separate machine. The Agraph program is responsible for receiving requests from clients, forwarding the requests to the distributed Dgraphs, and coordinating the results. From the perspective of the Endeca Presentation API, the Agraph program behaves similarly to the Dgraph program. Agraph-based implementations allow parallelization of query processing. The implementation of this parallelization results from partitioning the set of records into two or more disjoint subsets of records and then assigning each subset to its own Dgraph.  **Note:** Starting with the MDEX Engine version 6.0, (namely, with installations on the 64-bit platforms) a more powerful Dgraph can accommodate much larger data sets without the need to implement an Agraph.
Dgidx	Dgidx is the indexing program that reads the tagged Endeca records that were prepared by Forge and creates the proprietary indices for the Endeca MDEX Engine.
Agidx	Agidx is the program that creates a set of Agidx indices which support the Agraph program in a distributed environment.
dgwordlist	The `dgwordlist` utility is used to manually compile the text-based `worddat` dictionary into the binary `spelldat` dictionary. This enables use of the Aspell dictionary module in the MDEX Engine.
enecerts	The Endeca `enecerts` utility creates the SSL certificates.

About the Information Transformation Layer

The Endeca Information Transformation Layer transforms your source data into indices for the Endeca MDEX Engine.

This transformation process does not change the content of your source data, only its representation within your Endeca implementation.

The Information Transformation Layer is an off-line process that performs two distinct functions: data processing and indexing. You run the Information Transformation Layer components at intervals that are appropriate for your business requirements.

Full information about the Information Transformation Layer can be found in the *Endeca Forge Guide*.

Part 2

Record Features

Chapter 5

Working with Endeca Records

This section provides information on handling Endeca records in your Web application.

Displaying Endeca records

This section describes how to display Endeca records, including their properties and dimension values.

Endeca records are the individual items (such as CDs, books, or mutual funds) through which a user is trying to navigate. Note that detailed information on implementing this feature can also be found in the Developer Studio online help.

Displaying a list of Endeca records

Displaying a list of Endeca records is a common task in any Endeca implementation.

A typical implementation will display a summarized list of matching records for the user's current navigation state, together with controls for selecting further refinements.

The record list is often displayed as a table, with each row corresponding to a specific record. Each row displays some identifying information about that specific record, such as a name, title, or identification number.

A list of records is returned with every MDEX Engine query result. The Presentation API can iterate through this list, extract the identifying information for each record, and display a table containing the results.

Displaying each record in the ERecList object

The list of records is returned from an MDEX Engine query as an `ERecList` (Endeca records) or `AggrERecList` (aggregated Endeca records) object.

You use one of these methods to retrieve the records from the `Navigation` object:

- To obtain an `ERecList` object, use the `Navigation.getERecs()` method (Java) or the `Navigation.ERecs` property (.NET).
- To obtain an `AggrERecList` object, use the `Navigation.getAggrERecs()` method (Java) or the `Navigation.AggrERecs` property (.NET).

Note that the Java versions of `ERecList` and `AggrERecList` inherit from `java.util.AbstractList`, so all the iterator and indexing methods are available.

Examples of displaying records

The following code samples show how to obtain a record list, iterate through the list, and print out each record's `Name` property.

The number of records that are returned is controlled by:

- Java: the `ENEQuery.setNavNumERecs()` method
- .NET: the `ENEQuery.NavNumERecs` property

The default number of returned records is 10. These calls must be made before the `query()` method.

For aggregated Endeca records, use:

- Java: the `ENEQuery.setNavNumAggrERecs()` method
- .NET: the `ENEQuery.NavNumAggrERecs` property

The subset of records that are returned is determined by the combination of the offset specified in the `setNavERecsOffset()` method (Java) or the `NavERecsOffset` property (.NET) and the number of records specified in the `setNavNumERecs()` method (Java) or `NavNumERecs` property (.NET). For example, if the offset is set to 50 and the `setNavNumERecs()` method is called with an argument of 35, the MDEX Engine will return records 50 through 85.

Java example

```
// Make MDEX Engine request. usq contains user query
// string and nec is an ENEConnection object.
ENEQueryResults qr = nec.query(usq);
// Get navigation object result
Navigation nav = qr.getNavigation();
// Get record list
ERecList records = nav.getERecs();
// Loop through record list
ListIterator i = records.listIterator();
while (i.hasNext()) {
  ERec record = (ERec)i.next();
  PropertyMap recordProperties = record.getProperties();
  String propName = "";
  // If property has a value
  if (!((String)recordProperties.get("Name")).equals("")) {
    propName = (String)recordProperties.get("Name");
    out.print(propName);
  }
}
```

.NET example

```
// Make Navigation Engine request
ENEQueryResults qr = nec.Query(usq);
// Get Navigation object result
Navigation nav = qr.Navigation;
// Get records
ERecList recs = nav.ERecs;
// Loop over record list
for (int i=0; i<recs.Count; i++) {
  // Get individual record
  ERec rec = (ERec)recs[i];
  // Get property map for representative record
  PropertyMap propsMap = rec.Properties;
  // Get and print Name property
  String propName = "";
```

```
  if (((String)propmap["Name"]) != "") {
    propName = (String)propmap["Name"];
    Response.Write propName;
  }
}
```

Performance impact when listing records

The number of records that are returned from the MDEX Engine will affect performance.

The larger the number of requested records, the longer it will take the MDEX Engine to process them. Therefore, you should carefully use the `setNavNumERecs()` method (Java) or the `NavNumERecs` property (.NET) and the offset specified in the `setNavERecsOffset()` method (Java) or the `NavERecsOffset` property (.NET). These calls should return only the subset of records that you are interested in displaying to the end user.

Displaying record properties

The properties tagged on an Endeca record can be displayed with the record.

Properties are key/value pairs associated with Endeca records that are intended for display once the user has searched or navigated to a record list or an individual record. Properties generally contain more detail about a record than the higher-level dimension values used for navigation.

Common examples of properties for an e-commerce application might be Price, Product Description, and Part Number. As navigable dimensions, these concepts would not be very helpful to the user, because they are so specific. In this case, a dimension of Price Range would be more useful for navigation, with the exact price of each product being a property that is displayed to the user once the record has been located.

Note: There is often overlap between information used for navigation and the entire set of data displayed for each record. Properties are the key/value pairs from the raw data that have not been included for navigation but which are displayed. Thus, each record, when displayed, includes a combined set of navigable data (dimensions) and non-navigable data (properties).

Mapping and indexing record properties

How record properties are displayed depends on how they are mapped and indexed.

Mapping record properties

The property mapper component treats all properties that appear in the raw data files read by the pipeline. Depending on property mapper settings, you can handle each source data property as follows:

- Map the source data property to an existing Endeca dimension or a newly-created Endeca dimension.
- Map the source data property to an existing Endeca property or a newly-created Endeca property.
- Ignore the source data property.

To map record properties so they can be displayed, you configure the property mapper and then specify how the property should be displayed. Both of these steps take place in Developer Studio and are described in the online help.

For details on adding and configuring an Endeca property, see the *Endeca Forge Guide*.

Indexing all properties with Dgidx

By default, the Dgidx indexing program ignores any record property that does not have a corresponding property mapper and does not include it in the MDEX Engine indices. If you use the Dgidx `--nostric¬tattrs` flag, every property found on a record will be indexed.

The MDEX Engine Dgraph program does not have configuration flags to control the behavior of displaying properties.

Accessing properties from records

Properties can be accessed from any Endeca record returned from a navigation query (N parameter) or a record query (R parameter).

To access a property directly on an `ERec` or `AggrERec` object, use the `PropertyMap.getValues()` method (Java) or the `PropertyMap.GetValues()` method (.NET). These methods return a collection of all the values in a record for a particular property.

The following examples show how to access record properties.

Java example

```
if (eneResults.containsNavigation()) {
  Navigation nav = eneResults.getNavigation();
  ERecList erl = nav.getERecs();
  for (int i=0; i < erl.size(); i++) {
    ERec erec = (ERec) erl.get(i);
    // Retrieve all properties from the record
    PropertyMap pmap = erec.getProperties();
    // Retrieve all values for the property named Colors
    Collection colors = pmap.getValues("Colors");
    Iterator it = colors.iterator();
    while (it.hasNext()) {
      String colorValue = (String)it.next();
      // Insert code to use the colorValue variable
    }
  }
}
```

.NET example

```
if (eneResults.ContainsNavigation()) {
  Navigation nav = eneResults.Navigation;
  ERecList recs = nav.ERecs;
  // Loop over record list
  for (int i=0; i<recs.Count; i++) {
    // Get individual record
    ERec rec = (ERec)recs[i];
    // Get property map for record
    PropertyMap propsMap = rec.Properties;
    System.Collections.IList colors = propsMap.GetValues("Colors");
    // Retrieve all values for the Colors property
    for (int j =0; j < colors.Count; j++) {
      String colorValue = (String)colors[j];
      // Insert code to use the colorValue variable
    }
```

```
  }
}
```

Properties returned by the MDEX Engine

This topic describes which mapped properties are returned in response to queries.

The MDEX Engine typically returns additional information with a user query request. This information depends on the nature of the query.

Recall that for properties, you can specify two options in the Property Editor of Developer Studio, **Show with Record** and **Show with Record List**.

When you specify **Show with Record List**, the corresponding `RENDER_CONFIG.XML` file is updated. This indicates to the MDEX Engine which properties it must return as supplemental objects with the list of records.

In the case of mapped record properties, the MDEX Engine behaves as follows:

- It returns only those properties for which you specify **Show with Record List** in Developer Studio.
- It returns these properties consistently in record lists returned as a response to regular user queries, and in record lists returned by the dynamic business rules. (Dynamic business rules enable merchandizing and content spotlighting.)

Note: In terms of XML configuration settings, rule results from the MDEX Engine use the `RENDER_PROD_LIST` setting from the `RENDER_CONFIG.XML` file.

Displaying all properties on all records

You can loop through all properties on all records and display their values.

Once a `Property` object is obtained, its name and value can be accessed with these calls:

- For Java, use the `Property.getKey()` and `Property.getValue()` methods.
- For .NET, use the `Property.Key` and `Property.Value` properties.

Java example

```
if (eneResults.containsNavigation()) {
  Navigation nav = eneResults.getNavigation();
  ERecList erl = nav.getERecs();
  for (int i=0; i < erl.size(); i++) {
    // Get an individual record
    ERec rec = (ERec) erl.get(i);
    // Get property map for record
    PropertyMap propsMap = rec.getProperties();
    // Get property iterator for record
    Iterator props = propsMap.entrySet().iterator();
    // Loop over properties iterator
    while (props.hasNext()) {
      // Get individual record property
      Property prop = (Property)props.next();
      // Display property name and value
      %><tr>
      <td><%= prop.getKey() %>: </td>
```

```
      <td><%= prop.getValue() %></td>
      </tr><%
    }
  }
}
```

.NET example

```
Navigation nav = eneResults.Navigation;
ERecList recs = nav.ERecs;
// Loop over record list
for (int i=0; i<recs.Count; i++) {
  // Get individual record
  ERec rec = (ERec)recs[i];
  // Get property map for record
  PropertyMap propsMap = rec.Properties;
  System.Collections.IList props = propmap.EntrySet;
  // Loop over properties iterator
  for (int j =0; j < props.Count; j++) {
    Property prop = (Property)props[j];
    // Display property name and value
    %><tr>
    <td><%= prop.Key %>: </td>
    <td><%= prop.Value %></td>
    </tr><%
  }
}
```

Displaying dimension values for Endeca records

The dimension values tagged on an Endeca record can be displayed with the record.

Dimensions are the hierarchical, navigable concepts applied to Endeca records. Dimension values are the specific terms within a given dimension that describe a record or set of records.

Each record's dimension values can be displayed when the record appears in a record list or on an individual record page. The latter case is the more common use of this feature, because record properties are also available for display and are less expensive to use for this purpose.

A common purpose for displaying an individual record's dimension values is to allow the end user to pivot to a new record set based on a subset of dimension values displayed for the current record. For example, an apparel application might have a record page for shirt ABC that displays the shirt's dimension values:

```
Sleeve=short
fabric=100% cotton
Style=Oxford
Size=L
```

Each value has a checkbox next to it. The end user can then check the boxes for dimension values:

```
Sleeve=short
Style=Oxford
Size=L
```

The requested dimension values will arrive at a record set that includes shirt ABC along with all other Large, short-sleeve, Oxford shirts (regardless of whether the shirt fabric is 100% cotton).

Configuring how dimensions are displayed

You use Developer Studio to create and configure dimensions.

Dimensions and their hierarchy of values are created in Developer Studio Dimensions view, and are referenced in a dimension adapter component. See the *Endeca Forge Guide* for more information on creating dimensions.

By default, dimension values are displayable for a record query result but not for a navigation query result. This behavior can be changed in Developer Studio.

Dimension values are ranked in either Developer Studio or the `dval_rank.xml` file. Note that in either case, if dimension values are assigned ranks with values greater than 16,000,000, unpredictable ranking behavior may result.

No Dgidx or Dgraph flags are necessary to enable displaying dimension values.

Accessing dimensions from records

Dimension values can be accessed from any Endeca record returned from a record query (R parameter).

If dimensions have been configured as in the previous section, they can also be accessed from records returned from a navigation query (`N` parameter).

To access a dimension value directly on an `ERec` object, use:

- Java: the `ERec.getDimValues()` method
- .NET: the `ERec.DimValues` property

These return an `AssocDimLocationsList` object that contains all the values in a record for a particular dimension.

The following code snippets show how to retrieve the dimension values from a list of records.

Java example

```
ERecList recs = eneResults.getERecs();
// Loop over record list to get the dimension values
for (int i=0; i < recs.size(); i++) {
  ERec rec = (ERec)recs.get(i);
  // Get list of tagged dimension location groups for record
  AssocDimLocationsList dims = (AssocDimLocationsList)rec.getDimValues();
  for (int j=0; j < dims.size(); j++) {
    // Get individual dimension and loop over its values
    AssocDimLocations dim = (AssocDimLocations)dims.get(j);
    for (int k=0; k < dim.size(); k++) {
      // Get attributes from a specific dim val
      DimLocation dimLoc = (DimLocation)dim.get(k);
      DimVal dval = dimLoc.getDimValue();
      String dimensionName = dval.getDimensionName();
      long dimensionId = dval.getDimensionId();
      String dimValName = dval.getName();
      long dimValId = dval.getId();
      // Enter code to display the dimension name and
      // dimension value name. The Dimension ID and
      // dimension value ID may be needed for URLs.
    }
  }
}
```

.NET example

```
ERecList recs = eneResults.ERecs;
for (int i=0; i < recs.Count; i++) {
  ERec rec = (ERec)recs[i];
  // Get list of tagged dimension location groups for record
  AssocDimLocationsList dims = rec.DimValues;
  // Loop through dimensions
  for (int j=0; j < dims.Count; j++) {
    // Get individual dimension
    AssocDimLocations dim = (AssocDimLocations) dims[j];
    // Loop through each dim val in the dimension group
    for (int k=0; k < dim.Count; k++) {
      // Get specific dimension value and path
      DimLocation dimLoc = (DimLocation) dim[k];
      // Get dimension value
      DimVal dval = dimLoc.DimValue;
      String dimensionName = dval.DimensionName;
      Long dimensionId = dval.DimensionId;
      String dimValName = dval.Name;
      Long dimValId = dval.Id;
      // Enter code to display the dimension name and
      // dimension value name. The Dimension ID and
      // dimension value ID may be needed for URLs.
    }
  }
}
```

Performance impact when displaying dimensions

Displaying too many dimensions can cause a performance hit.

The main purpose of dimension values is to enable navigation through the records. Passing dimension values through the system consumes more resources than passing properties. Therefore, the default behavior of the MDEX Engine is to return dimension values on records only when a record query request has been made (not for navigation query requests).

As mentioned above, this behavior can be changed. However, the developer should exercise caution when passing dimension values through to the record list, because doing this with too many dimensions can cause a performance hit.

Paging through a record set

A paging UI control is helpful if many records are returned.

An MDEX Engine query 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.

In the following example of a user interface control for paging, Page 2 of 27 pages is currently being displayed:

Using the No parameter in queries

The No parameter can be used for paging.

Paging is implemented by using the `No` parameter in an MDEX Engine query, using the following syntax:

```
No=<number_of_records_offset>
```

The `No` parameter specifies the offset for the first record that is returned in the query result. The default offset is zero if the `No` parameter is not specified. For example, if you want an MDEX Engine query to return a list of records that starts at the 20th record, you would use this in the query:

```
No=20
```

It is important to note the `ERecList` object is one-based and the offset parameter is zero-based. For example, if there are ten records displayed in the record list and parameter `No=10` is in the navigation state, the `ERecList` object returned will have records 11-20.

The paging functionality does not require any Developer Studio configuration, and no Dgidx or Dgraph flags are necessary.

Using paging control methods

The Presentation API includes several methods that you can use for paging.

The `ENEQuery` object is the initial access point for providing the paging controls for the entire record set. By default, the navigation query returns a maximum of ten records to the `Navigation` object for display. To override this setting, use:

- Java: the `ENEQuery.setNavNumERecs()` method
- .NET: the `ENEQuery.NavNumERecs` property

The default offset for a record set is zero, meaning that the first ten records are displayed. The default offset can be overridden in one of two ways:

- Generate a URL with an explicit `No` parameter.
- For Java, use the `ENEQuery.setNavERecsOffset()` method. For .NET, use the `ENEQuery.NavERecsOffset` property

To find out the offset used in the current navigation state, use the `ENEQuery.getNavERecsOffset()` method (Java) or the `ENEQuery.NavERecsOffset` property (.NET). By adding one to the offset parameter, the application can calculate the number of the first record on display.

To ascertain the total number of records being returned by the navigation query, use the `Navigation.getTotalNumERecs()` method (Java) or the `Navigation.TotalNumERecs` property. If the number of records returned is less than the number of records returned by the `ENEQuery.setNavNumERecs()` method (Java) or the `ENEQuery.NavNumERecs` property (.NET), then no paging controls are needed.

The following table provides guidance about the paging logic necessary in your Web application to calculate the previous, next, and last pages.

|< **First**	**<** **Previous**	**>** **Next**	**>|** **Last**
set No = 0	offset - navNum	offset + navNum	totNum - remainder (if remainder < 0) totNum - navNum (if remainder = 0)

where:

- offset = `Navigation.getERecsOffset()` method (Java) or the `Navigation.ERecsOffset` property (.NET)
- navNum = `ENEQuery.getNavNumERecs()` method (Java) or the `ENEQuery.NavNumERecs` property (.NET)
- totNum = `Navigation.getTotalNumERecs()` method (Java) or the `Navigation.TotalNumERecs` property (.NET)
- remainder = totNum / navNum

Note: When using paging controls, consider how paging should interact with other aspects of the application. For example, if the user is paging through the record set and then decides to sort on a property, should the No parameter be reset? The answer depends on the desired functionality of the application.

Chapter 6

Sorting Endeca Records

The sorting functionality allows the user to define the order of Endeca records returned with each navigation query.

About record sorting

When making a basic navigation request, the user may define a series of property/dimension and order (ascending or descending) pairs.

If the user does not specify sort order as part of the query, the MDEX Engine returns query results in the same order that Dgidx stores the records in the index file. Most of the time, this is the same order in which Forge processed the records. For information on changing the order in which Dgidx stores records, see the "Changing the sort order with Dgidx flags" topic later in this section.

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 bottles of wine, all 100 bottles are considered when sorting, even though only the first ten bottles may be returned with the current request.

Record sorting only affects the order of records. It does not affect the ordering of dimensions or dimension values that are returned for query refinement.

Note: Additional information on implementing this feature can be found in the Developer Studio online help.

Related Links

Changing the sort order with Dgidx flags on page 65

You can use an optional Dgidx flag to change the sort order.

Configuring precomputed sort

You can optimize a sort key for a precomputed sort.

Although users can sort on any record at any time, it is also possible to optimize a property or dimension for sort in Developer Studio. This mainly controls the generation of a precomputed sort, and secondarily enables the field to be returned in the API sort keys function. The sort key is an Endeca property or

dimension that exists in the data set. It can be numeric, alphabetical, or geospatial, and determines the type of sort that occurs.

Configuring precomputed sort on a property

To configure precomputed sort on a property, check "Prepare sort offline" in the Property editor.

In addition, the property's Type attribute, which you also set in the Property editor, affects sorting in the following ways:

If Type is set to this:	Records are sorted:
Alpha	In alphabetical order.
Integer or Floating Point	In numeric order.
Geocode	In geospatial order (that is, according to the distance between the specified geocode property and a given reference point).
File Path	Deprecated. Do not use this type.

Configuring precomputed sort on a dimension

To configure a precomputed sort on a dimension, check "Prepare sort offline" in the Dimension editor.

In addition, the dimension's Refinements Sort Order setting, which you also set in the Dimension editor, affects sorting in the following ways:

If Refinements Sort Order is set to this:	Records are sorted:
Alpha	In alphabetical order.
Integer or Floating Point	In numeric order.

Numeric sort on semi-numeric and non-numeric dimension values

When numeric sorting is enabled for a dimension, all of the dimension values are assumed to consist of a numeric (double) part, followed by an optional non-numeric part. That is to say, 3 is evaluated as <3.0, "">. The non-numeric part is used as a secondary sort key when two or more numeric parts are equal. The non-numeric parts are sorted so that an empty non-numeric part comes first in the sort order.

In some cases, a set of primarily numeric dimension values may contain semi-numeric values, such as 1.3A (evaluated as <1.3, "A">, or non-numeric values, such as Other (evaluated as <0.0, "Other">. Numeric sort on such dimension values works as follows:

- For semi-numeric dimension values, dimension values with non-numeric parts are sorted after matching dimension values without non-numeric parts. For example, 1.3A appears after 1.3 when sorted.
- For non-numeric dimension values, the missing numeric part is treated as 0.0. In a data set containing the word Other and the number 0, the system would compare 0 and Other as <0.0, ""> and <0.0, "Other"> and sort 0 before Other.

Putting all of this together, a data set consisting of Other, 1.3A, 0, 3, and 1.3 would sort as follows:

```
0
Other
1.3
1.3A
3
```

Sorting behavior for records without a sort-key value

If an Endeca record does not include a value for the specified sort key, that record is sorted to the bottom of the list, regardless of the sort order. This behavior occurs in both the Dgraph and the Agraph.

For example, the following record set is sorted by P_Year ascending. Note that Record 4 has no P_Year property value.

```
Record 1 (P_Year 1998)
Record 2 (P_Year 2000)
Record 3 (P_Year 2003)
Record 4 (no P_Year property value)
```

If the sort order is reversed to P_Year descending, the new result set would appear in the following order:

```
Record 3 (P_Year 2003)
Record 2 (P_Year 2000)
Record 1 (P_Year 1998)
Record 4 (no P_Year property value)
```

Record 4, because it has no P_Year property value, will always appear last.

Changing the sort order with Dgidx flags

You can use an optional Dgidx flag to change the sort order.

No Dgidx flags are necessary to enable record sorting. If a property or dimension is properly enabled for sorting, it is automatically indexed for sorting.

To change the order in which Dgidx stores records, you can specify a sort order and sort direction (ascending or descending) by using the `--sort` flag with the following syntax:

```
--sort "key|dir"
```

where *key* is the name of a property or dimension on which to sort and *dir* is either `asc` for an ascending order or `desc` for descending (if not specified, the order will be ascending).

You can also specify multiple sort keys in the format:

```
--sort "key_1|dir_1||key_2|dir_2||...||key_n|dir_n"
```

If you specify multiple sort keys, the records are sorted by the first sort key, with ties being resolved by the second sort key, whose ties are resolved by the third sort key, and so on.

Note that if you are using the Endeca Application Controller (EAC) to control your environment, you must omit the quotation marks from the `--sort` flag. Instead, use the following syntax:

```
--sort key_1|dir_1||key_2|dir_2||...||key_n|dir_n
```

There are no Dgraph sort flags. If a property or dimension is properly enabled for sorting when indexed, it is available for sorting when those index files are loaded into the MDEX Engine.

Agraph default sort order and displayed record lists

For Agraph deployments, the sort property should be displayed with record lists.

If a default record sort order is specified in Dgidx based on a property which is not set to show in the record list, an Agraph managing the resulting Dgraphs will not consistently display records in the default sort order.

Each child Dgraph displays its own records in the correct order, but the Agraph does not reliably preserve this order when integrating its child record sets. The resulting record order will be close to—but not the same as—the actual specified default sort order.

To prevent this problem, use Developer Studio's Property editor to enable the "Show with record list" setting for the sort property. This ensures that the Agraph will determine the correct record display order.

URL parameters for sorting

The Ns parameter is used for record sorting.

In order to sort records returned for a navigation query, you must append a sort key parameter (`Ns`) to the query, using the following syntax:

```
Ns=sort-key-names[(geocode)][|order][||…]
```

The `Ns` parameter specifies a list of properties or dimensions by which to sort the records, and an optional list of directions in which to sort. The records are sorted by the first sort key, with ties being resolved by the second sort key, whose ties are resolved by the third sort key, and so on.

The optional order parameter specifies the order in which the property is sorted (`0` indicates ascending, `1` indicates descending). The default sort order for a property is ascending. Whether the values for the sort key are sorted alphabetically, numerically, or geospatially is specified in Developer Studio.

To sort records by their geocode property, add the optional `geocode` argument to the sort key parameter (noting that the sort key parameter must be a geocode property). Records are sorted by the distance from the geocode reference point to the geocode point indicated by the property key.

Sorting can only be performed when accompanying a navigation query. Therefore, the sort key (`Ns`) parameter must accompany a basic navigation value parameter (`N`).

Valid Ns examples

```
N=0&Ns=Price
N=101&Ns=Price|1||Color
N=101&Ns=Price|1||Location(43,73)
```

Related Links

Record Features on page 51

This part contains the following sections:

Sort API methods

The Presentation API includes several methods that you can use for record sorting.

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

However, there are specific objects and method calls that can be accessed from a `Navigation` object that return a list of valid record sort properties, as shown in the examples below. (This data is only available from navigation and record search requests.)

The `ERecSortKeyList` object is an array containing `ERecSortKey` objects. Use these calls to get the `ERecSortKey` sort keys in use for this navigation:

- Java: `Navigation.getSortKeys()` method
- .NET: `Navigation.SortKeys` property

Each `ERecSortKey` object contains the name of a property or dimension that has been enabled for record sorting, as well as a Boolean flag indicating whether the current request is being sorted by the given sort key, and an integer indicating the direction of the current sort, if any (`ASCENDING`, `DESCENDING`, or `NOT_ACTIVE`).

The `Navigation` object also has a method which provides an `ERecSortKeyList` containing only the sort keys used in the returned results:

- Java: `getActiveSortKeys()`
- .NET: `GetActiveSortKeys()`

Note that in order to get an active sort key that is not precomputed for sort, you must use:

- Java: the `ENEQuery.getNavActiveSortKeys()` method
- .NET: the `ENEQuery.GetNavActiveSortKeys()` method

Java example of methods that return sort properties

```
ERecSortKeyList keylist = nav.getSortKeys();
for (int i=0; i < keylist.size(); i++) {
  ERecSortKey key = keylist.getKey(i);
  String name = key.getName();
  int direction = key.getOrder();
}
```

.NET example of methods that return sort properties

```
ERecSortKeyList keylist = nav.SortKeys;
for (int i=0; i < keylist.Count; i++) {
  ERecSortKey key = keylist[i];
  String name = key.Name;
  int direction = key.GetOrder();
}
```

Related Links

Record Features on page 51
This part contains the following sections:

Troubleshooting application sort problems

This topic presents some approaches to solving sorting problems.

Although you can implement sorting without using the `ERecSortKey` objects and methods to retrieve a list of valid keys, this approach does require that the application have its parameters coordinated with the data set. The application must have the `Ns` parameters hard-coded, and will rely on the MDEX

Engine having corresponding parameters enabled. If a navigation request is made with an invalid `Ns` parameter, the MDEX Engine returns an error.

If the records returned with a navigation request do not seem to respect the sort key parameter, there are some potential problems:

- Was the property/dimension specified as a numeric when it is actually alphanumeric? Or vice versa? In this case, the MDEX Engine returns a valid response, but the sorting may be incorrect.
- Was the specified property a derived property? Derived properties cannot be used for sorting records.
- If a record has multiple property values or dimension values for a single property or dimension, the MDEX Engine 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, properties and dimensions that are enabled for sorting should only have one value assigned per record.
- If an application has properties and dimensions with the same name and a sort is requested by that name, the MDEX Engine arbitrarily picks either the property or dimension for sorting. In general, using the same name for a properties and dimensions should be avoided.
- If certain records in a record set 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.

Related Links

Record Features on page 51

This part contains the following sections:

Performance impact for sorting

Sorting records has an impact on performance.

Keep the following factors in mind when attempting to assess the performance impact of the sorting feature:

- Record sorting is a cached feature. That means that each dimension or property enabled for sorting increases the size of the Dgraph process. The specific size of the increase is related to the number of records included in the data set. Therefore, only dimensions or properties that are specifically needed by an application for sorting should be configured as such. Sorting gets slower as paging gets deeper.
- Because sorting is an indexed feature, each property enabled for sorting increases the size of both Dgidx process as well as the MDEX Engine process. (The specific size of the increase is related to the number of records included in the data set.) Therefore only properties that are specifically needed by an application for sorting should be configured as such.
- In cases where the precomputed sort is rarely or never used (such as when the number of search results is typically small), the memory can be saved.

Related Links

Record Features on page 51

This part contains the following sections:

Using geospatial sorting

You implement geospatial sorting by using geocode properties as sort keys.

Geocode properties represent latitude and longitude pairs to Endeca records.

Result sets that have geocode properties can be sorted by the distance of the values of the geocode properties to a given reference point. They can also be filtered (using the `Nf` parameter) by these same values.

For example, if the records of a particular data set represent individual books that a large vendor has for sale at a variety of locations, each book could be tagged with a geocode property (named Location) that holds the store location information for that particular book. Users could then filter result sets to see only books that are located within a given distance, and then sort those books so that the closest books display first.

A geocode property on an Endeca record may have more than one value. In this case, the MDEX Engine compares the query's reference point to all geocode values on the record and returns the record with the closest distance to the reference point.

Configuring geospatial sorting

You can configure a geocode property and add a Perl manipulator to the pipeline if necessary.

Configuring a geocode property as the sort key

Use Developer Studio's Property editor to configure a geocode property for record sort. In the Property editor, the "Prepare sort offline" checkbox enables record sorting on the property.

Configuring the pipeline for a geocode property

Dgidx accepts geocode data in the form:

```
latvalue,lonvalue
```

where each is a double-precision floating-point value:

- *latvalue* is the latitude of the location in whole and fractional degrees. Positive values indicate north latitude and negative values indicate south latitude.
- *lonvalue* is the longitude of the location in whole and fractional degrees. Positive values indicate east longitude, and negative values indicate west longitude.

For example, Endeca's main office is located at 42.365615 north latitude, 71.075647 west longitude. This geocode should be supplied to Dgidx as:

```
42.365615,-71.075647
```

If the input data is not available in this format, it can be assembled from separate properties with a Perl manipulator created in Developer Studio. The Method Override editor would have the following Perl code:

```
#Get the next record from the first record source.
my $rec = $this->record_sources(0)->next_record;
return undef unless $rec;
#Return an array of property values from the record.
my @pvals = @{$rec->pvals};
#Return the value of the Latitude property.
my @lat = grep {$_->name eq "Latitude"} @{$rec->pvals};
```

```
#Return the value of the Longitude property.
my @long = grep {$_->name eq "Longitude"} @{$rec->pvals};
#Exit if there is more than one Latitude property.
if (scalar (@lat) !=1) {
  die("Perl Manipulator ", $this->name,
  " must have exactly one Latitude property.");
}
#Exit if there is more than one Longitude property.
if (scalar (@long) !=1) {
  die("Perl Manipulator ", $this->name,
  " must have exactly one Longitude property.");
}
#Concatenate Latitude and Longitude into Location.
my $loc = $lat[0]->value . "," . $long[0]->value;
#Add new Location property to record.
my $pval = new EDF::PVal("Location", $loc);
$rec->add_pvals($pval);

return $rec;
```

URL parameters for geospatial sorting

The Ns parameter can specify a geocode property for record sorting.

As with general record sort, use the `Ns` parameter to specify a record sort based on the distance of a geocode property from a given reference point. The Ns syntax for a geocode sort is:

```
Ns=geocode-property-name(geocode-reference-point)
```

The geocode-reference-point is expressed as a latitude and longitude pair in exactly the same comma-separated format described in the previous topic. For example, if you want to sort on the distance from the value of the geocode property Location to the location of Endeca's main office, add the following sort specification to the query URL:

```
Ns=Location(42.365615,-71.075647)
```

Geocode properties cannot be sorted except in relation to their distance to a reference point. So, for example, the following specification is invalid and generates an error message:

```
Ns=Location
```

Geospatial sort API methods

The Presentation API includes methods that you can use for geospatial sorting.

The ERecSortKey class is used to specify all sort keys, including geocode sort keys.

To create a geocode sort key, use the four-parameter constructor:

```
ERecSortKey(String propertyName,
            boolean isAscending,
            double latitude,
            double longitude);
```

An `ERecSortKey` has accessor methods for the latitude and longitude of the reference location:

- Java: `getReferenceLatitude()` and `getReferenceLongitude()`
- .NET: `GetReferenceLatitude()` and `GetReferenceLongitude()`

Note that calling these methods on a non-geocode sort key causes an error.

The type of sort key (GEOCODE_SORT_KEY or ALPHA_NUM_SORT_KEY) can be determined using the getType() method (Java) or the Type property (.NET).

The code samples below show the use of the accessor methods.

Although you can implement sorting without first retrieving a list of valid sorting keys from the result object, this approach requires that the application have its parameters coordinated properly with the MDEX Engine. The application will have the Ns parameters hard-coded, and will rely on the MDEX Engine to have corresponding parameters. If a navigation request is made with an invalid Ns parameter, that request returns an error from the MDEX Engine.

Java example of geocode API methods

```
ERecSortKey sk = new ERecSortKey("Location", true, 43.0, -73.0);
// get sortKeyName == "Location"
String sortKeyName = sk.getName();
// get latitude == 43.0
double latitude = sk.getReferenceLatitude();
// get longitude == -73.0
double longitude = sk.getReferenceLongitude();
// get keyType == com.endeca.navigation.ERecSortKey.GEOCODE_SORT_KEY
int keyType = sk.getType();
// get sortOrder == com.endeca.navigation.ERecSortKey.ASCENDING
int sortOrder = sk.getOrder();
```

.NET example of geocode API methods

```
ERecSortKey sk = new ERecSortKey("Location", true, 43.0, -73.0);
// get sortKeyName == "Location"
string sortKeyName = sk.Name;
// get latitude == 43.0
double latitude = sk.GetReferenceLatitude();
// get longitude == -73.0
double longitude = sk.GetReferenceLongitude();
// get keyType == Endeca.Navigation.ERecSortKey.GEOCODE_SORT_KEY
int keyType = sk.Type;
// get sortOrder == com.endeca.navigation.ERecSortKey.ASCENDING
int sortOrder = sk.GetOrder();
```

Dynamic properties created by geocode sorts

When a geospatial sort is applied to a navigation query, the MDEX Engine creates a pair of dynamic properties for each record returned.

The dynamic properties showing the distance (in kilometers and miles, respectively) between the record's geocode address and that specified in the sort key.

The names of these properties use the format:

```
kilometers_to_key(latvalue,lonvalue)

miles_to_key(latvalue,lonvalue)
```

where *key* is the name of the geocode property, and *latvalue* and *lonvalue* are the values specified for the sort.

For example, if Location is the name of a geocode property, this Ns sort parameter:

```
Ns=Location(38.9,77)
```

will create these properties for the record that is tagged with the geocode value of 42.3,71:

```
kilometers_to_Location(38.900000,77.000000): 338.138890
miles_to_Location(38.900000,77.000000): 210.109700
```

These properties are not persistent and are informational only. There is no configuration associated with the properties and they cannot be disabled. Note that applying both a geocode sort and a geocode range filter in the same query causes both sets of dynamic properties to be generated.

Performance impact for geospatial sorting

Geospatial sorting affects query-time performance.

Geospatial sorting and filtering is a query-time operation. The computation time it requires increases as larger sets of records are sorted and filtered. For best performance, it is preferable to apply these operations once the set of records has been reduced by normal refinement or search.

Chapter 7

Using Range Filters

You can use range filters for navigation queries.

About range filters

Range filter functionality allows 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 dimension 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 wines within a price range, say between $10 and $20.

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 a dimension value, even though it is not a specific system-defined dimension value.

You can use a range filter in a query on record properties and on dimensions.

Configuring properties and dimensions for range filtering

Using range filters does not require Dgidx or Dgraph configuration flags.

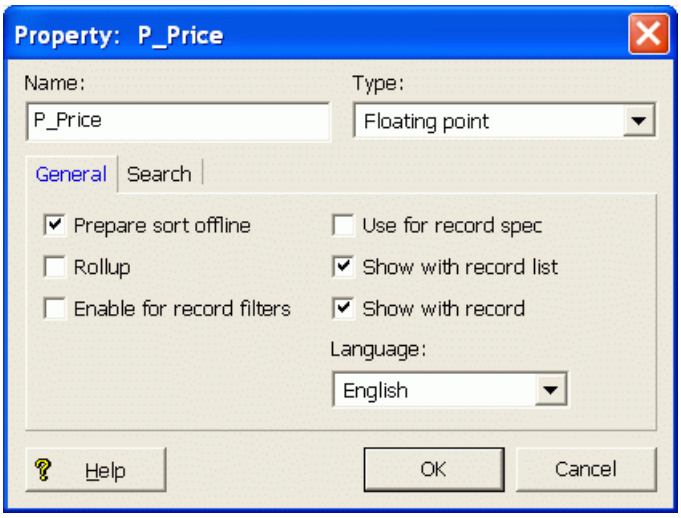
Range filters can be applied to either properties or dimensions of the following types:

- Properties of type Numeric (Integer, Floating point, DateTime) or type Geocode
- Dimensions of type Numeric that contain only Integer or Floating point values.

Note: Although dimensions do not have type, configuring a dimension's refinement sort order to be numeric causes the dimension to be treated as numeric in range filters, so long as all values can be parsed as integral or floating point values.

For values of properties and dimensions of type Floating point, you can specify values using both decimal (0.00...68), and scientific notation (6.8e-10).

Use Developer Studio to configure the appropriate property type. For example, the following property is configured to be of type Floating point:



Running queries with range filtering on dimensions is done with the same `Nf` parameter that is used for queries with range filtering on properties.

For example, this is a query with a range filter on a dimension. In this example, the name of the dimension is ContainsDigit and the records are numbers:

```
N=0&Nf=ContainsDigit|GT+8
```

This query returns all numbers that contain values greater than 8. As the example shows, running a query with a range filter on a dimension makes sense only for dimensions with values of type Integer or Floating Point.

No Dgidx flags are necessary to enable range filters. All range filter computational work is done at request-time.

Likewise, no MDEX Engine configuration flags are necessary to enable range filters. All numeric properties and dimensions and all geocode properties are automatically enabled for use in range filters.

URL parameters for range filters

The Nf parameter denotes a range filter request.

A range filter request requires an `Nf` parameter. However, because a range filter is actually a modifier for a basic navigation request, it must be accompanied by a standard `N` navigation request (even if that basic navigation request is empty).

Only records returned by the basic navigation request (`N`) are considered when evaluating the range filter. (Range filters and navigation dimension values together form a Boolean AND request.)

The `Nf` parameter has the following syntax:

```
Nf=filter-key|function[+geo-ref]+value[+value]
```

The single range filter parameter specifies three separate components of a complete range filter:

- filter-key
- function
- value

filter-key is the name of a numeric property, geocode property, or numeric dimension. Only a single property key can be specified per range filter.

function is one of the following:

- `LT` (less than)
- `LTEQ` (less than or equal to)
- `GT` (greater than)
- `GTEQ` (greater than or equal to)
- `BTWN` (between)
- `GCLT` (less than, for geocode properties)
- `GCGT` (greater than, for geocode properties)
- `GCBWTN` (between, for geocode properties)

value is one or more numeric fields defining the actual range. The `LT`, `LTEQ`, `GT`, and `GTEQ` functions require only a single value. The `BTWN` function requires two value settings, with the smaller value listed first and the larger value listed next, separated by a plus sign (+) delimiter.

geo-ref is a geocode reference point that must be specified if one of the geocode functions has been specified (`GCLT`, `GCGT`, `GCBTWN`). This is the only case where a geocode reference point may be specified. When a geocode filter is specified, the records are filtered by the distance from the filter key (a geocode property) to geo-ref (the geocode reference point).

URL parameters for geocode filters

When used with a geocode property, the Nf parameter specifies a range filter based on the distance of that geocode property from a given reference point.

The Nf syntax for a geocode range filter is:

`Nf=`*`filter-key`*`|`*`function`*`+`*`lat`*`,`*`lon`*`+`*`value`*`[+`*`value`*`]`

filter-key is the name of a geocode property and function is the name of a geocode function.

lat and *lon* are a comma-separated latitude and longitude pair: *latv* is the latitude of the location in whole and fractional degrees (positive values indicate north latitude and negative values indicate south latitude). *lon* is the longitude of the location in whole and fractional degrees (positive values indicate east longitude and negative values indicate west longitude). The records are filtered by the distance from the filter key to the latitude/longitude pair.

The available geocode functions are:

- `GCLT` – The distance from the geocode property to the reference point is less than the given amount.
- `GCGT` – The distance from the geocode property to the reference point is greater than the given amount.
- `GCBTWN` – The distance from the geocode property to the reference point is between the two given amounts.

Distance limits in range filters are always expressed in kilometers.

For example, assume that the following parameter is added to the URL:

```
Nf=Location|GCLT+42.365615,-71.075647+10
```

The query will return only those records whose location (in the Location property) is less than 10 kilometers from Endeca's main office.

Dynamic properties created by geocode filters

When a geocode filter is applied to a navigation query, the MDEX Engine creates a pair of dynamic properties for each record returned.

These dynamic properties are similar to those created from geocode sorts.

The properties show the distance (in kilometers and miles, respectively) between the record's geocode address and that specified in the filter.

The property names are composed using the name of the geocode property or dimension and the values specified in the geocode filter.

For example, if Location is the name of a geocode property, this `Nf` parameter:

```
Nf=Location|GCLT+38.9,77+500
```

will create these properties for the record that is tagged with the geocode value of 42.3,71:

```
kilometers_to_Location|GCLT 38.900000,77.000000 500.000000: 338.138890
miles_to_Location|GCLT 38.900000,77.000000 500.000000: 210.109700
```

The properties are not persistent and are informational only (that is, they indicate how far the record's geocode value is from the given reference point). There is no configuration associated with the properties and they cannot be disabled. Note that applying both a geocode sort and a geocode range filter in the same query causes both sets of dynamic properties to be generated.

Using multiple range filters

A query can contain multiple range filters.

In a more advanced application, users may want to filter against multiple range filters, each with a different filter key and function. Such a request is implemented with the following query parameter syntax:

```
Nf=filter-key1|function1+value[+value]|filter-key2|function2+value[+value]
```

In this case, each range filter is evaluated separately, and only records that pass both filters (and match any navigation parameters specified) are returned. For example, the following query is valid:

```
N=0&Nf=Price|BTWN+9+13|Score|GT+80
```

The user is searching for bottles of wine between $9 and $13 with a score rating greater than 80.

Examples of range filter parameters

This topic shows some valid and invalid examples of using the Nf parameter in queries.

Consider the following examples that use these four records:

Record	Wine Type dimension value	Price property	Description property
1	Red (Dim Value 101)	10	Dark ruby in color, with extremely ripe…
2	Red (Dim Value 101)	12	Dense, rich and complex describes this '96 California…

Record	Wine Type dimension value	Price property	Description property
3	White (Dim Value 102)	19	Dense and vegetal, with celery, pear, and spice flavors…
4	Other (Dim Value 103)	20	Big, ripe and generous, layered with honey…

Example 1

Assume that the following query is created:

```
N=0&Nf=Price|GT+15
```

This navigation request has a range filter specifying the Price property should be greater than 15 (with no dimension values specified). The following `Navigation` object is returned:

```
2 records (records 3 and 4)
2 refinement dimension values (White and Other)
```

Example 2

This example uses the following query:

```
N=101&Nf=Price|LT+11
```

This navigation request specifies the Red dimension value (dimension value 101) and a range filter specifying a price less than 11. The following `Navigation` object is returned:

```
1 record (record 1)
(No additional refinements)
```

Example 3

This query:

```
N=0&Nf=Price|BTWN+9+13
```

would return records 1 and 2 from the sample record set. Notice that the smaller value, `9`, is listed before the larger value, `13`.

Invalid examples

The following query is invalid because it is missing the Navigation parameter (`N`):

```
Nf=Price|LT+9
```

This following query is incorrect because of an invalid dimension (the Food dimension is misspelled as Foo):

```
N=0&Nf=Foo|LT+11
```

The following query, which has an incorrect number of values for the `GT` function, is also incorrect:

```
N=0&Nf=Price|GT+20+30
```

Rendering the range filter results

The results of a range filter request can be rendered in the UI like any 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.

Unlike the record search feature, however, there are no methods to access a list of valid range filter properties or dimensions. This is because the properties and dimensions 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 properties and dimensions that a user is allowed to filter against must be correctly identified as numeric or geocode in the instance configuration.

Troubleshooting range filter problems

This topic presents some approaches to solving range filter problems.

Similar to record search, the user-specified interaction of this feature allows a user to request a range that does not match any records (as opposed to the system-controlled interaction of Guided Navigation in which the MDEX Engine controls the refinement values presented to the user). Therefore, it is possible for a user to make a dead-end request when using a range filter. Applications implementing range filters need to account for this.

If a range filter request specifies a property or dimension that does not exist in the MDEX Engine, the query throws an `ENEConnectionException` in the application. The MDEX Engine error log will output the following message:

```
[Sun Dec 21 16:03:17 2008] [Error]
(PredicateFilter.cc::47) - Range filter does not specify a legal dimension

or property name.
```

If a range filter request does not specify numeric range values, the query also throws an `ENEConnectionException` in the application. The MDEX Engine error log will output the following message:

```
[Sun Dec 21 17:09:27 2008] [Error]
(ValuePredicate.cc::128) - Error parsing numeric argument
<argument> in predicate filter.
```

If the specified property or dimension exists but is not configured as numeric or geocode, the query will not throw an exception. But it is likely that no records will be correctly evaluated against the query and therefore no results will be returned.

You should also be careful of dollar signs or other similar characters in property or dimension values that would prevent a property or dimension from being defined as numeric.

Performance impact for range filters

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

Because range filters are not indexed, 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 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.

Chapter 8

Record Boost and Bury

This chapter describes the Record Boost and Bury feature.

About the record boost and bury feature

Record boost and bury is a mechanism by which the ranking of certain specific records is made much higher or lower than other records.

Record boost is a mechanism by which certain specific records are ranked highly relative to others. ***Record bury*** is the opposite, that is, certain specific records are ranked much lower relative to others. This mechanism therefore lets you manipulate ranking of results in order to push certain types of records to the top or bottom of the results list.

The feature depends on the use of the `stratify` relevance ranking module.

Note: The record boost and bury feature and the `stratify` relevance ranking module are not supported by the Aggregated MDEX Engine (Agraph).

Feature assumptions and limitations

The following applies to the record boost and bury feature:

- EQL (Endeca Query Language) is the language to use for defining which records are to be boosted or buried.
- Using an EQL statement, you can specify a set of records to be returned at the top of the results list.
- Using an EQL statement, you can specify a set of records to be returned at the bottom of the results list.
- Record boost and bury functionality is available even when no record search is performed.
- Record boost and bury is not supported by the Agraph.
- Record boost and bury is supported by the Java and .NET versions of the Presentation API, as well as the MDEX API through XQuery (MAX).

Some use-case assumptions are:

- This feature is expected to be used predominately with the Endeca Workbench and Page Builder products.

- A common usage pattern will be to specify the records to be boosted/buried dynamically (per-query). Typically, this will be done through Merchandising Workbench/Publishing Workbench and Page Builder, where a second query will be performed when boost/bury is used.
- Typical expectation is that only a handful of records will be boosted, that is, less than a page worth.
- The number of records buried may be higher, but ordering within this group is less important.
- If implemented for aggregated records, it is the base record ordering which will be affected by boost/bury.
- A record will be stratified in the highest strata it matches, so boosting will have priority over burying.

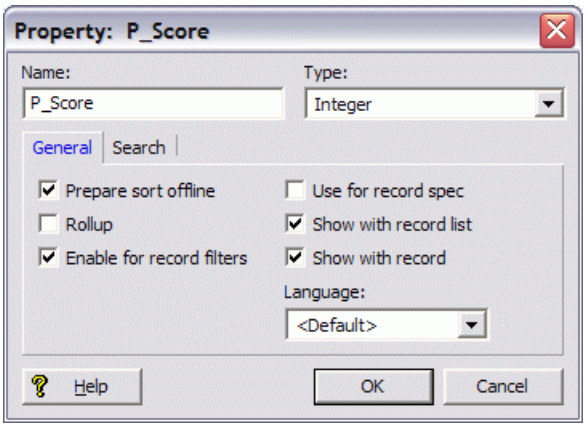
Enabling properties for filtering

Endeca properties must be explicitly enabled for use in record boost/bury filters.

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

To enable a property for use with record boost/bury filters:

1. In Developer Studio, open the Properties view.
2. Double-click on the Endeca property that you want to configure.
   The property is opened in the Property editor.
3. Check the **Enable for record filters** option, as in the following example.



4. Click **OK** to save your changes.

The stratify relevance ranking module

The `stratify` relevance ranking module is used to boost or bury records in the result set.

The `stratify` relevance ranking module ranks records by stratifying them into groups defined by EQL expressions. The module can be used:

- in record search options, via the `Ntx` URL query parameter or the `ERecSearch` class.
- as a component of a sort specification given as the default sort or in the API via the `Ns` URL query parameter or the `ENEQuery.setNavActiveSortKeys()` method.

The `stratify` module takes an ordered list of one or more EQL expressions that are used for boosting/burying records. The following example shows one EQL expression for the module:

```
N=0&Ntx=mode+matchall+rel+stratify(collection()/record[Score>95],*)&Ntk=Wine¬
Type&Ntt=merlot
```

This record search example queries for the term `merlot` in WineType values. Any record that has a Score value of greater than 95 will be boosted in relation to other records.

Note: When used for sort operations, you must prepend the `Endeca` prefix to the `stratify` module name for use in the sort specification (i.e., use `Endeca.stratify` as the name).

EQL expressions and record strata

Each EQL expression used in the `stratify` statement corresponds to a stratum, as does the set of records which do not match any expression, producing *k + 1* strata (where *k* is the number of EQL expressions). Records are placed in the stratum associated with the first EQL expression they match. The first stratum is the highest ranked, the next stratum is next-highest ranked, and so forth. Note a record will be stratified in the highest strata it matches, so boosting will have priority over burying.

If a record matches none of the specified EQL expressions, it is assigned to the *unmatched* stratum. By default, the unmatched stratum is ranked below all strata. However, you can change the rank of the unmatched stratum by specifying an asterisk (*) in the list of EQL expressions. In this case, the asterisk stands for the unmatched stratum.

The rules for using an asterisk to specify the unmatched stratum are:

- If an asterisk is specified instead of an EQL expression, unmatched records are placed in the stratum that corresponds to the asterisk.
- If no asterisk is specified, unmatched records are placed in a stratum lower than any expression's stratum.
- Only one asterisk can be used. If more than one asterisk is specified, the first one will be used and the rest ignored.

This `Ntx` snippet shows the use of an asterisk in the query:

```
N=0&Ntx=rel+stratify(collection()/record[Score>90],*,collec¬
tion()/record[Score<50])
```

The query will produce three strata of records:

- The highest-ranked stratum will be records whose Score value is greater than 90.
- The lowest-ranked stratum will be records whose Score value is less than 50.
- All other records will be placed in the unmatched stratum (indicated by the asterisk), which is the middle-ranked stratum.

Note that the EQL expressions must be URL-encoded. For example, this query:

```
collection()/record[status = 4]
```

should be issued in this URL-encoded format:

```
collection%28%29/record%5Bstatus%20%3D%204%5D
```

However, the examples in this chapter are not URL-encoded, in order to make them easier to understand.

Record boost/bury queries

Record queries can use the `stratify` relevance ranking module for boosting or burying records.

The `stratify` relevance ranking module can be specified in record search options, via the `Ntx` URL query parameter or the `ERecSearch` class.

Using the Ntx URL parameter

For record searches, the format for using the `Ntx` URL parameter with the `rel` option to specify the `stratify` relevance ranking module is:

```
Ntx=rel+stratify(EQLexpressions)
```

where *EQLexpressions* is one or more of the EQL expressions documented in the "Using the Endeca Query Language" chapter in the *Advanced Development Guide*.

This example uses an EQL property value query with the `and` operator:

```
N=0&Ntx=mode+matchall+rel+stratify(collection()/record[P_Region="Tuscany"
and P_Score>98],*)
&Ntk=P_WineType&Ntt=red
```

The results will boost red wine records that are from Tuscany and have a rating score of 98 or greater. These records are placed in the highest stratum and all other records are placed in the unmatched stratum.

Using the ERecSearch class

You can use the three-argument version of the `ERecSearch` constructor to create a record search query. The third argument can specify the use of the `stratify` module. The `ERecSearch` class is available in both the Java and .NET versions of the Presentation API.

The following example illustrates how to construct such a query using Java:

```
// Create query
ENEQuery usq = new UrlENEQuery(request.getQueryString(), "UTF-8");

// Create a record search query for red wines in the P_WineType property
// and boost records from the Tuscany region
String key = "P_WineType";
String term = "red";
String opt = "Ntx=rel+stratify(collection()/record[P_Region="Tuscany"],*)";
// Use the 3-argument version of the ERecSearch constructor
ERecSearch eSearch = new ERecSearch(key, term, opt);
// Add the search to the ENEQuery
ERecSearchList eList = new ERecSearchList();

eList.add(0, eSearch);
usq.setNavERecSearches(eList);
...
// Make ENE request
ENEQueryResults qr = nec.query(usq);
```

Boost/bury sorting for Endeca records

The record boost and bury feature can used to sort record results for queries.

The `Endeca.stratify` relevance ranking module can be specified in record search options, via the `Ns` URL query parameter or the API methods.

Note: When used for sorting, you must prepend the `Endeca` prefix to the `stratify` module name.

Using the Ns URL parameter

The format for using the `Ns` URL parameter with the `rel` option to specify the `stratify` relevance ranking module is:

```
Ns=Endeca.stratify(EQLexpressions)
```

where *EQLexpressions* is one or more of the EQL expressions documented in the "Using the Endeca Query Language" chapter in the *Advanced Development Guide*. Note that you must prepend the `Endeca` prefix to the module name.

For example, assume you wanted to promote Spanish wines. This `N=0` root node query returns all the records, with the Spanish wines boosted into the first stratum (i.e., they are displayed first to the user):

```
N=0&Ns=Endeca.stratify(collection()/record[P_Region="Spain"],*)
```

And if you wanted to boost your highly-rated Spanish wines, the query would look like this:

```
N=0&Ns=Endeca.stratify(collection()/record[P_Region="Spain" and
P_Score>90],*)
```

The query results will boost Spanish wines that have a rating score of 91 or greater. These records are placed in the highest stratum and all other records are placed in the unmatched stratum.

Using API methods

You can use the single-argument version of the `ERecSortKey` constructor to create a new relevance rank key that specifies the `Endeca.stratify` module. After adding the `ERecSortKey` object to an `ERecSortKeyList`, you can set it in the query with the Java `ENEQuery.setNavActiveSortKeys()` and the .NET `ENEQuery.SetNavActiveSortKeys` methods in the Presentation API.

The following Java sample code shows now to use these methods:

```
String stratKey = "Endeca.stratify(collection()/record[P_Region="Spain"],*)";
ERecSortKey stratSort = new ERecSortKey(stratKey);
ERecSortKeyList stratList = new ERecSortKeyList();
stratList.add(0, stratSort);
usq.setNavActiveSortKeys(stratList);
```

Chapter 9

Creating Aggregated Records

This section discusses the creation and use of aggregated records.

About aggregated records

The Endeca aggregated records feature allows the end user to group records by dimension or property values.

By configuring aggregated records, you enable the MDEX Engine to handle a group of multiple records as though it were a single record, based on the value of the rollup key. A rollup key can be any property or dimension that has its rollup attribute enabled.

Aggregated records are typically used to eliminate duplicate display entries. For example, an album by the same title may exist in several formats, with different prices. Each title is represented in the MDEX Engine as a distinct Endeca record. When querying the MDEX Engine, you may want to treat these instances as a single record. This is accomplished by creating an Endeca aggregated record.

From a performance perspective, aggregated Endeca records are not an expensive feature. However, they should only be used when necessary, because they add organization and implementation complexity to the application (particularly if the rollup key is different from the display information).

Enabling record aggregation

You enable aggregate Endeca record creation by allowing record rollups based on properties and dimensions.

Proper configuration of this feature requires that the rollup key is a single assign value. That is, each record should have at most one value from this dimension or property. If the value is not single assign, the first (arbitrarily-chosen) value is used to create the aggregated record. This can cause the results to vary arbitrarily, depending upon the navigation state of the user. In addition, features such as sort can change the grouping of aggregated records that are assigned multiple values of the rollup key.

To enable a property or dimension for record rollup:

1. In Developer Studio, open the target property or dimension.
2. Enable the rollup feature as follows:
   - For properties, check the **Rollup** checkbox in the General tab.
   - For dimensions, check the **Enable for rollup** checkbox in the Advanced tab.

3. Click **OK** to save the change.

Generating and displaying aggregated records

This section provides detailed information on creating and displaying aggregated records.

The general procedure of generating and displaying aggregated records is as follows:

1. Determine which rollup keys are available to be used for an aggregated record navigation query.
2. Create an aggregated record navigation query by using one of the available rollup keys. This rollup key is called the *active* rollup key, while all the other rollup keys are inactive.
3. Retrieve the list of aggregated records from the `Navigation` object and display their attributes.

These steps are discussed in detail in the following topics.

Determining the available rollup keys

The Presentation API has methods and properties to retrieve rollup keys.

Assuming that you have a navigation state, the following objects and calls are used to determine the available rollup keys. These rollup keys can be used in subsequent queries to generate aggregated records:

- The `Navigation.getRollupKeys()` method (Java) and `Navigation.RollupKeys` property (.NET) get the rollup keys applicable for this navigation query. The rollup keys are returned as an `ERecRollupKeyList` object.
- The `ERecRollupKeyList.size()` method (Java) and `ERecRollupKeyList.Count` property (.NET) get the number of rollup keys in the `ERecRollupKeyList` object.
- The `ERecRollupKeyList.getKey()` method (Java) and `ERecRollupKeyList.Item` property (.NET) get the rollup key from the `ERecRollupKeyList` object, using a zero-based index. The rollup key is returned as an `ERecRollupKey` object.
- The `ERecRollupKey.getName()` method (Java) and `ERecRollupKey.Name` property get the name of the rollup key.
- The `ERecRollupKey.isActive()` method (Java) and the `ERecRollupKey.IsActive()` method (.NET) return `true` if this rollup key was applied in the navigation query or `false` if it was not.

The rollup keys are retrieved from the `Navigation` object in an `ERecRollupKeyList` object. Each `ERecRollupKey` in this list contains the name and active status of the rollup key:

- The name is used to specify the rollup key in a subsequent navigation or aggregated record query.
- The active status indicates whether the rollup key was applied to the current query.

The following code fragments show how to retrieve a list of rollup keys, iterate over them, and display the names of keys that are active in the current navigation state.

Java example for getting rollup keys

```
// Get rollup keys from the Navigation object
ERecRollupKeyList rllupKeys = nav.getRollupKeys();
// Loop through rollup keys
for (int i=0; i< rllupKeys.size(); i++) {
  // Get a rollup key from the list
  ERecRollupKey rllupKey = rllupKeys.getKey(i);
  // Display the key name if the key is active.
```

```
  if (rllupKey.isActive()) {
    %>Active rollup key: <%= rllupKey.getName() %><%
  }
}
```

.NET example for getting rollup keys

```
// Get rollup keys from the Navigation object
ERecRollupKeyList rllupKeys = nav.RollupKeys;
// Loop through rollup keys
for (int i=0; i< rllupKeys.Count; i++) {
  // Get a rollup key from the list
  ERecRollupKey rllupKey = (ERecRollupKey)rllupKeys[i];
  // Display the key name if the key is active.
  if (rllupKey.IsActive()) {
    %>Active rollup key: <%= rllupKey.Name %><%
  }
}
```

Creating aggregated record navigation queries

You can generate aggregated records with URL query parameters or with Presentation API methods.

Note that regardless of how many properties or dimensions you have enabled as rollup keys, you can specify a maximum of one rollup key per navigation query.

Specifying the rollup key for the navigation query

To generate aggregated Endeca records, the query must be appended with an `Nu` parameter. The value of the `Nu` parameter specifies a rollup key for the returned aggregated records, using the following syntax:

```
Nu=rollupkey
```

For example:

```
N=0&Nu=Winery
```

The records associated with the navigation query are grouped with respect to the rollup key prior to computing the subset specified by the `Nao` parameter (that is, if `Nu` is specified, `Nao` applies to the aggregated records rather than individual records). Aggregated records only apply to a navigation query. Therefore, the `Nu` query parameter is only valid with an `N` parameter.

The equivalent API method to the `Nu` parameter is:

- Java: the `ENEQuery.setNavRollupKey()` method
- .NET: the `ENEQuery.NavRollupKey` property

Examples of these calls are:

```
// Java version
usq.setNavRollupKey("Winery");

// .NET version
usq.NavRollupKey("Winery");
```

When the aggregated record navigation query is made, the returned `Navigation` object which will contain an `AggrERecList` object.

Setting the maximum number of returned records

You can use the Np parameter to control the maximum number of Endeca records returned in any aggregated record. Set the parameter to 0 (zero) for no records, 1 for one record, or 2 for all records. For example:

```
N=0&Np=2&Nu=Winery
```

The equivalent API method to the Np parameter is:

- Java: the ENEQuery.setNavERecsPerAggrERec() method
- .NET: the ENEQuery.NavERecsPerAggrERec property

Creating aggregated record queries

You can create aggregated record queries with URL query parameters or with Presentation API methods.

An aggregated record request is similar to an ordinary record request with these exceptions:

- If you are using URL query parameters, the A parameter is specified (instead of R). The value of the A parameter is the record specifier of the aggregated record.
- If you are using the API, use the ENEQuery.setAggrERecSpec() method (Java) or the ENEQuery.AggrERecSpec property (.NET) to specify the aggregated record to be queried for.
- The element returned is an aggregated record (not a record).

You can use the As parameter to specify a sort that determines the order of the representative records. You can specify one or more sort keys with the As parameter. A sort key is a dimension or property name enabled for sorting on the data set. Optionally, each sort key can specify a sort order of 0 (ascending sort, the default) or 1 (descending sort). The As parameter is especially useful if you want to use the record boost and bury feature with aggregated records.

Similar to an ordinary record, An (instead of N) is the user's navigation state. Only records that satisfy this navigation state are included in the aggregated record. In addition, the Au parameter must be used to specify the aggregated record rollup key.

The following are two examples of queries using the An parameter:

```
An=0&A=32905&Au=Winery&As=Score

A=7&An=123&Au=ssn
```

For the API, the examples below show how the UrlGen class constructs the URL query string. Note the following in the examples:

- The ENEQuery.setAggrERecSpec() method (Java) and the ENEQuery.AggrERecSpec property (.NET) provide the aggregated record specifier to the A parameter.
- The ENEQuery.getNavDescriptors() method (Java) and the ENEQuery.NavDescriptors property (.NET) get the navigation values for the An parameter.
- The ENEQuery.getNavRollupKey() method (Java) and the ENEQuery.NavRollupKey property (.NET) get the name of the rollup key for the Au parameter.

Java example

```
// Create aggregated record request (start from empty request)
UrlGen urlg = new UrlGen("", "UTF-8");
urlg.addParam("A",aggrec.getSpec());
urlg.addParam("An",usq.getNavDescriptors().toString());
urlg.addParam("Au",usq.getNavRollupKey());
urlg.addParam("eneHost",(String)request.getAttribute("eneHost"));
```

```
urlg.addParam("enePort",(String)request.getAttribute("enePort"));
urlg.addParam("displayKey",String)request.getParameter("displayKey"));
urlg.addParam("sid",(String)request.getAttribute("sid"));
String url = CONTROLLER+"?"+urlg;
%><a href="<%= url %>">%>
```

.NET example

```
// Create aggregated record request (start from empty request)
urlg = new UrlGen("", "UTF-8");
urlg.AddParam("A", aggrec.Spec);
urlg.AddParam("An",usq.NavDescriptors.ToString());
urlg.AddParam("Au",usq.NavRollupKey);
urlg.AddParam("eneHost",(String)Request.QueryString["eneHost"]);
urlg.AddParam("enePort",(String)Request.QueryString["enePort"]);
urlg.AddParam("displayKey",(String)Request.QueryString["displayKey"]);
urlg.RemoveParam("sid");
urlg.AddParam("sid",(String)Request.QueryString["sid"]);
url = (String) Application["CONTROLLER"] + "?" + urlg.ToString();
%><a href="<%= url %>">%>
```

Getting aggregated records from record requests

The `ENEQueryResults` class has methods to retrieve aggregated record objects.

On an aggregated record request, the aggregated record is returned as an `AggrERec` object in the `ENEQueryResults` object. Use these calls:

- The `ENEQueryResults.containsAggrERec()` method (Java) and the `ENEQueryResults.ContainsAggrERec()` method (.NET) return `true` if the `ENEQueryResults` object contains an aggregated record.
- The `ENEQueryResults.getAggrERec()` method (Java) and the `ENEQueryResults.AggrERec` property (.NET) retrieve the `AggrERec` object from the `ENEQueryResults` object.

Java example

```
// Make MDEX Engine request
ENEQueryResults qr = nec.query(usq);
// Check for an AggrERec object in ENEQueryResults
if (qr.containsAggrERec()) {
  AggrERec aggRec = (AggrERec)qr.getAggrERec();
  ...
}
```

.NET example

```
// Make MDEX Engine request
ENEQueryResults qr = nec.Query(usq);
// Check for an AggrERec object in ENEQueryResults
if (qr.ContainsAggrERec()) {
  AggrERec aggRec = (AggrERec)qr.AggrERec;
  ...
}
```

Retrieving aggregated record lists from Navigation objects

The `Navigation` class calls can retrieve aggregated records.

On an aggregated record navigation query, a list of aggregated records (an `AggrERecList` object) is returned in the `Navigation` object.

To retrieve a list of aggregated records returned by the navigation query, as an `AggrERecList` object, use:

- Java: the `Navigation.getAggrERecs()` method
- .NET: the `Navigation.AggrERecs` property

To get the number of aggregated records that matched the navigation query, use:

- Java: the `Navigation.getTotalNumAggrERecs()` method
- .NET: the `Navigation.TotalNumAggrERecs` property

Note that by default, the MDEX Engine returns a maximum of 10 aggregated records. To change this number, use:

- Java: the `ENEQuery.setNavNumAggrERecs()` method
- .NET: the `ENEQuery.NavNumAggrERecs` property

Displaying aggregated record attributes

The `AggrERec` class calls can retrieve attributes of aggregated records.

After you retrieve an aggregated record, you can use the following `AggrERec` class calls:

- The `getERecs()` method (Java) and `ERecs` property (.NET) gets the Endeca records (`ERec` objects) that are in this aggregated record.
- The `getProperties()` method (Java) and `Properties` property (.NET) return the properties (as a `PropertyMap` object) of the aggregated record.
- The `getRepresentative()` method (Java) and `Representative` property (.NET) get the Endeca record (`ERec` object) that is the representative record of this aggregated record.
- The `getSpec()` method (Java) and `Spec` property (.NET) get the specifier of the aggregated record to be queried for.
- The `getTotalNumERecs()` method (Java) and `TotalNumERecs` property (.NET) return the number of Endeca records (`ERec` objects) that are in this aggregated record.

The following code snippets illustrate these calls.

Java example

```
Navigation nav = qr.getNavigation();
// Get total number of aggregated records that matched the query
long nAggrRecs = nav.getTotalNumAggrERecs();
// Get the aggregated records from the Navigation object
AggrERecList aggrecs = nav.getAggrERecs();
// Loop over the aggregated record list
for (int i=0; i<aggrecs.size(); i++) {
  // Get individual aggregate record
  AggrERec aggrec = (AggrERec)aggrecs.get(i);
  // Get number of records in this aggregated record
  long recCount = aggrec.getTotalNumERecs();
  // Get the aggregated record's attributes
  String aggrSpec = aggrec.getSpec();
  PropertyMap propMap = aggrec.getProperties();
  ERecList recs = aggrec.getERecs();
  ERec repRec = aggrec.getRepresentative();
}
```

.NET example

```
Navigation nav = qr.Navigation;
// Get total number of aggregated records that matched the query
long nAggrRecs = nav.TotalNumAggrERecs;
// Get the aggregated records from the Navigation object
AggrERecList aggrecs = nav.AggrERecs;
// Loop over the aggregated record list
for (int i=0; i<aggrecs.Count; i++) {
  // Get individual aggregate record
  AggrERec aggrec = (AggrERec)aggrecs[i];
  // Get number of records in this aggregated record
  long recCount = aggrec.TotalNumERecs;
  // Get the aggregated record's attributes
  String aggrSpec = aggrec.Spec;
  PropertyMap propMap = aggrec.Properties;
  ERecList recs = aggRec.ERecs;
  ERec repRec = aggrec.Representative;
}
```

Displaying refinement counts for aggregated records

The `Dgraph.AggrBins` property contains aggregated record statistics.

To enable dynamic statistics (aggregated record counts beneath a given refinement), use the `--statabins` flag with the Dgraph.

Statistics on aggregated records are returned as a property on each dimension value. For aggregated records, this property is `DGraph.AggrBins`. In other words, to retrieve the aggregated record counts beneath a given refinement, use the `DGraph.AggrBins` property.

The following code examples show how to retrieve the dynamic statistics for aggregated records.

Java example

```
DimValList dvl = dimension.getRefinements();
for (int i=0; i < dvl.size(); i++) {
  DimVal ref = dvl.getDimValue(i);
  PropertyMap pmap = ref.getProperties();
  // Get dynamic stats
  String dstats = "";
  if (pmap.get("DGraph.AggrBins") != null) {
    dstats = " ("+pmap.get("DGraph.AggrBins")+")";
  }
}
```

.NET example

```
DimValList dvl = dimension.Refinements;
for (int i=0; i < dvl.Count; i++) {
  DimVal ref1 = (DimVal)dvl[i];
  PropertyMap pmap = ref.Properties;
  // Get dynamic stats
  String dstats = "";
  if (pmap["DGraph.AggrBins"] != null) {
  dstats = " ("+pmap["DGraph.AggrBins"]+")";
  }
}
```

Displaying the records in the aggregated record

A record in an aggregated record can be displayed like any other Endeca record.

You display the Endeca records (`ERec` objects) in an aggregated record with the same procedures described in Chapter 5 ("Working with Endeca Records").

In the following examples, a list of aggregated records is retrieved from the `Navigation` object and the properties of each representative record are displayed.

Java example

```
Get aggregated record list from the Navigation object
AggrERecList aggrecs = nav.getAggrERecs();
// Loop over aggregated record list
for (int i=0; i<aggrecs.size(); i++) {
  // Get an individual aggregated record
  AggrERec aggrec = (AggrERec)aggrecs.get(i);
  // Get representative record of this aggregated record
  ERec repRec = aggrec.getRepresentative();
  // Get property map for representative record
  PropertyMap repPropsMap = repRec.getProperties();
  // Get property iterator to loop over the property map
  Iterator repProps = repPropsMap.entrySet().iterator();
  // Display representative record properties
  while (repProps.hasNext()) {
    // Get a property
    Property prop = (Property)repProps.next();
    // Display name and value of the property
    %>
    <tr>
    <td>Property name: <%= prop.getKey() %></td>
    <td>Property value: <%= prop.getValue() %>
    </tr>
    <%
  }
}
```

.NET example

```
/ Get aggregated record list from the Navigation object
AggrERecList aggrecs = nav.AggrERecs;
// Loop over aggregated record list
for (int i=0; i<aggrecs.Count; i++) {
  // Get an individual aggregated record
  AggrERec aggrec = (AggrERec)aggrecs[i];
  // Get representative record of this aggregated record
  ERec repRec = aggrec.Representative;
  // Get property map for representative record
  PropertyMap repPropsMap = repRec.Properties;
  // Get property list for representative record
  System.Collections.Ilist repPropsList = repPropsMap.EntrySet;
  // Display representative record properties
  foreach (Property repProp in repPropsList) {
    %>
    <tr>
    <td>Property name: <%= repProp.Key %></td>
    <td>Property value: <%= repProp.Value %>
    </tr>
    <%
```

```
  }
}
```

Related Links

Working with Endeca Records on page 53
This section provides information on handling Endeca records in your Web application.

Aggregated record behavior

Aggregated records behave differently than ordinary records.

Programmatically, an ordinary record is an `ERec` object while an aggregated record is an `AggrERec` object.

Two of the major differences between the two types of records are in their representative values and sorting behavior:

- Representative values – Given a single record, evaluating the record's information is straightforward. However, aggregated records consist of many records, which can have different representative values. Generally for display and other logic requiring record values, a single representative record from the aggregated record is used. The representative record is the individual record that occurs first in order of the underlying records in the aggregated record. This order is determined by either a specified sort key or a relevance ranking strategy.
- Sort – The sort feature is first applied to all records in the data set (prior to aggregating the records). The record at the top of this set is the record with the highest sort value. Given the sorted set of records, aggregated records are created by iterating over the set in descending order, aggregating records with the same rollup key. An aggregated record's rank is equal to that of the highest ranking record in that aggregated record set. The result is the same as aggregating all records on the rollup key, taking the highest value of the sort key for these aggregated records and sorting the set based on this value.

Note: If you have a defined list of sort keys, the first key is the primary sort criterion, the second key is the secondary sort criterion, and so on.

The presentation developer has more power over retrieving the representative values. The individual records are returned with the aggregated record. Therefore, the developer has all the information necessary to correctly represent aggregated records (at the cost of increased complexity). However, to achieve the desired sort behavior, the MDEX Engine must be configured correctly, because the internals of this operation are not exposed to the presentation developer.

Refinement ranking of aggregated records

The MDEX Engine uses the aggregated record counts beneath a given refinement for its refinement ranking strategy only if they were computed for the query sent to the MDEX Engine.

The MDEX Engine computes refinement ranking based on statistics for the number of records beneath a given refinement. In the case of aggregated records, refinement ranking depends on whether you have requested the MDEX Engine to compute statistics for aggregated record counts beneath a given refinement.

The following statements describe the behavior:

- To enable dynamic statistics for aggregated records (aggregated record counts beneath a given refinement), use the `--stat-abins` flag with the Dgraph.
- To retrieve the aggregated record counts beneath a given refinement, use the `DGraph.AggrBins` property.
- If you specify `--stat-abins` when starting a Dgraph and issue an aggregated query to the MDEX Engine, it then computes counts for aggregated records beneath a given refinement, and generates refinement ranking based on statistics computed for aggregated records.
- If you specify `--stat-abins` and issue a non-aggregated query to the MDEX Engine, it only computes counts for regular records (instead of aggregated record counts) beneath a given refinement, and generates refinement ranking based on statistics computed for regular records.
- If you do not specify `--stat-abins` and issue an aggregated query to the MDEX Engine, it only computes counts for regular records (instead of aggregated record counts) beneath a given refinement, and generates refinement ranking based on statistics computed for regular records.

To summarize, the MDEX Engine uses the aggregated record counts beneath a given refinement for its refinement ranking strategy only if they were computed. In all other cases, it uses only regular record counts for refinement ranking.

Part 3

Dimension and Property Features

Chapter 10

Property Types

You can assign the following types of properties to records in the MDEX Engine: Alpha, Integer, Floating point, Geocode, DateTime, Duration and Time. You assign property types in Developer Studio.

Formats used for property types

The MDEX Engine supports property types that use the following accepted formats:

Property type	Description
Alpha	Represents character strings.
Integer	Represents a 32-bit signed integer. Integer values accepted by the MDEX Engine on all platforms can be up to the value of 2147483647.
Floating point	Represents a floating point.
Geocode	Represents a latitude and longitude pair used for geospatial filtering and sorting. Each value is a double-precision floating-point value. The two values are comma-delimited. The accepted format is: `latvalue,lonvalue`, where: • `latvalue` is the latitude of the location in whole and fractional degrees. Positive values indicate north latitude and negative values indicate south latitude. • `lonvalue` is the longitude of the location in whole and fractional degrees. Positive values indicate east longitude, and negative values indicate west longitude. For example, to indicate the Location geocode property located at 42.365615 north latitude, 71.075647 west longitude, specify: `42.365615,-71.075647`
DateTime	A 64-bit signed integer that represents the date and time in milliseconds since the epoch (January 1, 1970).
Duration	A 64-bit signed integer that represents a length of time in milliseconds.
Time	A 32-bit unsigned integer that represents the time of day in milliseconds.

Temporal properties

This section describes temporal property types supported in the MDEX Engine — Time, DateTime and Duration.

Defining Time and DateTime properties

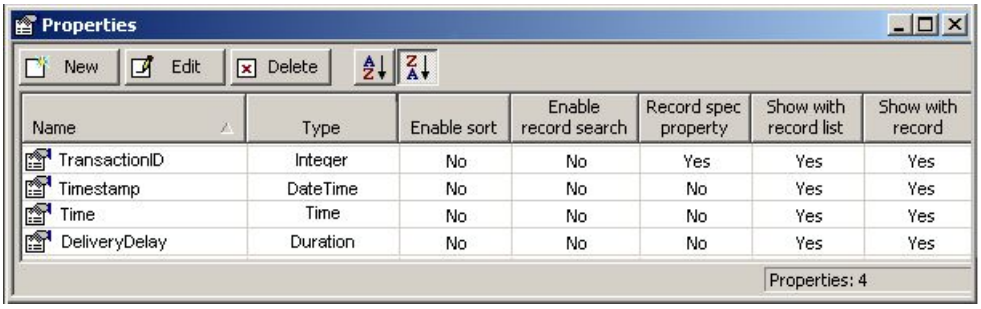
Time, DateTime and Duration properties are supported in the MDEX Engine. You define them in Developer Studio.

Note: The DateTime property is available in Developer Studio by default and does not require additional configuration. However, Time and Duration property types are only enabled if you configure Developer Studio for the use of Time and Duration Property Types" in the *Endeca Developer Studio Installation Guide*. The use of these property types also requires enabling Endeca Analytics. For information, see the *Enabling Endeca Analytics* guide.

The Property editor provides three temporal property types:

- Time values represent a time of the day
- DateTime values represent a time of the day on a given date
- Duration values represent a length of time

In the example below, the Time property has been declared to be of the Time type, the TimeStamp property has been declared to be of the DateTime type, and the DeliveryDelay property has been declared to be of the Duration type:



Properties of type Time, DateTime, and Duration can be used for:

- Temporal sorting using the record sort feature of the MDEX Engine
- The ORDER BY operator of the Analytics API
- Time-based filtering using the range filter feature of the MDEX Engine
- The WHERE and HAVING operators in the Analytics API
- As inputs to time-specific operators in the Analytics API (TRUNC and EXTRACT)

For information about temporal properties in Analytics queries, and time-specific operators in the Analytics API, see the *Analytics Guide*.

Time properties

Time properties represent the time of day to a resolution of milliseconds.

A string value in a Time property, both on input to the MDEX Engine and when accessed through the Analytics API, should contain an integer representing the number of milliseconds since the start of day, `midnight/12:00:00AM`. Time properties are stored as 32-bit integers.

For example, `1:00PM` or `13:00` would be represented as `46800000` because:

```
13 hours *
60 minutes / hour *
60 seconds / minute *
1000 milliseconds / second = 46800000
```

DateTime properties

DateTime properties represent the date and time to a resolution of milliseconds.

A string value in a DateTime property should contain an integer representing the number of milliseconds since the epoch (January 1, 1970). Additionally, values must be in Coordinated Universal Time (UTC) and account for the number of milliseconds since the epoch, in conformance with POSIX standards. DateTime values are stored as 64-bit integers.

For example, `August 26, 2004 1:00PM` would be represented as `1093525200000` because:

```
12656 days *
24 hours / day *
60 minutes / hour *
60 seconds / minute *
1000 milliseconds / second +
46800000 milliseconds (13 hrs) = 1093525200000
```

Duration properties

Duration properties represent lengths of time with a resolution of milliseconds.

A string value in a Duration property should contain an integer number of milliseconds. Duration values are stored as 64-bit integers.

For example, `100 days` would be represented as `8640000000` because:

```
100days *
24 hours / day *
60 minutes / hour *
60 seconds / minute *
1000 milliseconds / second = 8640000000
```

Working with time and date properties

Like all Endeca property types (Alpha, Floating Point, Integer, and so on), time and date values are handled during the data ingest process and in UI application code as strings, but are stored and manipulated as typed data in the Endeca MDEX Engine.

For non-Alpha property types, this raises the question of data manipulation in the Forge pipeline and appropriate presentation of typed data in the UI.

At data ingest time, inbound temporal data is unlikely to conform to the representations required by Endeca temporal property types. But time and date classes for performing needed conversions are readily available in the standard Java library (see `java.text.DateFormat`). These should be used (in the context of a JavaManipulator Forge component) to convert inbound data in the data ingest pipeline.

For example, the following code performs simple input conversion on source date strings of the form "August 26, 2009" to Endeca DateTime property format:

```
String sourceDate = … // String of form "August 26, 2009"
DateFormat dateFmt = DateFormat.getDateInstance(DateFormat.LONG);
Date date = dateFmt.parse(sourceDate);
Long dateLong = new Long(date.getTime());
String dateDateTimeValue = dateLong.toString();
```

Similarly, in most cases the integer representation of times and dates supported by the Endeca MDEX Engine is not suitable for application display. Again, the application should make use of standard library components (such as `java.util.Date` and `java.util.GregorianCalendar`) to convert Endeca dates for presentation.

For example, the following code performs a simple conversion of a DateTime value to a pretty-printable string:

```
String dateStr = … // Initialized to an Endeca DateTime value
long dateLong = Long.parseLong(dateStr);
Date date = new Date(dateLong);
String dateRenderString = date.toString();
```

Chapter 11

Working with Dimensions

This section provides information on handling and displaying Endeca dimensions in your Web application.

Displaying dimension groups

Dimensions are part of dimension groups and both the group and its dimensions can be displayed.

Dimension groups provide a way to impose relationships on dimensions. By creating a dimension group, you can organize dimensions for presentation purposes. Each explicit dimension group must be given a name; a unique ID is generated when the data is indexed.

Each dimension can belong to only a single dimension group. If you do not assign a dimension to an explicit dimension group, it is placed in an implicit dimension group of its own. These implicit groups have no name and an ID of zero. For example, if your project has ten dimensions and no explicit group is set, the project contains ten different groups with no names and with IDs of zero.

You use Developer Studio's Dimension Group editor to create dimension groups, and its Dimension editor to assign dimensions to groups. For details on these tasks, see the Developer Studio online help.

No Dgidx or Dgraph flags are necessary to enable dimension groups. In addition, no MDEX Engine URL parameters are required to access dimension group information.

Dimension group API methods

The Navigation and DimGroup classes have methods to access information about dimension groups.

The dimensions in a dimension group are encapsulated in a `DimGroup` object. In turn, a `DimGroupList` object contains a list of dimension groups (`DimGroup` objects).

The next two sections show how to access the `Navigation` and `DimGroupList` objects for dimension group information. The code samples show how to loop over a `DimGroupList` object, access each dimension group in the object, and get each group's name and ID.

Accessing the Navigation object

There are three calls on the `Navigation` object that access the `DimGroupList` object. All three return a `DimGroupList` object that contains group names, group IDs, and the child dimensions:

API method or property	Purpose
Java: `Navigation.getDescriptorDimGroups()` .NET: `Navigation.DescriptorDimGroups`	Gets an object that has information about the dimension groups for the dimensions with descriptors in the current navigation state.
Java: `Navigation.getRefinementDimGroups()` .NET: `Navigation.RefinementDimGroups`	Gets an object that contains the dimensions with refinements available in the current navigation state.
Java: `Navigation.getIntegratedDimGroups()` .NET: `Navigation.IntegratedDimGroups`	Gets an object that contains all of the information contained in the above two calls.

Accessing the DimGroupList object

Once the application has the `DimGroupList` object, it can render the dimension group information with these methods and properties:

API method or property	Purpose
Java: `DimGroupList.size()` .NET: `DimGroupList.Count`	Used on the `DimGroupList` object to initiate a loop over all the dimension groups, implicit and explicit. Once this loop is initiated, a `DimGroup` object is created.
Java: `DimGroup.getId()` .NET: `DimGroup.Id`	With these calls, the application is able to assess whether the current group is implicit (having an ID of zero) or explicit (having an ID greater than zero).
Java: `DimGroup.getName()` .NET: `DimGroup.Name`	Used to access the name of the current dimension group. If this returns a null object, then the current dimension group was implicitly created.
Java: `DimGroup.size()` .NET: `DimGroup.Count`	Used in initiating a loop in order to access the dimensions in the group.
Java: `DimGroup.getDimension()` .NET: `DimGroup.GetDimension`	Used to access a specific dimension in the group without looping. This method requires either a dimension ID or a dimension name to be passed in.

Java example of getting a dimension group ID and name

```
DimGroupList refDimGroups = nav.getRefinementDimGroups();
// Loop over the list of dimension groups
for (int i=0; i<refDimGroups.size(); i++) {
  // Get an individual dimension group
  DimGroup dg = (DimGroup)refDimGroups.get(i);
  long dimGroupId = dg.getId();
  // If ID is zero, group is implicit, otherwise get its name
  if (dimGroupId != 0) {
    String dimGroupName = dg.getName();
```

```
  }
  for (int j=0; j<dg.size(); j++) {
    // retrieve refinement dimension values
    ...
  }
}
```

.NET example of getting a dimension group ID and name

```
DimGroupList refDimGroups = nav.RefinementDimGroups;
// Loop over the list of dimension groups
for (int i=0; i<refDimGroups.Count; i++) {
  // Get individual dimension group
  DimGroup dg = (DimGroup)refDimGroups[i];
  long dimGroupId = dg.Id;
  // If ID is zero, group is implicit, otherwise get its name
  if (dimGroupId != 0) {
    String dimGroupName = dg.Name;
  }
  for (int j=0; j<dg.Count; j++) {
    // retrieve refinement dimension values
    ...
  }
}
```

Notes on displaying dimension groups

This section contains information that further explains how dimension group data is displayed.

Dimension groups versus dimension hierarchy

Dimension groups allow the user to select values from each of the dimensions contained in them. If the relationships made by a dimension group were instead created with hierarchy, once a value had been selected from one of the branches, then the remaining dimension values would no longer be valid for refinement.

For example, in mutual funds data, a user may want to navigate on a variety of performance criteria. A Performance dimension group that contains the YTD Total Returns, 1 Year Total Returns, and Five Year Total Returns dimensions would allow the user to select criteria from all three dimensions. If the same relationship had been created using dimension hierarchy, then once a selection had been made from the 1 Year Total Returns branch, the other two branches would no longer be available for navigation.

Ranking and dimension groups

The display order of dimension groups is determined by the ranking of the individual dimensions within the groups. A dimension group inherits the highest rank of its member dimensions. For example, if the highest-ranked dimension in dimension group A has a rank of 5, and the highest-ranked dimension in group B has a rank of 7, then group B will be ordered before group A.

Dimension groups are also ranked relative to dimensions not within explicit groups. Continuing the previous example, an implicit dimension with a rank of 6 would be ordered after dimension group B, but before group A.

Dimensions with the same rank are ordered by name. It is important to note that dimension name, not dimension group name, determines the display order in this situation: Dimension groups are ordered

according to their highest alphanumerically-ranked member dimensions. Therefore, dimension group Z, which contains dimension H, will be ordered before dimension group A, which contains dimension I.

For more information on ranking, see the Developer Studio online help.

Performance impact when displaying dimension groups

The use of dimension groups has minimal impact on performance.

Displaying refinements

Displaying dimensions and corresponding dimension values for query refinement is the core concept behind Guided Navigation.

After a user creates a query using record search and/or dimension values, only valid remaining dimension 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.

Configuring dimensions for query refinement

No dimension configuration is necessary for query refinement.

Assuming that a dimension is created in Developer Studio and that the dimension is used to classify records, the corresponding dimension values will be available to create or refine a query. The only exception is if a dimension is flagged as hidden in Developer Studio.

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

Dgidx flags for refinement dimensions

There are no Dgidx flags necessary to enable displaying refinement dimensions. If a dimension has been created and used to classify records, and has not been flagged as hidden, that dimension will automatically be indexed as a possible refinement dimension.

MDEX Engine flags

There are no MDEX Engine configuration flags necessary to enable the basic displaying of dimension refinements. However, there are some flags that control how and when these dimension refinements are displayed. These flags are documented in the appropriate feature sections (such as dynamic ranking).

URL parameters for dimension refinement values

Use the `Ne` parameter to expose refinement dimension values.

Refinement dimension values are only returned with a valid navigation query. Therefore the `N` (Navigation) parameter is required for any request that will render navigation refinements. The other parameter required in most cases to render navigation refinements is the `Ne` (Exposed Refinements) parameter.

The Ne parameter specifies which dimension, out of all valid dimensions returned with a Navigation query, should return actual refinement dimension values. Note that only the top-level refinement dimension values are returned. If a dimension value is a parent, you can also use the Ne parameter with that dimension value and return its child dimension values (again, only the top-level child dimension values are returned).

Keep in mind that the Ne parameter is an optional query parameter. The default query (where Ne is not used) is intended to improve computational performance of the MDEX Engine, as well as reduce the resulting object and final rendered page sizes.

For example, in a simple dataset, the query:

```
N=0
```

will return three dimensions (Wine Type, Year, and Score) but no refinement dimension values. This is faster for the MDEX Engine to compute, and returns only three root dimension values.

However, the query:

```
N=0&Ne=6
```

(where 6 is the root dimension value ID for the Wine Type dimension) will return all three dimensions, as well as the top-level refinement dimension values for the Wine Type dimension (such as Red, White, and Other). This is slightly more expensive for the MDEX Engine to compute, and returns the three root dimension values (Wine Type, Year, and Score) as well as the top-level refinement dimension values for Wine Type, but is necessary for selecting a valid refinement.

A more advanced query option does not require the Ne parameter and returns all the top-level dimension value refinements for all dimensions (instead of a single dimension). This option involves the use of the `ENEQuery.setNavAllRefinements()` method (Java) or the `ENEQuery.NavAllRefinements` property (.NET). If an application sets this call to true, the query:

```
N=0
```

will return three dimensions (Wine Type, Year, and Score) as well as all valid top-level refinement dimension values for each of these dimensions (Red, White, Other for Wine Type; 1999, 2001, 2003 for Year; and 70-80, 80-90, 90-100 for Score).

This is the equivalent of the query:

```
N=0&Ne=6+2+9
```

(where 6, 2, and 9 are the root dimension value IDs for the three dimensions). This is the most expensive type of query for the MDEX Engine to compute, and returns three root dimension values as well as the nine top-level refinement dimension values, creating a larger network and page size strain. This method, however, is effective for creating custom navigation solutions that require all possible refinement dimension values to be displayed at all times.

Retrieving refinement dimensions

The first step in displaying refinements is to retrieve the dimensions that potentially have refinements.

Types of refinements

Refinement dimensions contain refinement dimension values for the current record set, including both *standard refinements* and *implicit refinements*.

- Standard refinements (also called normal refinements) are refinements which, if selected, will refine the record set.

- Implicit refinements are refinements which, if selected, will not alter the navigation state record set. (The navigation state is the set of all dimension values selected in the current query context; the navigation state record set consists of the records selected by the navigation state.)

Descriptor dimensions contain the dimension values (or descriptors) that were used to query for the current record set. Integrated dimensions represent a consolidation of those dimensions that contain either descriptors or refinement values for the current record set.

Complete dimensions represent a consolidation of all dimensions that have at least one of the following: a descriptor, a standard refinement, or an implicit refinement.

Retrieving a list of dimensions or dimension groups

Accessing refinement dimension values for a given Navigation query begins with accessing the `Navigation` object from the query results object. Once an application has retrieved the `Navigation` object, there are a number of methods for accessing dimensions that contain dimension values.

The following calls access dimensions directly:

API method or property	Purpose
Java: `Navigation.getRefinementDimensions()` .NET: `Navigation.RefinementDimensions`	Returns a `DimensionList` object that has dimensions that potentially still have refinements available with respect to this query.
Java: `Navigation.getDescriptorDimensions()` .NET: `Navigation.DescriptorDimensions`	Returns a `DimensionList` object that has the dimensions for the descriptors for this navigation.
Java: `Navigation.getIntegratedDimensions()` .NET: `Navigation.IntegratedDimensions`	Returns a `DimensionList` object that has the dimensions integrated from the refinement dimensions and the descriptor dimensions.
Java: `Navigation.getCompleteDimensions()` .NET: `Navigation.CompleteDimensions`	Returns a `DimensionList` object that has the complete dimensions integrated from the refinement dimensions, the descriptor dimensions, and those that are completely implicit.

The following calls access dimension groups directly:

API method or property	Purpose
Java: `Navigation.getRefinementDimGroups()` .NET: `Navigation.RefinementDimGroups`	Returns a `DimGroupList` object that contains the dimensions that potentially have refinements available in the current navigation state.
Java: `Navigation.getDescriptorDimGroups()`	Returns a `DimGroupList` object that contains the dimension groups of the dimensions for the descriptors for this navigation.

API method or property	Purpose
.NET: `Navigation.DescriptorDimGroups`	
Java: `Navigation.getIntegratedDimGroups()` .NET: `Navigation.IntegratedDimGroups`	Returns a `DimGroupList` object that contains the dimension groups of the dimensions integrated from the refinement and descriptor dimensions.
Java: `Navigation.getCompleteDimGroups()` .NET: `Navigation.CompleteDimGroups`	Returns a `DimGroupList` object that contains the dimension groups of the complete dimensions integrated from the refinement dimensions, the descriptor dimensions, and those that are completely implicit.

Extracting refinement values

The Presentation API has methods to extract standard and implicit refinements from dimensions.

Extracting standard refinements from a dimension

Once a refinement dimension has been retrieved, these calls can extract various refinement information from the dimension:

API method or property	Purpose
Java: `Dimension.getName()` .NET: `Dimension.Name`	Retrieves the dimension name.
Java: `Dimension.getId()` .NET: `Dimension.Id`	Retrieves the dimension ID. This ID can then be used with the `Ne` query parameter to allow an application to expose refinements for this dimension.
Java: `Dimension.getRefinements()` .NET: `Dimension.Refinements`	Retrieves a list of refinement dimension values. This list will be empty unless the dimension has been specified by the `Ne` parameter or the `ENEQuery.setNavAllRefinements()` method (Java) or `ENEQuery.NavAllRefinements` property (.NET) has been set to true. If the dimension has been specified, however, and the refinements are exposed, this list will contain dimension values that can be used to create valid refined Navigation queries.

The following code samples show how to retrieve refinement dimension values from a navigation request where a dimension has been identified in the `Ne` parameter.

Java example of extracting standard refinements

```
Navigation nav = ENEQueryResults.getNavigation();
DimensionList dl = nav.getRefinementDimensions();
for (int I=0; I < dl.size(); I++) {
  Dimension d = (Dimension)dl.get(I);
  DimValList refs = d.getRefinements();
```

```
  for (int J=0; J < refs.size(); J++) {
    DimVal ref = (DimVal)refs.get(J);
    String name = ref.getName();
    Long id = ref.getId();
  }
}
```

.NET example of extracting standard refinements

```
Navigation nav = ENEQueryResults.Navigation;
DimensionList dl = nav.RefinementDimensions;
for (int I=0; I < dl.Count; I++) {
  Dimension d = (Dimension)dl[I];
  DimValList refs = d.Refinements;
  for (int J=0; J < refs.Count; J++) {
    DimVal ref = (DimVal)refs[J];
    String name = ref.Name;
    Long id = ref.Id;
  }
}
```

Extracting implicit refinements from a dimension

If a dimension contains implicit refinements, they can be extracted from the dimension with:

- Java: `Dimension.getImplicitLocations()` method
- .NET: `Dimension.ImplicitLocations` property

The call returns a `DimLocationList` object, which (if not empty) encapsulates `DimLocation` objects that contain the implicit dimension value (a `DimVal` object) and all of the dimension location's ancestors (also `DimVal` objects) up to, but not including, the dimension root.

You can also use these methods to test whether a dimension is fully implicit (that is, if the dimension has no non-implicit refinements and has no descriptors):

- Java: `Dimension.isImplicit()`
- .NET: `Dimension.IsImplicit()`

The following code samples show how to test if a dimension is fully implicit and, if so, how to retrieve the implicit refinement dimension values from that dimension.

Java example of extracting implicit refinements

```
Navigation nav = ENEQueryResults.getNavigation();
DimensionList compDims = nav.getCompleteDimensions();
for (int j=0; j<compDims.size(); ++j) {
  Dimension dim = (Dimension) compDims.get(j);
  if (dim.isImplicit()) {
    DimLocationList dimLocList = dim.getImplicitLocations();
    for (int i = 0; i < dimLocList.size(); i++) {
      %> Implicit dimension value: <%=
      ((DimLocation)dimLocList.get(i)).getDimValue().getName()
      %><%
    }
  }
}
```

.NET example of extracting implicit refinements

```
Navigation nav = ENEQueryResults.Navigation;
DimensionList compDims = nav.CompleteDimensions;
for (int j=0; j<compDims.Count; ++j) {
  Dimension dim = (Dimension) compDims[j];
  if (dim.IsImplicit()) {
    DimLocationList dimLocList = dim.ImplicitLocations;
    for (int i = 0; i < dimLocList.Count; i++) {
      %> Implicit dimension value: <%=
      ((DimLocation)dimLocList[i]).DimValue.Name %> <%
    }
  }
}
```

Creating a new query from refinement dimension values

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

As an example of creating a new Navigation query, assume that this `Red Wine` query:

```
N=40
```

returns two refinement dimensions (Year and Score).

The application needs to create a new query from the current query results to expose the refinement dimension values for the Year dimension. Using the `Dimension.getId()` method (Java) or the `Dimension.Id` property (.NET), the application needs to build a link to a second request:

```
N=40&Ne=2
```

Now that we have results with actual refinement values exposed, we need to create a third query that combines the current query (Red Wine) with the new refinement dimension value (1992). To create this new value for the Navigation (`N`) parameter, use the `ENEQueryToolkit` class. The application creates a `DimValIdList` object by using the following method with Navigation and DimVal parameters:

- Java: ENEQueryToolkit.selectRefinement(nav, ref)
- .NET: ENEQueryToolkit.SelectRefinement(nav, ref)

Calling the `toString()` method (Java) or the `ToString()` method (.NET) on this object will produce the proper Navigation (`N`) parameter for this third query. If the refinement dimension value ID is 66 for the dimension value 1992, the following query would be created for this refinement:

```
N=40+66
```

If you want to render implicit refinements differently than standard refinements, you can use this method to determine if a refinement is implicit:

- Java: `ENEQueryToolkit.isImplicitRefinement()`
- .NET: `ENEQueryToolkit.IsImplicitRefinement()`

You can also use the procedure documented in the previous section, "Extracting implicit refinements from a dimension."

Java example of creating refinement queries from current query results

```
DimVal ref = (DimVal)refs.get(J);
DimValIdList nParams =
   Navigation ENEQueryToolkit.selectRefinement(nav, ref);
%>
```

```
<a href="N=<%= nParams.toString() %>"><%= ref.getName() %></a>
<%
```

.NET example of creating refinement queries from current query results

```
DimVal ref = (DimVal)refs[J];
DimValIdList nParams =
    Navigation ENEQueryToolkit.SelectRefinement(nav,ref);
%>
<a href="N=<%= nParams.ToString() %>"><%= ref.Name %></a>
<%
```

Accessing dimensions with hierarchy

For dimensions that contain hierarchy, the refinement dimension object may contain additional information that is useful when displaying refinement values for that dimension.

Ancestors

For ancestors, these calls return a list of dimension values that describe the path from the root of a dimension to the current selection within the dimension:

- Java: `Dimension.getAncestors()` method
- .NET: `Dimension.Ancestors` property

For example, if a Wineries dimension contained four levels of hierarchy (Country, State, Region, Winery) and the current query was at the region level (Sonoma Valley), the ancestor list would consist of the dimension value United States first and the dimension value California second:

```
Wineries (root) > United States (ancestor) >
California (ancestor) > Sonoma Valley (descriptor)
```

Refinement dimension values, in this case specific wineries, may still exist for this dimension to refine the query even further. Even though ancestors are normally used to describe selected dimension values, they can also be used to help qualify a list of refinement dimension values. (The refinements are not just wineries, they are United States > California > Sonoma Valley wineries.)

Refinement parent

The refinement parent dimension value is accessed with:

- Java: `Dimension.getRefinementParent()` method
- .NET: `Dimension.RefinementParent` property

These calls return the single dimension value directly above the list of refinements for a given dimension. (In the Ancestors example above, the refinement parent would be Sonoma Valley.)

If no dimension values have already been selected for a given dimension, this refinement parent is the root dimension value (Wineries). If a dimension value has already been selected for a given dimension with hierarchy, this refinement parent is the descriptor dimension value (Sonoma Valley). This single call to retrieve either the root or the descriptor makes creating navigation controls simpler. (There is no need to check whether a hierarchical dimension has already been selected from or not.)

For a flat dimension with no hierarchy, the refinement parent will always be the dimension root, because there would be no further refinements if a value had already been selected for the dimension.

Important note about hierarchy

Refinements for a given dimension can only be returned from the MDEX Engine on the same level within the dimension. For example, the MDEX Engine could never return a list of refinement choices that included a mix of countries, states, and regions. (The only exception is flat dimensions that are dynamically organized and/or promoted by the MDEX Engine.)

But in all cases where hierarchy is explicitly defined for a dimension, only refinements on an equal level of hierarchy will be returned for a given query.

Non-navigable refinements

There is a special type of refinement dimension value, found only in dimensions with either explicitly defined or dynamically generated hierarchy, that is referred to as a non-navigable refinement dimension value.

These special values do not actually refine the records returned with a navigation request, but instead specify a deeper level of hierarchy from which to display normal refinement dimension values.

For example, if the Wineries dimension contained 1000 wineries and there was no geographic information from which to create meaningful hierarchy (as in the example above), the best option would be to have the MDEX Engine create dynamic alphabetical hierarchy.

The first set of refinements that would be returned for this dimension would be non-navigable refinements (such as A, B, C, etc.). When a user selects the refinement dimension value A, the resulting query would not limit the record set to only bottles of wine whose winery begins with A. It would, however, return the same record set but with only valid refinement wineries that begin with A. After selecting a specific winery, the resulting query would then limit the record set to only wines from the selected winery.

By this definition, it is important to note that refinement dimension value IDs for non-navigable choices are not valid Navigation (`N`) parameter values. Therefore, they should not be used with these methods:

- Java: `ENEQueryToolkit.selectRefinement()`
- .NET: `ENEQueryToolkit.SelectRefinement()`

(Note that these methods will ignore the request to refine based on a non-navigable refinement.) In order to expose the next level of refinements, this non-navigable dimension value ID must be used with the `Ne` (Exposed Refinements) parameter.

If a non-navigable refinement (or more than one) has been selected for a given dimension, the non-navigable dimension values can be retrieved from the resulting dimension object with:

- Java: `Dimension.getIntermediates()`
- .NET: `Dimension.Intermediates`

Using ENEQueryToolkit.selectRefinement

This ENEQueryToolkit method is necessary for querying hierarchical dimensions.

When generating a new Navigation parameter for a refinement, it is important to use this method:

- Java: `ENEQueryToolkit.selectRefinement()`
- .NET: `ENEQueryToolkit.SelectRefinement()`

One reason for using this method is that it actually implements important business logic.

For example, the query `Red Wine`:

```
N=40
```

returns a refinement dimension value Merlot (ID=41).

Due to the hierarchical nature of the Wine Type dimension, the Merlot refinement is actually in the same dimension as the dimension value in the current query. The new query that is generated by the `selectRefinement()` method (`SelectRefinement()` in .NET), therefore, is:

```
N=40
```

It is not:

```
N=40+41
```

This is an important distinction: When querying hierarchical dimensions, only a single dimension value can be used for each dimension within the Navigation (`N`) parameter. (Multi-select AND or OR dimensions can have more than one dimension value in the Navigation parameter, but cannot be hierarchical.) Therefore, it is important and safer to always use the `selectRefinement()` method (`SelectRefinement()` in .NET) when creating new queries for refinement dimension values.

Related Links

Creating a new query from selected dimension values on page 127
You can use selected dimension values to create additional queries.

Performance impact for displaying refinements

Run-time performance of the MDEX Engine is directly related to the number of refinement dimension values being computed for display.

If any refinement dimension values are being computed by the MDEX Engine but not being displayed by the application, stricter use of the `Ne` parameter is recommended. Obviously, dimensions containing large numbers of refinements also affect performance.

The worst-case scenario for run-time performance is having a data set with a large number of dimensions, each dimension containing a large number of refinement dimension values, and setting the `ENEQuery.setNavAllRefinements()` method (Java) or `ENEQuery.NavAllRefinements` property (.NET) to `true`. This would create a page with an overwhelming number of refinement choices for the user.

Displaying disabled refinements

You can display disabled refinements in the user interface of your front-end Endeca application. These are refinements that are currently disabled in the navigation state but that would have been available if the users didn't make some of the choices they have made by reaching a particular navigation state.

About disabled refinements

Disabled refinements represent those refinements that end users could reach if they were to remove some of the top-level filters that have been already selected from their current navigation state.

A core capability of the MDEX Engine is the ability to provide meaningful navigation options to the users at each step in the guided navigation process. As part of this approach, the MDEX Engine does not return "dead ends" -- these are refinements under which no records are present. In other words,

at each step in the guided navigation, the users are presented with a list of refinements that are valid based on their current navigation state.

In many front-end applications, it is desirable to have a user interface that allows users to see the impact of their refinement selections. In particular, once the users make their initial selections of dimensions and refine by one or more of them, it is often useful to see not only the refinements that are available at each step in the navigation but also the disabled refinements that would have been available if some of the other selections were made.

Such refinements are typically displayed in the front-end application as grayed out, that is, they are not valid for clicking in the current state but could be valid if the navigation state were to change.

To configure disabled refinements, you do not need to change the Endeca project configuration XML files used with Forge, Endeca Workbench, and Developer Studio. You also do not change any settings in the Endeca Workbench and Developer Studio. No changes are required to existing Forge pipelines. The index format of the Dgidx output does not change.

You configure the display of the disabled refinements on a per query basis. You can do this using either of these methods:

- Presentation API methods, or URL parameters. For information, see the topics in this section.
- The MDEX XQuery (MAX) API (if you are using XQuery and Web services for Endeca). For information, see the *XQuery and Web Services Developer's Guide*.

Configuring disabled refinements

Front-end application developers who wish to display disabled refinements need to introduce a specific front-end application code that augments queries with the configuration for disabled refinements.

The MDEX Engine computes the refinements that must be returned based on two navigation states:

- **The base navigation state**. This is the regular navigation state with some of the top-level filters removed.

  Note: In this context, filters refer to the previously chosen range filters, record filters, EQL filters, text searches, and dimensions (including multiselect-OR dimensions) that act as filters for the current navigation state.

- **The default navigation state**. This is the navigation state against which the MDEX Engine computes all operations other than those it needs to compute for returning disabled refinements.

The MDEX Engine computes disabled refinements using the following logic:

- It computes refinements as usual, based on the default navigation state.
- For each dimension that has valid refinements in the base navigation state, it computes the additional disabled refinements that would be reachable from the base navigation state.

About top-level filters used for computing the base navigation state

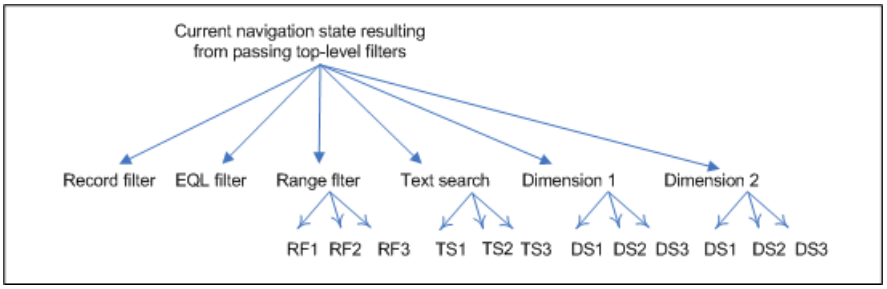
Typically, the MDEX Engine computes refinements and other portions of the response that define the current navigation state based on records that have passed various top-level filters. This section discusses top-level filters, and explains how selections in each of them affect the base navigation state.

The top-level filters can be one of the following:

- Record filters
- EQL filters

- Range filters
- Text searches
- Dimension selections

The following diagram shows these filters:



When the front-end application users make their selections, they can choose items from each of these filters. To compute results for the base navigation state, the MDEX Engine then decides whether to include or remove these filters.

Within each of these filters, users can make multiple selections. For example, for a given Dimension 1, users can make one or more selections, such as DS1, DS2, or DS3. Similarly, they can make more than one selection with text search, or within a specific range filter. It is important to note how the granularity of these choices affects the base navigation state: All selections (and not some) from a given dimension are removed from the base navigation state. Similarly, all text searches and all range filters (and not some) are removed from the base navigation state.

Java class and methods

Use the `DisabledRefinementsConfig` class to display disabled refinement results. The MDEX Engine returns disabled refinements together with the query results.

The methods of this class allow you to specify various parts of the base navigation state. (The MDEX Engine uses the base navigation state to compute disabled refinements.) For example, using the methods from this class, you can specify the following parts of your current navigation state:

- Navigation selections from the dimension specified by the `dimensionId`
- EQL filters
- Range filters
- Text searches

In addition, the following two methods of the ENEQuery class are used for disabled refinements:

- `ENEQuery.setNavDisabledRefinementsConfig()` sets the disabled refinements configuration. A null in disabled refinements configuration means that no disabled refinements will be returned.
- `ENEQuery.getNavDisabledRefinementsConfig()` retrieves the disabled refinements configuration.

 **Note:** If you do not call these methods, the MDEX Engine does not return disabled refinements.

For more information on this class and methods, see the *Endeca API Javadocs*.

Java example

The following example illustrates the front-end application code required for returning disabled refinements along with the query results:

```
ENEQuery query = new ENEQuery();

// ...
// Set up other query parameters appropriately
// ...

DisabledRefinementsConfig drCfg = new DisabledRefinementsConfig();
// Include text searches in base navigation state
drCfg.setTextSearchesInBase(true);
// Include navigation selections from the dimension with ID 100000 in base
 navigation state
drCfg.setDimensionInBase(100000, true);
// Provide the disabled refinements configuration
query.setNavDisabledRefinementsConfig(drCfg);
```

.NET class and methods

The `DisabledRefinementsConfig` class lets you configure disabled refinement results which are returned with the query results.

In addition, use the following property of the ENEQuery class to configure the display of disabled refinements:`ENEQuery.Nav.DisabledRefinementsConfig`

For more information on this class and property, see the *Endeca API Guide for .NET*.

.NET example

The following example illustrates the front-end application code required for returning disabled refinements along with the query results:

```
ENEQuery query = new ENEQuery();

// ...
// set up other query parameters appropriately
// ...

DisabledRefinementsConfig drCfg = new DisabledRefinementsConfig();
// Include text searches in base navigation state
drCfg.TextSearchInBase = true;
// Include navigation selections from the dimension with ID 100000 in base
 navigation state
drCfg.setDimensionInBase(100000, true);
// Provide the disabled refinements configuration
query.NavDisabledRefinementsConfig = drCfg;
```

URL query parameter for displaying disabled refinements

The `Ndr` parameter of the Endeca Navigation URL query syntax lets you display disabled refinements.

The `Ndr` parameter links to:

- Java: `ENEQuery.setNavDisabledRefinementsConfig()` method
- .NET: `ENEQuery.NavDisabledRefinementsConfig` property

The Ndr parameter has a dependency on the N parameter, because a navigation query is being performed.

Configuration settings for the Ndr parameter include:

- <basedimid> — an ID of a dimension that is to be included in the base navigation state.
- <eqlfilterinbase> — a true or false value indicating whether the EQL filter is part of the base navigation state.
- <textsearchesinbase> — a true or false value indicating whether text searches are part of the base navigation state.
- <rangefiltersinbase> — a true or false value indicating whether range filters are part of the base navigation state.

When the Ndr parameter equals zero, no disabled refinement values are returned for any dimensions (which improves performance).

Examples of queries with the Ndr parameter

The first example illustrates a query that lets you return disabled refinements. In this example, the Ndr portion of the UrlENEQuery URL indicates that:

- Text search should be included in the base navigation state.
- The navigation selections from the dimension with ID 100000 should be included in the base navigation state.

```
/graph?N=110001+210001&Ne=400000&Ntk=All&Ntt=television&Ndr=textsearchesin¬
base+true+basedimid+100000
```

In the second example of a query, in addition to text searches, the EQL filters and range filters are also listed (they are set to false):

```
N=134711+135689&Ntk=All&Ntt=television&Ndr=basedimid+100000+textsearchesin¬
base+true+eqlfilterinbase+false+rangefiltersinbase+false
```

Identifying disabled refinements from query output

Disabled refinements are returned in the same way regular refinements are returned. In addition, you can identify from query output whether a particular dimension value is a disabled refinement.

In the Java API, you can identify the dimension value with the Dgraph.DisabledRefinement property. You can identify the value of this property by accessing the PropertyMap with the DimVal.getProperties() method.

For example:

```
DimValList dvl = dimension.getRefinements();
for (int i=0; i < dvl.size(); i++) {
  DimVal ref = dvl.getDimValue(i);
  PropertyMap pmap = ref.getProperties();
  // Determine whether this DimVal is a disabled refinement
  String disabled = "";
  if (pmap.get("DGraph.DisabledRefinement") != null) {
    disabled = " ("+pmap.get("DGraph.DisabledRefinement")+")";
  }
}
```

In the .NET API, to determine whether a dimension value is a disabled refinement, use the `Dimval.Properties` property to obtain the `Dgraph.DisabledRefinement` property. For example:

```
DimValList dvl = dimension.Refinements;
for (int i=0; i < dvl.Count; i++) {
  DimVal ref = dvl[i];
  PropertyMap pmap = ref.Properties;
  // Determine whether this DimVal is a disabled refinement
  String disabled = "";
  if (pmap["DGraph.DisabledRefinement"] != null) {
    disabled = " ("+pmap["DGraph.DisabledRefinement"]+")";
  }
}
```

Interaction of disabled refinements with other navigation features

This feature has several interactions with other navigation features.

- Dimensions with hierarchy. Disabled refinements are not returned for hierarchical dimensions.
- Dynamic ranking. Any dimension that is dynamically ranked does not have disabled refinements returned for it. In other words, to display disabled refinements, you need to turn off dynamic ranking.
- Implicit refinements. Using the `--noimplicit` flag to Dgidx disables computation of dimension values for disabled refinements.

Performance impact of disabled refinements

Performance impact from enabling the display of disabled refinements falls into three categories. They are discussed in the order of importance.

- The cost of computation involved in determining the base and default navigation states.

  The base and default navigation states are computed based on the top-level filters that may belong to these states. These filters are text searches, range, EQL and record filters and selections from dimensions. The types and numbers of these top-level filters in the base and default navigation states affect the MDEX Engine processing involved in computing the default navigation state. The more filters exist in the current navigation state, the more expensive is the task; some filters, such as EQL, are more expensive to take into account than others.

- The trade off between using dynamic refinement ranking and disabled refinements.

  In general, these two features pursue the opposite goals in the user interface — dynamic ranking allows you to intelligently return less information to the users based on most popular dimension values, whereas disabled refinements let you return more information to the users based on those refinements that are not available in the current navigation state but would have been available if some of the selections were not made by the users.

  Therefore, carefully consider your choices for the user interface of your front-end application and decide for which of your refinements you would like to have one of these user experiences:

  - Dynamically ranked refinements
  - Disabled refinements

  If, for example, for some dimensions you want to have only the most popular dimension values returned, you need dynamic ranking for those refinements. For it, you set the sampling size of records (with `--esampin`), which directly affects performance: the smaller the sampling, the quicker the computation. However, for those dimensions, the MDEX Engine then does not compute (and therefore, does not return) disabled refinements.

If, on the other hand, in your user experience you would like to show grayed out (disabled) refinements, and your performance allows it, you can decide to enable them, instead of dynamic ranking for those dimensions. This means that for those dimensions, you need to disable dynamic ranking. As a side effect, this involves a performance cost, since computing refinements without dynamic ranking is more expensive. In addition, with dynamic ranking disabled, the MDEX Engine will need to compute refinement counts for more dimension values.

- The cost of navigation queries.

  Disabled refinements computation slightly increases the navigation portion of your query processing. This increase is roughly proportional to the number of dimensions for which you request the MDEX Engine to return disabled refinements.

Implementing dynamic refinement ranking

A core capability of the MDEX Engine is the ability to dynamically order and present the most popular refinement dimension values to the user.

When the dynamic refinement ranking feature is implemented, the refinement dimension values that are returned for a query are pruned to those values that occur most frequently in the requested navigation state; that is, the refinement dimension values that are most popular.

There are two ways that you can configure dynamic refinement ranking for your application:

- By configuring specific dimensions in Developer Studio.
- By using API calls for query-time control of dynamic refinement ranking. Note that by using these calls, you can override the Developer Studio settings for a given dimension.

The following sections describe how to implement these methods.

Tie breaker for dynamic ranking

Dynamic ranking orders the refinement dimension values by:

1. refinement count (descending), then by
2. static rank assigned (descending), then by
3. dimension value id (descending)

If static ranking is not used, all refinement dimension values will have been assigned a static rank of 1 and the dimension value Id will be the ultimate tie breaker. (Static ranking is also known as manual dimension value ranking.) Therefore, you can control the dynamic ranking tie breaker by either assigning a static rank to the dimension value or by controlling the dimension value ID assigned.

Configuring dynamic refinement ranking

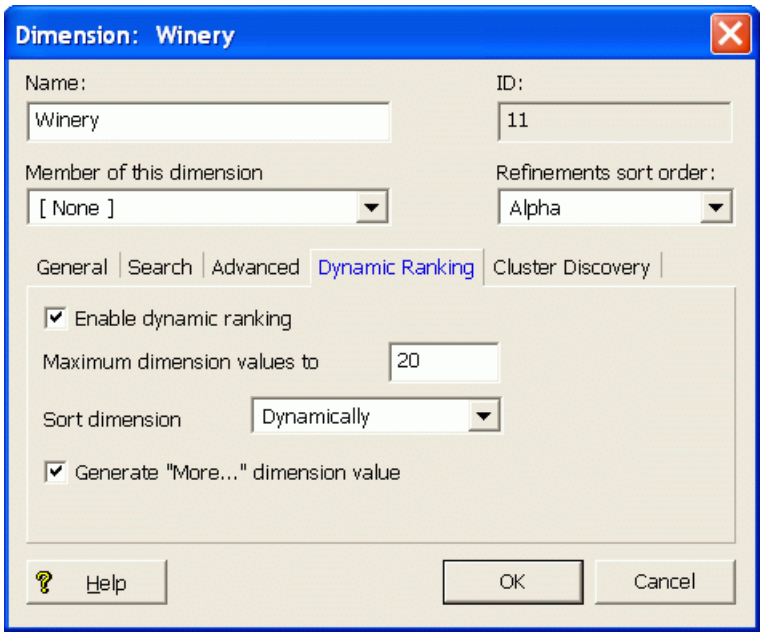
Developer Studio allows you to configure dynamic refinement ranking on a per-dimension basis.

Make sure that you have created the dimension for which you want to enable dynamic refinement ranking.

To configure dynamic refinement ranking:

1. In Developer Studio, open the target dimension in the Dimension editor.
2. Click the **Dynamic Ranking** tab.

3. Check **Enable dynamic ranking**, as in this example.



4. Configure other dimension attributes. The following table lists the meanings of all the fields and checkboxes.

Field	Meaning
Enable dynamic ranking	If checked, enables dynamic refinement ranking for this dimension.
Maximum dimension values to return	Sets the number of most popular refinement dimension values to return.
Sort dimension values	Sets the sort method used for the returned refinement dimension values: • **Alphabetically** uses the sort order specified in the "Refinements sort order" setting on the main part of the Dimension editor. • **Dynamically** orders the most popular refinement values according to their frequency of appearance within a data set. Dimension values that occur more frequently are returned before those that occur less frequently.
Generate "More..." dimension value	If checked, when the actual number of refinement options exceeds the number set in "Maximum dimension values to return", an additional child dimension value (called More) is returned for that dimension. If the user selects the More option, the MDEX Engine returns all of the refinement options for that dimension. If not checked, only the number of dimension values defined in "Maximum dimension values to return" is displayed.

5. Click **OK**.

Related Links

Displaying refinements on page 104
Displaying dimensions and corresponding dimension values for query refinement is the core concept behind Guided Navigation.

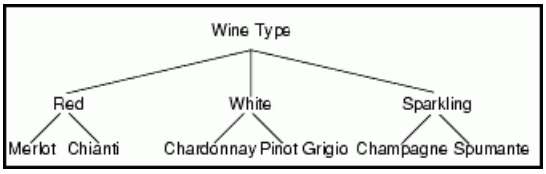
Using query-time control of dynamic refinement ranking

You can configure dynamic refinement ranking to be used on a per-query basis.

The Endeca Presentation API lets you configure dynamic refinement ranking to be switched on and off on a per-query, per-dimension basis, including the number and sort order of refinements to return. This control includes the ability to override the dynamic ranking settings in Developer Studio for a given dimension.

A use case for this dynamic refinement configuration feature 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 dynamic refinement configuration at the dimension value level that you want to control. That is, dynamic ranking will be applied to that dimension value and all its children. For example, assume that you have a dimension named `Wine_Type` that has three child dimension values, `Red`, `White`, and `Sparkling`, which in turn have two child dimension values each. The dimension hierarchy would look like this:



You would set the dynamic refinement configurations depending on which level of the hierarchy you want to order and present, for example:

- If you set the configuration on the root dimension value (which has the same name and ID as the dimension itself), the refinements in the `Red`, `White`, and `Sparkling` dimension values will be returned.
- If there are multiple child dimension values, you can set a configuration on only one sibling. In this case, the refinements from the other siblings will not be exposed. For example, if you set a dynamic refinement configuration on the `Red` dimension value, only the refinements of the `Merlot` and `Chianti`dimension values will be returned. The refinements from the `White` and `Sparkling` dimension values will be not be shown, even if you explicitly set dynamic refinement configurations for them.

Keep the following items in mind when using this feature:

- The settings of the dynamic refinement configuration are not persistent. That is, after the query has been processed by the MDEX Engine, the dynamic ranking settings for the dimension values revert to their Developer Studio settings.
- Setting a dynamic refinement configuration will suppress the generation of a "More..." child dimension value (assuming that the "Generate "More..." dimension value" option has been enabled for the dimension). You can determine whether there are more refinements than the ones shown by checking the `DGraph.More` property on the refinements' parent dimension value.
- The behavior of hidden dimensions is not changed by setting a dynamic refinement configuration on it. That is, the MDEX Engine still will not return the dimension or any of its values as refinement options.
- This bullet discusses the interaction of dynamic refinement ranking with collapsible dimensions. By default, the MDEX Engine considers only leaf dimension values for dynamic ranking, removing all intermediate dimension hierarchy from consideration. With this default behavior, when a hierarchical dimension's mid-level values (all except the root and leaf values) are configured as collapsible in Developer Studio, and when the dimension is also set to use dynamic refinement ranking, the dimension collapses and displays only leaf values for all navigation queries. The

mid-level dimension values are never displayed regardless of the number of leaf values present in the navigation state.

You can use the `--dynrank_consider_collapsed` flag to force the MDEX Engine to consider intermediate collapsible dimension values as candidates for dynamic ranking.

URL query parameter for setting dynamic refinement ranking

The `Nrc` parameter sets the dynamic refinement configuration for the navigation query.

The `Nrc` parameter links to:

- Java: `ENEQuery.setNavRefinementConfigs()` method
- .NET: `ENEQuery.NavRefinementConfigs` property

The `Nrc` parameter has a dependency on the `N` parameter, because a navigation query is being performed.

 **Note:** The `Nrc` parameter works only if dynamic refinement ranking has been enabled.

Nrc parameter syntax

The `Nrc` parameter will have one or more sets of dynamic refinement configurations, with each set being delimited by the pipe character. Each dynamic refinement configuration must begin with the id setting, followed by up to four additional settings, using this syntax:

```
id+dimvalid+exposed+bool+dynrank+setenable+dyncount+maxnum+dynorder+sortorder
```

The meanings of the individual settings are:

- `id` specifies the ID of the dimension value (the *dimvalid* argument) for which the configuration will be set.
- `exposed` specifies whether to expose the dimension value's refinements. The *bool* value is either `true` (expose the refinements) or `false` (do not expose the refinements). The default is `true`. Note that this setting does not have a corresponding setting in Developer Studio.
- `dynrank` specifies whether the dimension value has dynamic ranking enabled. The valid values are `enabled`, `disabled`, or `default`. This setting corresponds to the "Enable dynamic ranking" setting in Developer Studio.
- `dyncount` sets the maximum number of refinement dimension values to return. The valid values are either `default` or an integer that is equal to or greater than 0. This setting corresponds to the "Maximum dimension values to return" setting in Developer Studio.
- `dynorder` sets the sort method for the returned refinements. The valid values are `static`, `dynamic`, or `default`. The `static` value corresponds to the "Alphabetically" value and the `dynamic` value corresponds to the "Dynamically" value in the "Sort dimension values" setting in Developer Studio.

The omission of a setting (other than `id`) or specifying the value `default` results in using the setting in Developer Studio.

Nrc example

The following example sets a dynamic ranking configuration for two dimension values with IDs of 134711 and 132830:

```
N=0&Nrc=id+134711+exposed+true+dynrank+enabled+dyncount
+default+dynorder+dynamic|id+132830+dyncount+7
```

Dimension value 134711 will have its refinements exposed, have dynamic ranking enabled, use the Developer Studio setting for the maximum number of refinement values to return, and use a dynamic sorting order. Dimension value 132830 will have its refinements exposed (because `true` is the default), return a maximum of 7 refinement values, and use the Developer Studio values for the `dynrank` and `dynorder` settings.

Using refinement configuration API calls

You can use API calls to set the dynamic refinement configuration for the navigation query.

An alternative to the `Nrc` parameter is to use API calls to create and set the dynamic refinement configuration for the navigation query. The general procedure is:

1. You first create a refinement configuration for each dimension value by using the calls of the `RefinementConfig` class. Each refinement configuration will be a `RefinementConfig` object.
2. You then encapsulate the `RefinementConfig` objects in a `RefinementConfigList` object.
3. Finally, you set the refinement configuration list for the query by using the `ENEQuery.setNavRefinementConfigs()` method (Java) or the `ENEQuery.NavRefinementConfigs` property (.NET).

Creating a refinement configuration for a dimension value

The constructor of the `RefinementConfig` class takes the ID of a dimension value to create a `RefinementConfig` object for that dimension value and its children (if any). You then use various setter calls to set the specific configuration attributes. Note that these calls correspond to settings of the `Nrc` parameter.

Dynamic ranking for the dimension value is set by these `RefinementConfig` calls (which correspond to the `Nrc` `dynrank` setting):

- Specifically enabled with the Java `setDynamicRankingEnabled()` method or the .NET `DynamicRanking` property with an argument of `ENABLED`.
- Specifically disabled with the Java `setDynamicRankingDisabled()` method or the .NET `DynamicRanking` property with an argument of `DISABLED`.
- Set to use the Developer Studio setting with the Java `setDynamicRankingDefault()` method or the .NET `DynamicRanking` property with an argument of `DEFAULT`.

The `RefinementConfig.setExposed()` method (Java) or `RefinementConfig.Exposed` property (.NET) specify whether to expose the dimension value's refinements. These calls correspond to the `Nrc` `exposed` setting.

The sort method for the returned dimension value is set by these `RefinementConfig` calls (which correspond to the `Nrc` `dynorder` setting):

- Set a dynamic sort order with the Java `setDynamicRankOrderDynamic()` method or the .NET `DynamicRankOrder` property with an argument of `DYNAMIC`.
- Set a static sort order with the Java `setDynamicRankOrderStatic()` method or the .NET `DynamicRankOrder` property with an argument of `STATIC`.

- Use the Developer Studio settings with the Java `setDynamicRankOrderDefault()` method or the .NET `DynamicRankOrder` property with an argument of `DEFAULT`)

The maximum number of dimension values to return is set with the `RefinementConfig.setDynamicRefinementCount()` method (Java) or the `RefinementConfig.DynamicRefinementCount` property (.NET). Use an empty `OptionalInt` argument to use the Developer Studio setting. These calls correspond to the `Nrc dyncount` setting.

The following is a simple Java example of setting a dynamic refinement configuration on the dimension value with an ID of 7:

```
// create an empty refinement config list
RefinementConfigList refList = new RefinementConfigList();
// create a refinement config for dimval 7
RefinementConfig refConf = new RefinementConfig(7);
// enable dynamic refinement ranking for this dimval
refConf.setDynamicRankingEnabled();
// set a dynamic sort order
refConf.setDynamicRankOrderDynamic();
// expose the refinements
refConf.setExposed(true);
// set maximum number of returned refinements to 5
OptionalInt refCount = new OptionalInt(5);
refConfsetDynamicRefinementCount(refCount);
// add the refinement config to the list
refList.add(0, refConf);
// set the refinement config list in the query
usq.setNavRefinementConfigs(refList);
```

Setting the refinement configurations for the query

The constructor of the `RefinementConfigList` class will create an empty list. You then insert `RefinementConfig` objects into the list with:

- Java: the `add()` method
- .NET: the `Add` property

You set the refinement configuration list for the query by using:

- Java: the `ENEQuery.setNavRefinementConfigs()` method
- .NET: the `ENEQuery.NavRefinementConfigs` property

Displaying the returned refinement values

The refinement dimension values can be displayed like any other dimension values.

Regardless of whether you used the `Nrc` parameter or the API calls for the dynamic refinement configuration, you display the returned refinement dimension values in the same way as you display refinements.

As mentioned earlier, setting a dynamic refinement configuration on a dimension value will suppress the generation of a "More..." child dimension value. You can determine whether there are more refinements by checking the `DGraph.More` property on the refinements' parent dimension value:

- If the value of the `DGraph.More` property is 0 (zero), there are no more refinements to display.
- If the value of the `DGraph.More` property is 1 (one), there are more refinements to display.

Related Links

Displaying refinements on page 104

Displaying dimensions and corresponding dimension values for query refinement is the core concept behind Guided Navigation.

Performance impact of dynamic refinement ranking

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

For dynamic refinement ranking, the MDEX Engine first sorts the refinements by the dynamic counts assigned to them, and then cuts to the value you specify in Developer Studio ("Maximum dimension values to return" in the Dynamic Ranking tab of the Dimension editor). Those remaining values are sorted again, alpha- or dynamic-based on your configuration ("Sort dimension values" in the Dynamic Ranking tab), and then finally a "More" link is appended to the returned refinements.

The actual cut is not done using the actual refinement counts of the refinement, as that would be very expensive. Instead, the records in your navigation state are sampled to see if they have a given value or not. After a given number have been sampled, the list is sorted according to the sample counts, and then cut. This means that even with the dynamic rank sorting, you could have the scenario where refinements with more records assigned fall below the More link while others with less records assigned are included above the More link.

The sample size is configurable, but keep in mind that sampling the entire navigation state can be one of the more performance intensive operations the engine does, so you should be very careful in tweaking the size. This accomplished with the Dgraph `--esampmin` option, which allows you to specify the minimum number of records to sample during refinement computation. The default is 0.

For most applications, larger values for `--esampmin` reduce performance without improving dynamic refinement ranking quality. For some applications with extremely large, non-hierarchical dimensions (if they cannot be avoided), larger values can meaningfully improve dynamic refinement ranking quality with minor performance cost.

Displaying descriptors

Displaying descriptors is the ability to display a summary of the navigation refinements that have been made within the current navigation query.

Descriptors (also called selected dimension values) are the dimension values that were used to query for the current record set. The display of these values can take various forms, dependent upon the application. They could be displayed in a linear, navigation history format, or through a stacked list of values. With these values displayed to the user, the user can also be given the ability to remove individual refinement values from their navigation query, thereby increasing the scope of their search.

No Dgidx or Dgraph flags are necessary to enable displaying descriptors. Any dimension value that has been selected is available to be displayed.

URL parameters for descriptors

Selected dimension values are only returned with a valid navigation query.

Because descriptors (selected dimension values) are only returned with a valid navigation query, the Navigation parameter (`N`) is required for any request that will render navigation selections:

```
N=dimension-value-id1+dimension-value-id2[+...]
```

The Navigation parameter is used to indicate the selections made to the MDEX Engine via this set of *dimension-value-ids*. These selected dimension value IDs are the descriptors of the Navigation query. That is, the descriptors are what describe a navigation query. The descriptors are what a user has already selected.

The only exception to this is the URL query:

```
N=0
```

where the descriptors consist of a single ID of zero that does not correspond to any dimension value. Instead a dimension value ID of 0 indicates the absence of any descriptors. It indicates that no dimension values have been selected. When a navigation query is issued with a descriptor of 0, there will be no selected dimension values to render.

Note that the MDEX Engine combines selections from the same dimension into similar dimension objects. This consolidation is why ancestors and descriptors exist, because they were independent selections, but then combined into one dimension object that relates them by the dimension's hierarchy.

Performance impact for descriptors

Performance is rarely impacted by rendering the selected dimension values, because rendering selected dimension values is merely a product of displaying what has already been computed. Like other features related to navigation, performance of the system as a whole is dependent on the complexity and specifics of the data and the dimension structure itself.

Retrieving descriptor dimension values

The Navigation and Dimension classes have methods for getting descriptor dimensions and their dimension values.

To retrieve descriptor dimension values:

1. Access the `Navigation` object from the query results object.
2. After the application has retrieved the `Navigation` object, retrieve a list of dimensions (a `DimensionList` object) that contain descriptors with:

Option	Description
Java	`Navigation.getDescriptorDimensions()` method
.NET	`Navigation.DescriptorDimensions` property

These calls return descriptor dimension values.

An alternative way is to use:

Option	Description
Java	`Navigation.getDescriptorDimGroups()` method
.NET	`Navigation.DescriptorDimGroups` property

These calls return a list of dimension groups (a `DimGroupList` object) instead of a list of dimensions. Each dimension group then contains a list of one or more dimensions with descriptors.

If one of the descriptors is a hierarchical ancestor of another, the MDEX Engine consolidates descriptors into single dimensions. The only exception to this is when a dimension is marked for `multi-select`. When a dimension is marked for `multi-select and` or `multi-select or`, the consolidation is not made and each descriptor gets its own dimension object.

3. Once a descriptor dimension has been retrieved, use these calls to extract various selected dimension value information from the dimension:

Option	Description
`Dimension.getDescriptor()` method (Java) and `Dimension.Descriptor` property (.NET)	Retrieve the dimension value that has been selected from this dimension.
`Dimension.getAncestors()` method (Java) and `Dimension.Ancestors` property (.NET)	Retrieve a list of the ancestors of the descriptor of this dimension. Each member of this list is also a selected dimension value from the same dimension as the descriptor. The distinction between each member of this list and the descriptor is that each ancestor is a hierarchical ancestor to the descriptor by the dimension structure. These ancestors are ordered from parent to child.

Examples: retrieving and rendering descriptors

Java example of retrieving descriptors:

```
Navigation nav = ENEQueryResults.getNavigation();
// Get list of the dimensions with descriptors
DimensionList dl = nav.getDescriptorDimensions();
// Loop through the list
for (int I=0; I < dl.size(); I++) {
  // Get a dimension from the list
  Dimension d = (Dimension)dl.get(I);
  // Get the descriptor and then its name and ID
  DimVal desc = d.getDescriptor();
  String descName = desc.getName();
  long descId = desc.getId();
  // Get list of descriptor’s ancestors and their info
  DimValList ancs = d.getAncestors();
  for (int J=0; J < ancs.size(); J++) {
    DimVal anc = (DimVal)ancs.get(J);
    String ancName = anc.getName();
    long ancId = anc.getId();
  }
}
```

.NET example of retrieving descriptors:

```
Navigation nav = ENEQueryResults.Navigation;
// Get list of the dimensions with descriptors
DimensionList dl = nav.DescriptorDimensions;
// Loop through the list
for (int I=0; I < dl.Count; I++) {
  // Get a dimension from the list
  Dimension d = (Dimension)dl[I];
  // Get the descriptor and then its name and ID
  DimVal desc = d.Descriptor;
  string descName = desc.getName();
  long descId = desc.Id;
  // Get list of descriptor’s ancestors and their info
  DimValList ancs = d.Ancestors;
  for (int J=0; J < ancs.Count; J++) {
```

```
    DimVal anc = (DimVal)ancs[J];
    String ancName = anc.Name;
    long ancId = anc.Id;
  }
}
```

Java example of rendering descriptors:

```
<table>
<%
Navigation nav = ENEQueryResults.getNavigation();
DimensionList dl = nav.getDescriptorDimensions();
for (int I=0; I < dl.size(); I++) {
  Dimension d = (Dimension)dl.get(I);
  %> <tr>
  <%
  DimValList ancs = d.getAncestors();
  for (int J=0; J < ancs.size(); J++) {
    DimVal anc = (DimVal)ancs.get(J);
    %> <td><%= anc.getName() %>
<%
  }
  DimVal desc = d.getDescriptor();
  %> <td><%= desc.getName() %></td></tr>
  <%
}
%>
</table>
```

.NET example of rendering descriptors:

```
<table>
<%
Navigation nav = ENEQueryResults.Navigation;
DimensionList dl = nav.DescriptorDimensions;
for (int I=0; I < dl.Count; I++) {
  Dimension d = (Dimension)dl[I];
  %> <tr>
  <%
  DimValList ancs = d.Ancestors;
  for (int J=0; J < ancs.Count; J++) {
    DimVal anc = (DimVal)ancs[J];
    %> <td><%= anc.Name %>
<%
  }
  DimVal desc = d.Descriptor;
  %> <td><%= desc.Name %></td></tr>
  <%
}
%>
</table>
```

Creating a new query from selected dimension values

You can use selected dimension values to create additional queries.

The following two sections show how you can use the selected refinements to generate queries that remove selected dimension values as well as select ancestors of the selected descriptors.

Removing descriptors from the navigation state

Once you have the selected dimension values, additional queries can be generated for the action of removing a selection. A descriptor is a specific type of selected dimension value. The descriptor is the hierarchically lowest selected dimension value for a dimension.

One query that can be generated from the descriptor is the query where a descriptor is removed. You can use the `ENEQueryToolkit` to generate the query where the descriptor is removed from the current query. You pass in the `Navigation` object and the descriptor to generate the navigation query, as in these examples:

```
// Java version
DimValIdList removed = ENEQueryToolkit.removeDescriptor(nav, desc);

// .NET version
DimValIdList removed = ENEQueryToolkit.RemoveDescriptor(nav, desc);
```

The Java `removeDescriptor()` and .NET `RemoveDescriptor()` methods generate a `DimValIdList` object. The object can be used as the Navigation (`N`) parameter for the additional query by calling the Java `toString()` or .NET `ToString()` method of this object.

The following code snippets show how to create queries that remove descriptors.

Java example of creating queries that remove descriptors

```
// Get the descriptor from the dimension
DimVal desc = dim.getDescriptor();
// Remove the descriptor from the navigation
DimValIdList dParams = ENEQueryToolkit.removeDescriptor(nav,desc);
%>
<a href="/controller.jsp?N=<%= dParams.toString() %>">
<img src="delete.gif"></a>
<%
```

.NET example of creating queries that remove descriptors

```
// Get the descriptor from the dimension
DimVal desc = dim.Descriptor;
// Remove the descriptor from the navigation
DimValIdList dParams = ENEQueryToolkit.RemoveDescriptor(nav,desc);
%>
<a href="/controller.aspx?N=<%= dParams.ToString() %>">
<img src="delete.gif"></a>
<%
```

Selecting ancestors

Another query that you could generate from selected dimension values would be a query for selecting an ancestor. An ancestor is any hierarchical ancestor of a dimension's current descriptor. The resulting query from selecting an ancestor is the existing navigation state with the current descriptor removed, and the ancestor that is selected as the new descriptor. As with removing a descriptor, you would use the `ENEQueryToolkit` class:

```
// Java version
DimValIdList selected = ENEQueryToolkit.selectAncestor(nav,anc,desc);

// .NET version
DimValIdList selected = ENEQueryToolkit.SelectAncestor(nav,anc,desc);
```

The Java `selectAncestor()` and .NET `SelectAncestor()` methods take the `Navigation` object, the ancestor to select, and the descriptor as parameters.

Java example of selecting an ancestor as the new descriptor

```
// Get the ancestor
DimVal anc = (DimVal)ancestors.get(i);
// Use the ancestor in the navigation
DimValIdList sParams = ENEQueryToolkit.selectAncestor(nav,anc,desc);
%>
<a href="/controller.jsp?N=<%= sParams.toString() %>">
<%= anc.getName() %></a>
<%
```

.NET example of selecting an ancestor as the new descriptor

```
// Get the ancestor
DimVal anc = (DimVal)ancestors[i];
// Use the ancestor in the navigation
DimValIdList sParams = ENEQueryToolkit.SelectAncestor(nav,anc,desc);
%>
<a href="/controller.aspx?N=<%= sParams.ToString() %>">
<%= anc.Name %></a>
<%
```

Displaying refinement statistics

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

Dimension value statistics count the number of records (in the current navigation state) or aggregated records beneath a given dimension value. These statistics are dynamically computed at run-time by the Endeca MDEX Engine and are displayed in the user interface.

By providing the user with an indication of the number of records (or aggregated records) that will be returned for each refinement, dimension value statistics can enhance the Endeca application's navigation controls by providing more context at each point in the Endeca application.

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

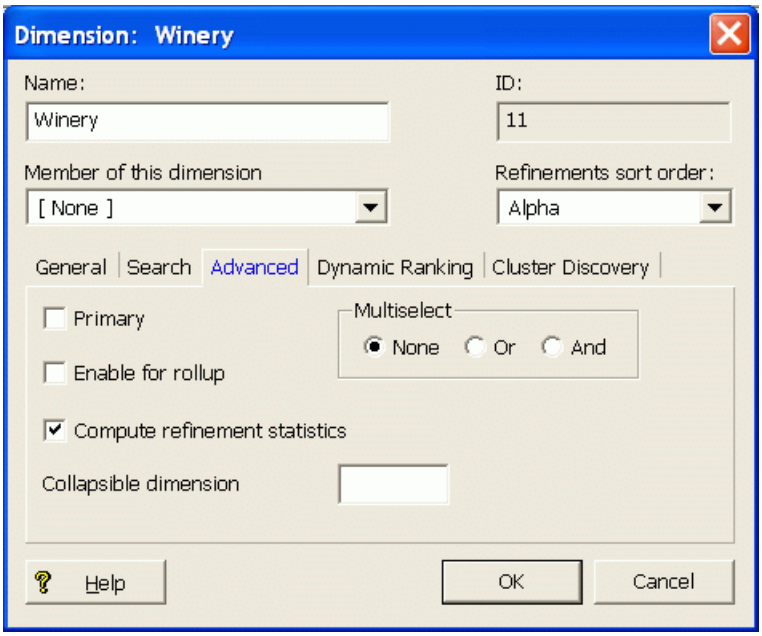
Note that there is no special URL query parameter to request dimension value statistics. So long as there are dimension values returned for a given request, dimension value statistics will be returned as a property attached to each dimension value.

Enabling refinement statistics for dimensions

You configure refinement statistics for regular (non-aggregated) records in Developer Studio.

To configure dimensions for refinement statistics:

1. In Developer Studio, open the target dimension in the Dimension editor.
2. Click the **Advanced** tab.
3. Check **Compute refinement statistics**, as in this example.



4. Click **OK**.

Only the configured dimensions will be considered for computation of dynamic dimension value statistics by the Endeca MDEX Engine.

To enable refinement statistics for aggregated records (that is, those records that are rolled up into a single record for display purposes), use the `--stat-abins` flag with the Dgraph. You cannot enable refinement statistics for aggregated records using Developer Studio.

Retrieving refinement counts for records

Record counts are returned in two Dgraph properties.

To retrieve the counts for regular (non-aggregated) or aggregated records beneath a given refinement (dimension value), use these Dgraph properties:

- Counts for regular (non-aggregated) records on refinements are returned as a property on each dimension value. For regular records, this property is `DGraph.Bins`.
- Counts for aggregated records are also returned as a property on each dimension value. For aggregated records, this property is `DGraph.AggrBins`.

For a given `Navigation` object, request all refinements within each dimension with:

- Java: `Dimension.getRefinements()` method
- .NET: `Dimension.Refinements` property

The refinements are returned in a `DimValList` object.

For each refinement, the dimension value (`DimVal` object) that is a refinement beneath the dimension can be returned with:

- Java: `DimValList.getDimValue()` method
- .NET: `DimValList.Item` property

To get a list of properties (PropertyMap object) associated with the dimension value, use:

- Java: `DimVal.getProperties()` method

- .NET: `DimVal.Properties` property

Calling the `PropertyMap.get()` method (Java) or `PropertyMap` object (.NET) at this point, with the `DGraph.Bins` or `DGraph.AggrBins` argument will return a list of values associated with that property. This list should contain a single element, which is the count of non-aggregated or aggregated records beneath the given dimension value.

The following code samples show how to retrieve the number of records beneath a given dimension value. The examples retrieve the number of regular (non-aggregated) records, because they use the `DGraph.Bins` argument for the calls. To retrieve the number of aggregated records, use the same code, but instead use the `DGraph.AggrBins` argument.

Java example of getting the record counts beneath a refinement

```
DimValList dvl = dimension.getRefinements();
for (int i=0; i < dvl.size(); i++) {
  DimVal ref = dvl.getDimValue(i);
  PropertyMap pmap = ref.getProperties();
  // Get dynamic stats
  String dstats = "";
  if (pmap.get("DGraph.Bins") != null) {
    dstats = " ("+pmap.get("DGraph.Bins")+")";
  }
}
```

.NET example of getting the record counts beneath a refinement

```
DimValList dvl = dimension.Refinements;
for (int i=0; i < dvl.Count; i++) {
  DimVal ref = dvl[i];
  PropertyMap pmap = ref.Properties;
  // Get dynamic stats
  String dstats = "";
  if (pmap["DGraph.Bins"] != null) {
    dstats = " ("+pmap["DGraph.Bins"]+")";
  }
}
```

Retrieving refinement counts for records that match descriptors

For each dimension that has been enabled to return refinement counts, the MDEX Engine returns refinement counts for records that match descriptors. Descriptors are selected dimension values in this navigation state.

The refinement counts that the Dgraph returns for descriptors are returned with the `DGraph.Bins` or `DGraph.AggrBins` property on the descriptor `DimVal` object returned through the Endeca navigation API.

The count represents the number of records (or aggregate records, in the case of `DGraph.AggrBins`) that match this dimension value in the current navigation state.

- For a multi-AND or a single-select dimension, this number is the same as the number of matching records.
- For a multi-OR dimension, this number is smaller than the total number of matching records if there are multiple selections from that dimension.

This capability of retrieving refinement counts for descriptors is the default behavior of the MDEX Engine. No additional configuration (for example, Dgraph command line options) is needed to enable this capability.

To access the refinement counts for descriptors:

- Retrieve the list of dimensions with descriptors. To do this use the `Navigation.getDescriptorDimensions()` method (Java), or the `Navigation.DescriptorDimensions` property (.NET).
- For each dimension, retrieve the dimension value that has been selected from this dimension (the descriptor). To do this, use the `Dimension.getDescriptor()` method (Java) or `Dimension.Descriptor` property (.NET).
- Retrieve the `PropertyMap` object which represents the properties of the dimension value. To do this, use the `DimVal.getProperties()` method (Java) or the `DimVal.Properties` property (.NET) on that dimension value.
- Obtain a list of values associated with that property. Use the `PropertyMap.get()` method (Java) or `PropertyMap` object (.NET) with the `DGraph.Bins` or `DGraph.AggrBins` argument.

  This list should contain a single element which is the number of records (or aggregate records) that match this dimension value in the current navigation state.

Java example of getting refinement counts for a descriptor

```
Navigation nav = ENEQueryResults.getNavigation();
// Get the list of dimensions with descriptors
DimensionList dl = nav.getDescriptorDimensions();
// Loop through the list
for (int i = 0; i < dl.size(); i++) {
 // Get a dimension from the list
 Dimension d = (Dimension)dl.get(i);
 // Get the descriptor and then its count(s)
 DimVal desc = d.getDescriptor();
 // Get the map of properties for the descriptor
 PropertyMap pmap = desc.getProperties();
 // Get the record count
 String recordCount = "";
 if (pmap.containsKey("DGraph.Bins")) {
  recordCount = " (" + pmap.get("DGraph.Bins") + ")";
 }
 // Get the aggregate record count
 String aggregateRecordCount = "";
 if (pmap.containsKey("DGraph.AggrBins")) {
  aggregateRecordCount = " (" + pmap.get("DGraph.AggrBins") + ")";
 }
}
```

.NET example of getting refinement counts for a descriptor

```
Navigation nav = ENEQueryResults.Navigation;
// Get the list of dimensions with descriptors
DimensionList dl = nav.DescriptorDimensions;
// Loop through the list
for(int i = 0; i < dl.Count; i++) {
 // Get a dimension from the list
 Dimension d = (Dimension)dl[i];
 // Get the descriptor and then its count(s)
 DimVal desc = d.Descriptor;
 // Get the map of properties for the descriptor
 PropertyMap pmap = desc.Properties;
```

```
 // Get the record count
 String recordCount = "";
 if (pmap["DGraph.Bins"] != null) {
  recordCount = " (" + pmap["DGraph.Bins"] + ")";
 }
 // Get the aggregate record count
 String aggregateRecordCount = "";
 if (pmap["DGraph.AggrBins"] != null) {
  aggregateRecordCount = " (" + pmap["DGraph.Bins"] + ")";
 }
}
```

Related Links

Retrieving descriptor dimension values on page 125
The Navigation and Dimension classes have methods for getting descriptor dimensions and their dimension values.

Performance impact of refinement counts

Dynamic statistics on regular and aggregated records are expensive computations for the Endeca MDEX Engine.

You should only enable a dimension for dynamic statistics if you intend to use the statistics in your Endeca-enabled front-end application. Similarly, you should only use the `--stat-abins` flag with the Dgraph to calculate aggregated record counts if you intend to use the statistics in your Endeca-enabled front-end application. Because the Dgraph does additional computation for additional statistics, there is a performance cost for those that you are not using.

In applications where record counts or aggregated record counts are not used, these lookups are unnecessary. The MDEX Engine takes more time to return navigation objects for which the number of dimension values per record is high.

Note that Dgidx performance is not affected by dimension value statistics.

Displaying multiselect dimensions

The MDEX Engine supports two types of multiselect dimensions.

The default behavior of the Endeca MDEX Engine permits only a single dimension value from a dimension to be added to the navigation state. This type of dimension is called a ***single-select*** dimension.

By default, after a user selects a leaf refinement from any single-select dimension, that dimension is removed from the list of dimensions available for refinement in the query results. For example, after selecting "Apple" from the Flavors dimension, the Flavors dimension is removed from the navigation controls.

However, sometimes it is useful to allow the user to select more than one dimension value from a dimension. For example, you can give a user the ability to show wines that have a flavor of "Apple" and "Apricot". This function is accomplished by tagging the dimension as a ***multiselect*** dimension. The MDEX Engine provides support for two types of multiselect dimensions that apply Boolean logic to the dimension values selected:

- `multiselect-AND`

- `multiselect-OR`

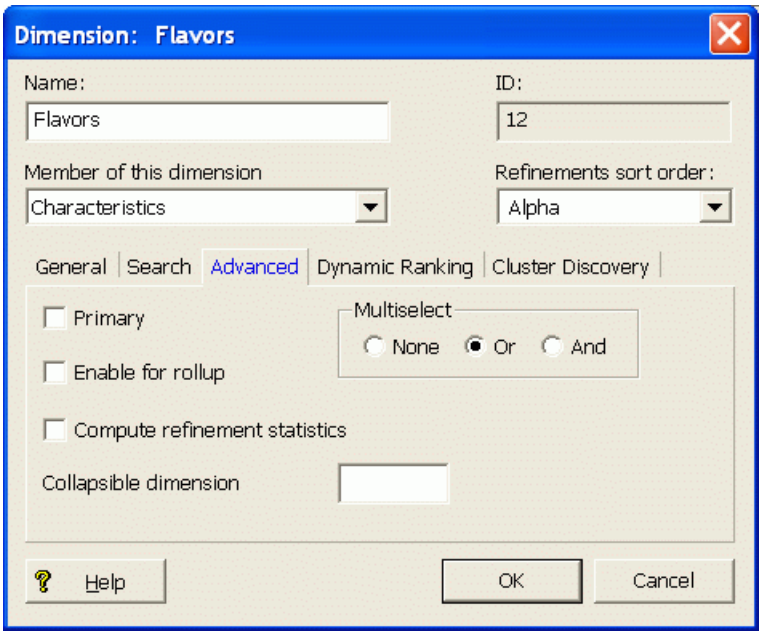
The multiselect feature is only fully supported for flat dimensions (that is, dimensions that do not contain hierarchy). In other words, multiselect-OR queries are restricted to leaf dimension values. In a flat dimension, all possible refinements are leaf dimension values, so no extra configuration is necessary. In a hierarchical dimension, you must configure all non-leaf dimension values to be inert (non-navigable) to prevent them from appearing in the navigation query.

Configuring multiselect dimensions

You use Developer Studio to configure the multiselect feature for a dimension.

To configure a multiselect dimension:

1. In Developer Studio, open the target dimension in the Dimension editor.
2. Click the **Advanced** tab.
3. In the Multiselect frame, select either **Or** or **And**, as in this example which configures a Multiselect-OR dimension.



4. Click **OK**.

After you re-run Forge and Dgidx, the dimension will be enabled for multiselect queries.

Handling multiselect dimensions

The behavior of multiselect dimensions may require changes in the UI.

The fact that a dimension is tagged as multiselect should be transparent to the Presentation API developer. There is no special Presentation API development required to enable multiselect dimensions. There are no URL Query Parameters or API objects that are specific to multiselect dimensions.

However, the semantics of how the MDEX Engine interprets navigation queries and returns available refinements changes once a dimension is tagged as multiselect. After tagging a dimension as

multiselect, the MDEX Engine will then allow multiple dimension values from the same dimension to be added to the navigation state.

The MDEX Engine behaves differently for the two types of multiselect dimensions:

- **Multiselect-AND** – The MDEX Engine treats the list of dimension values selected from a multiselect-AND dimension as a Boolean AND operation. That is, the MDEX Engine will return all records that satisfy the Boolean AND of all the dimension values selected from a multiselect-AND dimension (for example, all records that have been tagged with "Apple" AND "Apricot"). The MDEX Engine will also continue to return refinements for a multiselect-AND dimension. The list of available refinements will be the set of dimension values that have not been chosen, and are still valid refinements for the results.
- **Multiselect-OR** – A multiselect-OR dimension is analogous to a multiselect-AND dimension, except that a Boolean OR operation is performed instead (that is, all records that have been tagged with "Apple" OR "Apricot"). Keep in mind that selections from the multiselect-OR dimension do not affect what is returned. Though the result record set is determined using all selections in the navigation state, the MDEX Engine chooses the set of multiselect-OR refinements by looking at the set of records and ignoring existing selections from that multiselect-OR dimension. Also note that as more multiselect-OR dimension 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.

Comparing single-select and multiselect-OR dimensions

A comparison of single-select and multiselect-OR dimensions shows the difference in the generation of standard and implicit refinements. The table shows these differences using a simplified case with only one selected dimension value:

Single-select dimension	Multiselect-OR dimension
Children of the current dimension value are potential refinements because selecting one could reduce your record set. Those that would change your record set if selected are standard refinements, while those that would not change your record set if selected are implicit refinements.	Children of the selected dimension value are not potential refinements, because selecting one would not expand the record set. Therefore, they are the implicit selections.
Ancestors of the dimension value are not potential refinements, because selecting one would not reduce the record set. They are the implicit selections.	Ancestors of the selected dimension value are potential refinements, because selecting one could expand your record set. Those that would change your record set if selected are standard refinements, while those that would not change your record set if selected are implicit refinements.
Dimension values in the subtrees rooted at the siblings of the selected dimension value and its ancestors are also not potential refinements, because they correspond to record sets which are disjoint (or at least uninteresting to the user, based on their selected dimension value.) Note that these dimension values are not available as refinements in single-select dimensions, but are accessible in multiselect-AND dimensions.	Dimension values in the subtrees rooted at the siblings of the selected dimension value and its ancestors are also potential refinements, because selecting one could expand your record set. Those that would change your record set if selected are standard refinements, while those that would not change your record set if selected are implicit refinements.

The process of navigation in a single-select dimension can be conceptualized as walking up and down the dimension value tree. Multiselect-OR dimensions, in constrast, are inverted with respect to

refinement generation: dimension values in the subtrees rooted at selections are implicit refinements, while all other dimension values are potential refinements.

Avoiding dead-end query results

Be careful when rendering the selected dimension values of multiselect-OR dimensions. It is possible to create an interface that might result in dead-ends when removing selected dimension values.

Consider this example: Dimension Alpha has been flagged as multiselect-OR, and contains dimension values 1 and 2. Dimension Beta contains dimension value 3.

Assume the user's current query contains all three dimension 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 dimension values from Dimension Alpha, a dead end could be reached. For example, if the user removes dimension 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 dimension 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 dimension values from a multiselect-OR dimension when making changes to the list of selected dimension values.

Refinement counts for multiselect-OR refinements

Refinement counts on a refinement that is multiselect-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 is the same as it would be for single select and multi select dimensions. However, for subsequent selections, the refinement count may differ from the total number of records in the result set.

For example, suppose a `Cuisine` refinement is configured as multiselect-OR. In the data set, there are 2 records tagged with only a `Chinese` property, 3 records tagged with only a `Japanese` property, and 1 record that has both of these properties.

Record	Tagged with a `Chinese` property	Tagged with a `Japanese` property
1	x	
2	x	
3	x	x
4		x
5		x
6		x

If an application user has not made any refinement selections yet, the refinement counts are as follows:

`Cuisine`

`[ ] Chinese` (3)

`[ ] Japanese` (4)

The record result list for this navigation state includes records 1, 2, 3, 4, 5, and 6.

If an application user first selects only `Chinese`, the refinement count is as follows:

```
Cuisine
[x] Chinese
[ ] Japanese (4)
```

The record result list for this navigation state includes records 1, 2, and 3.

If an application user selects both `Chinese` and `Japanese`, as shown:

```
Cuisine
[x] Chinese
[x] Japanese
```

Then the record result list for this navigation state includes records 1, 2, 3, 4, 5, and 6.

Performance impact for multiselect dimensions

Refinements for multiselect-OR dimensions are more expensive than refinements from single-select dimensions.

When making decisions about when to tag a dimension as multiselect, keep the following in mind: Users will take longer to refine the list of results, because each selection from a multiselect dimension still allows for further refinements within that dimension.

Using hidden dimensions

Hidden dimensions are not returned as refinement options.

A hidden dimension is like a regular dimension in that it is composed of dimension values that allow the user to refine a set of records. It differs from a non-hidden dimension in its accessibility in the user interface.

If a dimension is marked as hidden, the MDEX Engine will not return the dimension or any of its values as a refinement option in the navigation menu. However, if a given record is tagged with a value from a hidden dimension, the MDEX Engine returns this value with a record query, assuming the dimension is configured to render on the product page.

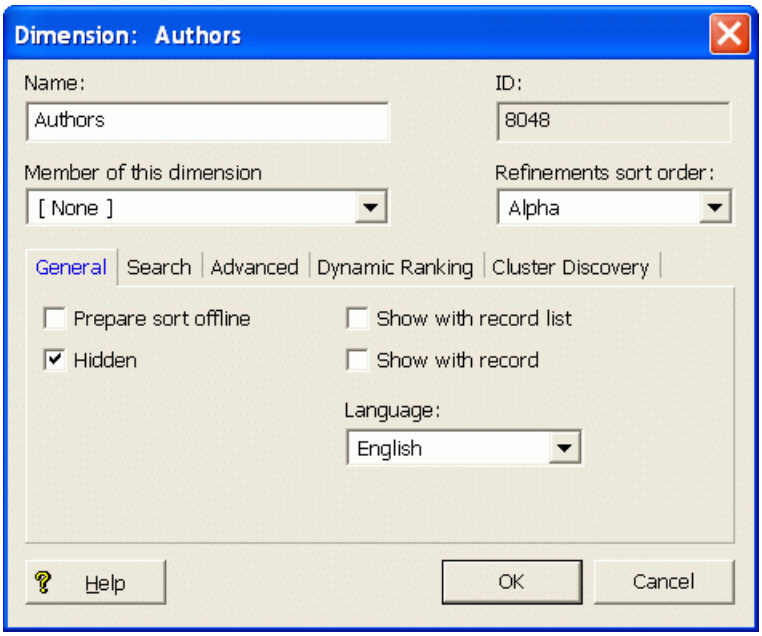
Although hidden dimensions are not rendered in UI navigation, records are still indexed with relevant values from these dimensions. Therefore, a user is able to search for records based on values within hidden dimensions.

Configuring hidden dimensions

You use Developer Studio to configure a dimension as hidden.

To configure a hidden dimension:

1. In Developer Studio, open the target dimension in the Dimension editor.
2. In the **General** tab, check **Hidden**, as in this example.



3. Click **OK**.

There are no Dgidx or Dgraph flags necessary to enable hidden dimensions. If a dimension was properly specified as hidden in Developer Studio, it will automatically be indexed as a hidden dimension.

Handling hidden dimensions in an application

The UI can add hidden dimensions to the navigation state.

As a rule, the Endeca MDEX Engine only returns hidden dimensions and their values for single record requests and not for navigation requests. Hidden dimensions, when returned, are accessed in the same manner as regular (non-hidden) dimensions.

Example of using a hidden dimension

Marking a dimension as hidden is useful in cases where the dimension is composed of numerous values and returning these values as navigation options does not add useful navigation information. Consider, for example, an Authors dimension in a bookstore. Scanning thousands of authors for a specific name is less useful than simply using keyword search to find the desired author.

In this case, you would specify that the Authors dimension be hidden. The user will be able to perform a keyword search on a particular author, but will not be able to browse on author names in order to find books by the author. Also, once the user has located a desired book (either by keyword search or by navigating within other dimensions), she may be interested in other books by the same author.

While the user would have been unable to refine her navigation by choosing an author, after finding a particular book she can include that author in her navigation state, in effect creating a store of books by that author. (The activity of adding or removing dimension values to or from the navigation state is known as pivoting.)

Performance impact of hidden dimensions

In cases where certain dimensions in an application are composed of many values (see the Authors dimension example above), marking such dimensions as hidden will improve Endeca Presentation

API and Endeca MDEX Engine performance to the extent that queries on large dimensions will be limited, reducing the processing cycles and amount of data the engine must return.

When a dimension is hidden, the precompute phase of indexing will be shortened because refinements from hidden dimensions need not be computed.

Using inert dimension values

You can create and use inert dimension values, which are dimension values that are not navigable.

Marking a dimension value as inert makes it non-navigable. That is, the dimension value should not be included in the navigation state.

From an end user perspective, the behavior of an inert dimension value is similar to the behavior of a dimension within a dimension group: With dimension groups, the dimension group behaves like a dimension and the dimension itself behaves like an inert child dimension value. When the user selects the dimension, the navigation state is not changed, but instead the user is presented with the child dimension values. Similarly, when a user selects an inert dimension value, the navigation state is not changed, but the children of the dimension value are displayed for selection.

Whether or not a dimension value should be inert is a subjective design decision about the navigation flow within a dimension. Two examples of when you might use inert dimension values are the following:

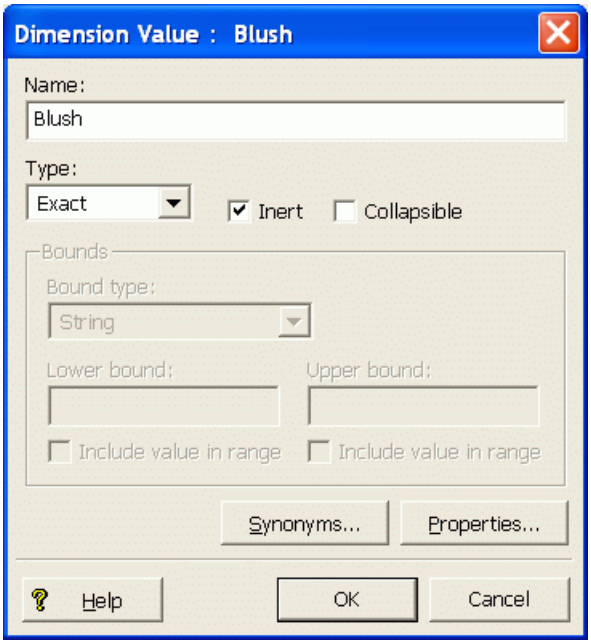
- You want the "More..." option to be displayed at the bottom of an otherwise long list. To do this, use Developer Studio's Dimension editor to enable dynamic ranking for the dimension and generate a "More…" dimension value.
- You want to define other dimension values that provide additional information to users, but for which it is not meaningful to filter items.

Configuring inert dimension values

You use Developer Studio to configure dimension values as inert (non-navigable).

To configure dimension values as inert:

1. In the Project tab of Developer Studio, double-click **Dimensions** to open the Dimensions view.
2. Select a dimension and click **Edit**. The Dimension editor is displayed.
3. Select a dimension and click **Values**. In the Dimension Values view, the Inert column indicates which dimension values have been marked as inert.
4. Select a dimension value and click **Edit**. The Dimension Value editor is displayed.
5. Check **Inert**, as in this example.



6. Click **OK**. The Dimensions view is redisplayed, with a Yes indicator in the Inert column for the changed dimension.

There are no Dgidx or Dgraph flags necessary to mark a dimension value as inert. Once a dimension has been marked as inert in Developer Studio, the Presentation API will be aware of its status.

Handling inert dimension values in an application

If you are using inert dimension values, the UI should check whether the DimVal object is navigable.

When sending the new navigation state to the MDEX Engine, the Endeca application should check the value of the Java `isNavigable()` or .NET `IsNavigable()` method on each `DimVal` object. Only dimension values that are navigable (that is, not inert) should be sent to the MDEX Engine, for example, via the Java `ENEQuery.setNavDescriptors()` method or the `ENEQuery.NavDescriptors` property.

Setting the Inert attribute for a dimension value indicates to the Presentation API that the dimension value should be inert. However, it is up to the front-end application to check for inert dimension values and handle them in an appropriate manner.

The following code snippets show how a `DimVal` object is checked to determine if it is a navigable or inert dimension value. In the example, the `N` parameter is added to the navigation request only if the dimension value is navigable (not inert).

Java example of handling inert dimension values

```
// Get refinement list for a Dimension object
DimValList refs = dim.getRefinements();
// Loop over refinement list
for (int k=0; k < refs.size(); k++) {
  // Get refinement dimension value
  DimVal dimref = refs.getDimValue(k);//
  // Create request to select refinement value
  urlg = new UrlGen(request.getQueryString(), "UTF-8");
  // If refinement is navigable, change the Navigation parameter
  if (dimref.isNavigable()) {
```

```
    urlg.addParam("N",
      (ENEQueryToolkit.selectRefinement(nav,dimref)).toString());
    urlg.addParam("Ne",Long.toString(rootId));
  }
  // If refinement is non-navigable, change only the exposed
  // dimension parameter (leave the Navigation parameter as is)
  else {
    urlg.addParam("Ne",Long.toString(dimref.getId()));
  }
}
```

.NET example of handling inert dimension values

```
// Get refinement list for a Dimension object
DimValList refs = dim.Refinements;
// Loop over refinement list
for (int k=0; k < refs.Count; k++) {
  // Get refinement dimension value
  DimVal dimref = (DimVal)refs[k];
  // Create request to select refinement value
  urlg = new UrlGen(Request.Url.Query.Substring(1), "UTF-8");
  // If refinement is navigable, change the Navigation parameter
  if (dimref.IsNavigable()) {
    urlg.addParam("N",
      (ENEQueryToolkit.SelectRefinement(nav,dimref)).ToString());
    urlg.AddParam("Ne",rootId.ToString());
  }
  // If refinement is non-navigable, change only the exposed
  // dimension parameter (Leave the Navigation parameter as is)
  else {
    urlg.AddParam("Ne",dimref.Id.ToString());
  }
}
```

Displaying dimension value properties

Dimension value properties provide descriptive information about a given dimension value and can be used for display purposes.

Dimension value properties are used to pass data about dimension values through the system for interpretation by the Presentation API. The data stored in the properties is typically ignored by Forge and the MDEX Engine. Instead, the Presentation API uses the information to support display features. For example, a property could contain the URL of an icon that should be displayed next to the dimension value.

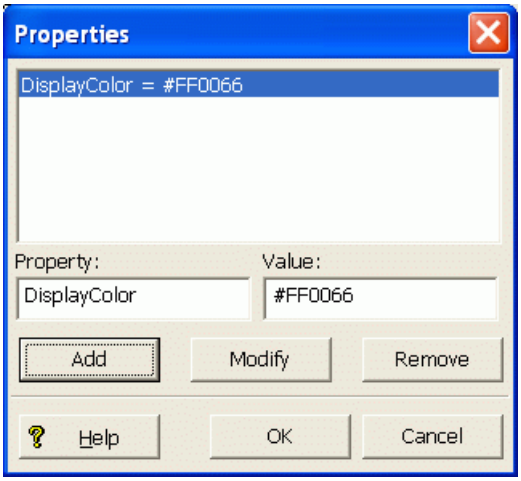
Configuring dimension value properties

You use Developer Studio to configure properties for dimension values.

To configure dimension value properties:

1. In the Project tab of Developer Studio, double-click **Dimensions** to open the Dimensions view.
2. Select a dimension and click **Edit**. The Dimension editor is displayed.

3. Select a dimension and click **Values**. In the Dimension Values view, the Properties column indicates which dimension values have properties.
4. Select a dimension value to which you want to add a property and click **Edit**. The Dimension Value editor is displayed.
5. Click **Properties**. The Properties editor is displayed.
6. Enter the name of the property in the Property field, the property's value in the Value field, and click **Add** to add the property. The Property editor should look like this example.



7. You can add multiple properties. When you have finished adding properties, click **OK**. You are returned to the Dimension Value editor.
8. In the Dimension Value editor, click **OK**. The Dimensions view is redisplayed, with the new property listed in the Properties column for the changed dimension.

Note that no Dgidx or Dgraph flags are necessary to enable the use of dimension value properties.

Accessing dimension value properties

The application can access the dimension value properties via PropertyMap objects.

After a dimension value (`DimVal` object) has been retrieved, the application can access the dimension value properties by calling:

- Java: the `DimVal.getProperties()` method
- .NET: the `DimVal.Properties` property

Working with dimension value properties is similar to working with record properties. In both cases, the same `PropertyMap` object is returned.

The following code fragments which show how to iterate through all properties of a dimension value.

Java example of accessing dimension value properties

```
// Loop over refinement list
// refs is a DimValList object
for (int k=0; k < refs.size(); k++) {
  // Get refinement dimension value
  DimVal ref = refs.getDimValue(k);
  // Get properties for refinement value
  PropertyMap pmap = ref.getProperties();
  // Get all property names and their values
  Iterator props = pmap.entrySet().iterator();
```

```
  while (props.hasNext()) {
    Property prop = (Property)props.next();
    String pkey = prop.getKey();
    String pval = prop.getValue();
    // Perform operation on pkey and/or pval
  }
}
```

.NET example of accessing dimension value properties

```
// Loop over refinement list
// refs is a DimValList object
for (int k=0; k < refs.Count; k++) {
  // Get refinement dimension value
  DimVal ref = refs[k];
  // Get properties for refinement value
  PropertyMap pmap = ref.Properties;
  // Get all property names and their values
  System.Collections.IList props = pmap.EntrySet;
  foreach (Property prop in props) {
    String pkey = prop.Key;
    String pval = prop.Value;
    // Perform operation on pkey and/or pval
  }
}
```

Getting specific properties by name

Note that instead of iterating through all properties for a given dimension value, you can also get specific properties by name from the `PropertyMap` object, as shown in these examples.

Java example of getting a specific property

```
<%
// Get properties for refinement value
PropertyMap pmap = ref.getProperties();
// Get the desired property
String propVal = "";
if (pmap.get("DisplayColor") != null) {
  propVal = pmap.get("DisplayColor");
%>
  <FONT COLOR="<%= propVal %>">Best Buy</FONT>
<%
}
```

.NET example of getting a specific property

```
<%
// Get properties for refinement value
PropertyMap pmap = ref.Properties;
// Get the desired property
String propVal = "";
// If property has a value
if ((String)pmap["DisplayColor"] != "")
  propVal = (String)pmap["DisplayColor"];
%>
  <FONT COLOR="<%= propVal %>">Best Buy</FONT>
<%
}
```

Performance impact for displaying dimension value properties

Dimension value properties could slightly increase the processing and/or querying time because additional data is moved through the system, but this effect will generally be minimal.

If your Endeca application does complex formatting on the properties, this could slow down page-loads, but ideally the information will be used to add formatting HTML or perform other trivial operations, which will have minimal impact on performance.

Working with external dimensions

Endeca applications can use dimensions created outside of Developer Studio.

You can also import or otherwise access dimensions created or managed outside of Endeca Developer Studio. For details, see the *Endeca Forge Guide*.

Chapter 12

Dimension Value Boost and Bury

This chapter describes the Dimension Value Boost and Bury feature.

About the dimension value boost and bury feature

Dimension value boost and bury is a mechanism by which the ranking of certain specific dimension values is made much higher or lower than others.

Dimension value boost and bury is a feature that allows users to re-order returned dimension values. With ***dimension value boost*** , you can assign specific dimension values to ranked strata, with those in the highest stratum being shown first, those in the second-ranked stratum shown next, and so on. With ***dimension value bury*** , you can specify that specific dimension values should be ranked much lower relative to others. This boost/bury mechanism therefore lets you manipulate ranking of returned dimension values in order to promote or push certain types of records to the top or bottom of the results list.

The feature depends on the use of the `Nrcs` URL parameter or the related Presentation API methods. The feature also works with the use of static refinement ranking as well as dynamic refinement ranking.

Note: The dimension value boost and bury feature and the `Nrcs` parameter are not supported by the Aggregated MDEX Engine (Agraph).

Use cases

This feature is especially suited for eCommerce sites, in which it can be used for two distinct use cases:

- Site promotion of a house brand (i.e., globally boost a dimension value over all pages). For example, a site may have a private label that they would like to ensure always shows up as a refinement everywhere on the site for business reasons.
- Landing page promotion of a single dimension value or refinement that is important to that category. Assume, for example, a site that sells CDs. Willie Nelson has produced many records, some of which are categorized as both country and rock. The site wants to promote (boost) Willie Nelson in the Country category rather than in the Rock category.

Immediate consumers of this feature are sites using Endeca Merchandising Workbench. Using Merchandising Workbench and Page Builder, a merchandiser defines a set of rules to fire and to boost or bury individual dimension values based on an end user's navigation state.

Nrcs parameter

The `Nrcs` parameter sets the list of stratified dimension values for use during refinement ranking by the MDEX Engine.

The `Nrcs` parameter groups specified dimension values into strata. The stratified dimension values specified in the parameter are delimited by semi-colons (;) and each stratified dimension value is in the format:

```
stratumInt,dimvalID
```

where *dimvalID* is the ID of the dimension value and *stratumInt* is a signed integer that signifies the stratum into which the dimension value will be placed.

The `Nrcs` parameter thus provides a mapping of dimension values to strata in the query:

- Boosted dimension values will use a strata of 1 or greater (> 0).
- Buried dimension values will use a strata of less than 0 (< 0).
- Dimension values that are not specified will be assigned the strata of 0.

You can define as many strata as you wish, but keep the following in mind:

- For boosted strata (i.e., strata defined with a positive >0 integer), numerically-higher strata are boosted above numerically-lower strata. For example, dimension values in strata 2 are boosted above dimension values in strata 1.
- Dimension values within a specific stratum are returned in an indeterminate manner. For example, if the dimension values with IDs of 5000 and 6000 are assigned to a stratum, it is indeterminate as to which dimension value (5000 or 6000) will be returned first from a query.
- Ties will be broken with whichever type of dynamic refinement ranking is in use (alphabetically or dynamically).

Note that a dimension value will be stratified in the highest strata it matches, so boosting will have priority over burying.

Nrcs example

In this example, three strata are defined (strata 2, strata 1, and strata -1):

```
Nrcs=2,3001;2,3002;1,4001;1,4002;1,4003;-1,5001;-1,5002
```

When the query is processed, the dimension values are returned in this order:

1. Dimension values 3001 and 3002 are boosted above all others (i.e., are in the highest-ranked stratum).
2. Dimension values 4001 and 4002 are returned next (i.e., are in the second-ranked stratum).
3. All non-assigned dimension values are returned as part of stratum 0 (i.e., are in the third-ranked stratum).
4. Finally, dimension values 5001 and 5002 are buried (i.e., are in the lowest-ranked stratum).

This example shows how you can construct a hierarchy for the returned dimension values, and control the strata in which they are placed.

Nrcs setter methods

The Nrcs parameter is linked to these methods in the Presentation API:

- The `ENEQuery.setNavStratifiedDimVals()` method in the Java version of the API.
- The `ENEQuery.NavStratifiedDimVals` property in the .NET version of the API.

Stratification API methods

The Presentation API has methods that can programmatically set the dimension boost and bury configuration in the query.

ENEQuery class

The `ENEQuery` class has these stratification calls:

- The Java `setNavStratifiedDimVals()` method and .NET `NavStratifiedDimVals` setter property set the list of stratified dimension values in the query for use during refinement ranking by the MDEX Engine. These calls link to the `Nrcs` URL query parameter.
- The Java `getNavStratifiedDimVals()` method and .NET `NavStratifiedDimVals` getter property retrieves the list of stratified dimension values.

StratifiedDimVal and StratifiedDimValList classes

A `StratifiedDimVal` object represents the assignment of a dimension value to a specific stratum for sorting. The object thus contains:

- A long that specifies the ID of the dimension value.
- An integer that represents the stratum to which the dimension value is assigned. A positive integer indicates that the dimension value will be boosted, while a negative integer indicates that the dimension value will be buried.

A `StratifiedDimValList` object encapsulates a collection of `StratifiedDimVal` objects. The `StratifiedDimValList` object is set in the `ENEQuery` object by the `setNavStratifiedDimVals()` Java method and the `NavStratifiedDimVals` .NET property.

Example of using the API methods

The following Java example illustrates how to use these methods to send the dimension value boost and bury configuration to the MDEX Engine:

```
// Create a query
ENEQuery usq = new ENEQuery();

// Create an empty stratified dimval list
StratifiedDimValList stratList = new StratifiedDimValList();

// Set dimval 3001 to be boosted and add it to stratList
StratifiedDimVal stratDval1 = new StratifiedDimVal(1,3001);
stratList.add(0,stratDval1);

// Set dimval 5001 to be buried and add it to stratList
StratifiedDimVal stratDval2 = new StratifiedDimVal(-1,5001);
stratList.add(1,stratDval2);

// Set the stratified dval list in the query object
usq.setNavStratifiedDimVals(stratList);
// Set other ENEQuery parameters
...
```

The example sets the dimension value with an ID of 3001 to be boosted and dimension value ID 5001 to be buried. The .NET of this example

Retrieving the DGraph.Strata property

Dimension values that are stratified have the `DGraph.Strata` property set to include the strata value used for sorting.

You can identify from query output whether a particular dimension value has been stratified by checking whether the `DGraph.Strata` property exists and, if it exists, the stratum value. If the stratum value was specified as "0" or not specified at all, then the property is not returned. Note that navigation descriptors that were stratified will also have the `DGraph.Strata` property set.

In Java, you can identify the value of this property by accessing the dimension value's `PropertyMap` with the `DimVal.getProperties()` method, as in this example:

```
DimValList dvl = dimension.getRefinements();
for (int i=0; i < dvl.size(); i++) {
  DimVal ref = dvl.getDimValue(i);
  PropertyMap pmap = ref.getProperties();
  // Determine whether this DimVal is stratified
  String isStrat = "";
  if (pmap.get("DGraph.Strata") != null) {
    isStrat = " ("+pmap.get("Dgraph.Strata")+")";
  }
}
```

The .NET version of the Presentation API uses the `Dimval.Properties` property:

```
DimValList dvl = dimension.Refinements;
for (int i=0; i < dvl.Count; i++) {
  DimVal ref = dvl[i];
  PropertyMap pmap = ref.Properties;
  // Determine whether this DimVal is stratified
  String isStrat = "";
  if (pmap["DGraph.Strata"] != null) {
    isStrat = " ("+pmap["DGraph.Strata"]+")";
  }
}
```

Interaction with disabled refinements

The dimension value boost and bury feature works correctly with disabled refinements.

To illustrate the interaction of both features, assume that your query (with disabled refinements being enabled) returns the following:

```
Dimension X:
  A (disabled)
  B
  C
  D (disabled)
  E
  F (disabled)
```

You then use the dimension value boost and bury feature. You decide to bury A and boost E and D. The same disabled refinements query would now return:

```
Dimension X:
  D (disabled)
  E
  B
```

```
C
F (disabled)
A (disabled)
```

When using these features in concert, you must be very careful to provide a consistent user experience in your UI. It is very easy to create a situation where implicitly selecting a dimension value will cause a rule to fire which may decide to boost or bury some dimension values. It is very important for the disabled refinements features that the order of dimension values on the page remain the same in order to present a good user experience. Changing the order (by using the boost and bury feature) may confuse the user. Therefore, in general you should try to make sure your set of boosted and buried dimension values is the same in your default and base navigation queries.

Chapter 13

Using Derived Properties

This section describes derived properties and their behavior.

About derived properties

A derived property is a property that is calculated by applying a function to properties or dimension values from each member record of an aggregated record.

Derived properties are created by Forge, based on the configuration settings in the `Derived_props.xml` file. After a derived property is created, the resultant derived property is assigned to the aggregated record.

Aggregated records are a prerequisite to derived properties. If you are not already familiar with specifying a rollup key and creating aggregated records, see the "Creating Aggregated Records" chapter in this guide.

To illustrate how derived properties work, consider a book application for which only unique titles are to be displayed. The books are available in several formats (various covers, special editions, and so on) and the price varies by format. Specifying Title as the rollup key aggregates books of the same title, regardless of format. To control the aggregated record's representative price (for display purposes), use a derived property.

For example, the representative price can be the price of the aggregated record's lowest priced member record. The derived property used to obtain the price in this example would be configured to apply a minimum function to the Price property.

 **Note:** Derived properties cannot be used for record sorting.

Derived property performance impact

Some overhead is introduced to calculate derived properties. In most cases this should be negligible. However, large numbers of derived properties and more importantly, aggregated records with many member records may degrade performance.

Configuring derived properties

The DERIVED_PROP element in the Derived_props.xml file specifies a derived property.

The attributes of the DERIVED_PROP element are:

- DERIVE_FROM specifies the property or dimension from which the derived property will be calculated.
- FCN specifies the function to be applied to the DERIVE_FROM properties of the aggregated record. Valid functions are MIN, MAX, AVG, or SUM. Any dimension or property type can be used with the MIN or MAX functions. Only INTEGER or FLOAT properties may be used in AVG and SUM functions.
- NAME specifies the name of the derived property. This name can be the same as the DERIVE_FROM attribute.

The following is an example of the XML element that defines the derived property described in the book example above:

```
<DERIVED_PROP
   DERIVE_FROM="PRICE"
   FCN="MIN"
   NAME="LOW_PRICE"
/>
```

Similarly, a derived property can derive from dimension values, if the dimension name is specified in the DERIVE_FROM attribute. In addition, the function attribute (FCN) can be MAX, AVG, or SUM, depending on the desired behavior.

Note: Developer Studio currently does not support configuring derived properties. The workaround is to hand-edit the Derived_props.xml file to add the DERIVED_PROP element.

Troubleshooting derived properties

A derived property can derive from either a property or a dimension. The DERIVE_FROM attribute specifies the property name or dimension name, respectively. Avoid name collisions between properties and dimensions, as this is likely to be confusing.

Displaying derived properties

Displaying derived properties in the UI is similar to displaying regular properties.

The Presentation API's semantics for a derived property are similar to those of regular properties, though there are a few differences. Derived properties apply only to aggregated Endeca records. Therefore, the MDEX Engine query must be properly formulated to include a rollup key.

Use the following calls to work with the aggregated record (an AggERec object):

API method or property	Purpose
Java: AggERec.getProperties() .NET: AggERec.Properties	Returns a PropertyMap object that has the derived properties of the aggregated record.
Java: AggERec.getRepresentative() .NET: AggERec.Representative	Returns an ERec object that is the representative record of the aggregated record.

The following code examples demonstrate how to display the names and values of an aggregated record's derived properties.

Java example of displaying derived properties

```
// Get aggregated record list
AggrERecList aggrecs = nav.getAggrERecs();
for (int i=0; i<aggrecs.size(); i++) {
  // Get individual aggregated record
  AggrERec aggrec = (AggrERec)aggrecs.get(i);
  // Get all derived properties.
  PropertyMap derivedProps = aggrRec.getProperties();
  Iterator derivedPropIter = derivedProps.entrySet().iterator();
  // Loop over each derived property,
  // handle as an ordinary property.
  while (derivedPropIter.hasNext()) {
    Property prop = (Property) derivedPropIter.next( );
    // Display property
    %>
    <tr>
    <td>Derived property name: <%= prop.getKey() %></td>
    <td>Derived property value: <%= prop.getValue() %></td>
    </tr>
    <%
  }
}
```

.NET example of displaying derived properties

```
Get aggregated record list
AggrERecList aggrecs = nav.AggrERecs;
// Loop over aggregated record list
  for (int i=0; i<aggrecs.Count; i++) {
  // Get an individual aggregated record
  AggrERec aggrec = (AggrERec)aggrecs[i];
  // Get all derived properties.
  PropertyMap derivedPropsMap = aggrec.Properties;
  // Get property list for agg record
  System.Collections.IList derivedPropsList = derivedPropsMap.EntrySet;
  // Loop over each derived property,
  // handle as an ordinary property.
  foreach (Property derivedProp in derivedPropsList) {
    // Display property
    %>
    <tr><td>Derived property name: <%= derivedProp.Key %></td>
    <td>Derived property value: <%= derivedProp.Value %></td></tr>
    <%
  }
}
```

Chapter 14

Configuring Key Properties

This section describes how to annotate property and dimension keys with metadata using key properties.

About key properties

Endeca analytics applications require the ability to manage and query meta-information about the properties and dimensions in the data.

On a basic level, applications need the ability to determine the types (dimension, Alpha property, numeric (floating point or Integer) property, time/date (Time, DateTime, Duration) property, and so on) of keys in the data set.

For example, knowledge of the set of numeric properties allows the application to present reasonable end-user choices for analytics measures. Knowledge of the set of date/time properties allows the application to present the end-user with reasonable GROUP BY selections using date bucketing operators.

Dimension-level configuration is also useful at the application layer. Knowledge of the multi-select settings for a dimension allows the application to present a tailored user interface for selecting refinements from that dimension (for example, radio buttons for a single select dimension versus check boxes for a dimension enabled for multi-select OR). Knowledge of the precedence rule configuration is useful for rendering dimension tree views. Encoding such information as part of the data rather than hard-coding it into the application enables a cleaner application design that requires less maintenance over time as the data changes.

In addition to Endeca-level information about properties and dimensions, analytics applications require support for managing user-level information about properties and dimensions. Examples of this include:

- Rendering text descriptions of properties and dimensions presented in the application. An example would be mouse-over tool tips that describe the definition of the dimension or property.
- Management of “unit” information for properties. For example, a Price property might be in units of dollars, euros, and so on. A Weight property might be in units of pounds, tons, kilograms, and so on. Knowledge of the appropriate units for a property allows the application to render units on things like charts, while also allowing the application to dynamically conditionalize analytics behavior (so that it would, for example, multiply the euros property by the current conversion rate before adding it to the dollars property).
- General per-property and per-dimension application behavior controls. For example, if the data is stored in a denormalized form, a nested GROUP BY may be required before using a property as an analytics measure (for example, with denormalized transaction data, you must GROUP BY “CustomerId” before computing average “Age” to avoid double counting).

The key property feature in the MDEX Engine addresses these needs. The key property feature allows property and dimension keys to be annotated with metadata key/value pairs called key properties (since they are properties of a dimension or property key). These key properties are configured as PROP elements in a new XML file that is part of the application configuration.

In a traditional data warehousing environment, metadata from the warehouse could be exported to an XML key properties file and propagated onwards to the application and rendered to the end user.

Access to key properties is provided to the application through new API methods: the application calls `ENEQuery.setNavKeyProperties` to request key properties, then calls `Navigation.getKeyProperties` to retrieve them.

In addition to developer-specified key properties, `Navigation.getKeyProperties` also returns automatically generated key properties populated by the MDEX Engine. These indicate the type of the key (dimension, Alpha property, Double property, and so on), features enabled for the key (such as sort or search), and other application configuration settings.

Defining key properties

Key properties are defined in an XML file that is part of the application configuration: `<app_config>.key_props.xml`.

A new, empty version of this file is created whenever a new Endeca Developer Studio project is created. Editing this file and performing a **Set Instance Configuration** operation in Developer Studio causes a new set of key properties to be loaded into the system.

The DTD for the `<app_config>.key_props.xml` is located in `$ENDECA_ROOT/conf/dtd/key_props.dtd`. The contents of this DTD are:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  Copyright (c) 2001-2004, Endeca Technologies, Inc.
      All rights reserved.
-->

<!ENTITY % common.dtd SYSTEM "common.dtd">
%common.dtd;

<!--  The KEY_PROPS top level element is the container for a set
      of KEY_PROP elements, each of which contains the
      "key properties" for a single dimension or property key.
-->
<!ELEMENT KEY_PROPS (COMMENT?, KEY_PROP*)>


<!--  A KEY_PROP element contains the list of property
      values associated with the dimension or property key
      specified by the NAME attribute.
-->
<!ELEMENT KEY_PROP (PROP*)>
<!ATTLIST KEY_PROP
    NAME  CDATA  #REQUIRED
>
```

Each KEY_PROPS element in the file corresponds to a single dimension or property and contains the key properties for that dimension or property. Key properties that do not refer to a valid dimension or property name are removed by the MDEX Engine at startup or configuration update time and are logged with error messages.

Here is an example of a key properties XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE KEY_PROPS SYSTEM "key_props.dtd">

<KEY_PROPS>
  <KEY_PROP NAME="Gross">
    <PROP NAME="Units"><PVAL>$</PVAL></PROP>
    <PROP NAME="Description">
    <PVAL>Total sale amount, exclusive of any deductions.
    </PVAL>
    </PROP>
  </KEY_PROP>

  <KEY_PROP NAME="Margin">
    <PROP NAME="Units"><PVAL>$</PVAL></PROP>
    <PROP NAME="Description">
    <PVAL>Difference between the Gross of the transaction
    and its Cost.</PVAL></PROP>
  </KEY_PROP>

</KEY_PROPS>
```

Automatic key properties

In addition to user-specified key properties, the Endeca MDEX Engine automatically populates the following properties for each key: `Endeca.Type`, `Endeca.RecordFilterable`, `Endeca.DimensionId`, `Endeca.PrecedenceRule`, and `EndecaMultiSelect`

Property	Description
`Endeca.Type`	The type of the key. Value is one of: Dimension, String, Double, Int, Geocode, Date, Time, DateTime, or RecordReference.
`Endeca.RecordFilterable`	Indicates whether this key is enabled for record filters. Value one of: true or false.
`Endeca.DimensionId`	The ID of this dimension (only specified if `Endeca.Type=Dimension`).
`Endeca.PrecedenceRule`	Indicates that a precedence rule exists with this dimension as the target, and the indicated dimension as the source. Value: Dimension ID of the source dimension.
`Endeca.MultiSelect`	If `Endeca.Type=Dimension` and this dimension is enabled for multi-select, then this key property indicates the type of multi-select supported. Value one of: OR or AND.

Key property API

Key properties can be requested as part of an Endeca Navigation query (ENEQuery).

By default, key properties are not returned by navigation requests to avoid extra communication when not needed. To request key properties, use the `ENEQuery.setNavKeyProperties` method:

```
ENEQuery query = ...
query.setNavKeyProperties(KEY_PROPS_ALL);
```

To retrieve the key properties from the corresponding Navigation result, use the Navigation.getKeyProperties method:

```
ENEQueryResults results = …
Map keyPropMap = results.getNavigation().getKeyProperties();
```

This method returns a Map from String key names to KeyProperties objects, which implement the `com.endeca.navigation.PropertyContainer` interface, providing access to property values through the same interface as an Endeca record (ERec object).

Example: rendering all key properties

For example, to render all of the key properties returned by the MDEX Engine, one could use the following code sample:

```
Map keyProps = nav.getKeyProperties();
Iterator props = keyProps.values().iterator();
while (props.hasNext()) {
  KeyProperties prop = (KeyProperties)props.next();

  // Each key property has a key and a set of values
  String keyPropName = (String)prop.getKey();

  // Get the values which are stored as a PropertyMap
  PropertyMap propVals = (PropertyMap)prop.getProperties();
%>
  <tr><td <%= keyPropName %>  </td></tr>
<%
  // Display properties
  Iterator containedProps = propVals.entrySet().iterator();
  // Iterate over the properties
  while (containedProps.hasNext()) {
    // Display property
    Property propMap = (Property)containedProps.next();
    String propKey = (String)propMap.getKey();
    String propVal = (String)propMap.getValue();
%>
<tr><td><%= propKey %>:</td><td><%= propVal %></td></tr>
<%
  }
}
```

Part 4

Basic Search Features

Chapter 15

About Record Search

This section discusses record search, which is an Endeca equivalent of full-text search, and is one of the fundamental building blocks of Endeca search capabilities.

Record search overview

Record search allows a user to perform a keyword search against specific properties or dimension values assigned to records.

The resulting records that have matching properties or dimension values are returned, along with any valid refinement dimension values.

Unlike dimension search, record search returns a complete `Navigation` object, the same object that is returned when a user filters records by selecting a dimension value.

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 a dimension value does, even though it is not a specific dimension value.

Example of record search

For example, consider the following records:

Rec ID	Dimension value (Wine Type)	Name property	Description property
1	Red (Dim Value 101)	Antinori Toscana Solaia	Dark ruby in color, with extremely ripe...
2	Red (Dim Value 101)	Chateau St. Jean	Dense, rich, and complex describes this California...
3	White (Dim Value 103)	Chateau Laville	Dense and vegetal, with celery, pear, and spice flavors...
4	Other (Dim Value 103)	Jose Maria da Fonseca	Big, ripe, and generous, layered with honey...

When the user performs a record search on the Description property using the keyword `dense`, the following `Navigation` object is returned:

- 2 records (records 2 and 3)
- 2 refinement dimension values (Red and White)

When performing a record search on the Description property using the keyword `ripe`, this `Navigation` object is returned:

- 2 records (records 1 and 4)
- 2 refinement dimension values (Red and Other)

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 property record searches, and property group record searches. These are discussed in detail in their respective sections.

Making properties or dimension searchable

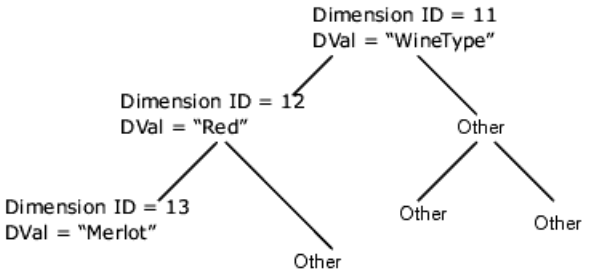
The first step in implementing basic record search is to use Developer Studio to configure a property or dimension for record searching.

Enabling hierarchical record search

If you want to consider ancestor dimension values when matching a record search query, you can enable hierarchical record search in Developer Studio.

By default, a record search that uses a dimension as the search key returns only those records that are assigned a dimension value whose text matches the search terms. As part of this behavior, record search does not consider ancestors which are not directly assigned.

For example, consider the following dimensions hierarchy:



In this hierarchy, the `Red` dimension (with an ID of 12) is an ancestor of the `Merlot` dimension (ID of 13). A search against the `WineType` dimension for the keyword `merlot` matches any records assigned the dimension value 13. But a search in `WineType` for `red merlot` does not match these records, because record search does not normally consider ancestors which are not directly assigned.

In such cases, you may want record search to consider ancestor dimension values when matching a record search query. You can enable this sort of hierarchical record search in Developer Studio.

Adding search synonyms to dimension values

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

When a dimension is used as the record search key, the text strings considered by record search for matching are the individual names of the dimension values within the dimension. The dimension name is automatically added as a searchable string.

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

Features for controlling record search

You can control the various features related to record search either at indexing time or at run-time. This topic lists ways in which you can control record search behavior.

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

- To control indexing behavior, you can use phrase search, wildcard search or other advanced features of record search. For more information, see sections about phrase search, wildcard search and sections about the advanced search capabilities.
- To configure run-time record search behavior, you must create one or more search interfaces. For more information, see the section about search interfaces.
- There are no Dgidx flags necessary to enable record search. If a property or dimension was properly enabled for record search, it will automatically be indexed for searching.
- There are no MDEX Engine configuration flags necessary to enable record searching. If a property or dimension was properly enabled for record searching when indexing, it will automatically be available for record searching when index files are loaded into the MDEX Engine.
- Multiple MDEX Engine configuration flags are available to manage different controls for record search, such as spelling support and relevance ranking. See specific feature sections for details.

Related Links

Using Phrase Search on page 217
Phrase search allows users to specify a literal string to be searched. This section discusses how to use phrase search.

Using Wildcard Search on page 227
Wildcard search allows users to match query terms to fragments of words in indexed text. This section discusses how to use wildcard search.

Working with Search Interfaces on page 173
A *search interface* is a named collection of properties and dimensions, each of which is enabled for record search in Developer Studio.

URL query parameters for record search

A basic record search requires two separate request parameters, `Ntk` and `Ntt`. This topic describes them and contains examples of valid record search queries that use `Ntk` and `Ntt` .

The search key parameters are described as follows:

- `Ntk=<search_key>`. The search key parameter, `Ntk`, specifies which property or dimension is going be evaluated when searching. You specify a property or dimension as a value for this parameter. (You can also specify a search interface as a value for the `Ntk` parameter.)
- `Ntt=<search_term>`. The keyword parameter, `Ntt`, specifies the actual search terms that are submitted.

The URL query parameters for record search have the following characteristics:

- Record search parameters must accompany a standard navigation request, even if that basic navigation request is empty. This is because a record search actually acts as a custom filter on a basic navigation request.

  For example, a request is considered invalid if only the property key (`Ntk`), and keyword (`Ntt`) are specified, without specifying a Navigation value (`N`).
- Likewise, only records currently returned by the basic navigation request (`N`) are considered when performing a record search.
- Record search terms and navigation dimension values together form an `AND` Boolean request.

Examples of queries with Ntt and Ntk

For example, consider the following records:

Rec ID	Dimension value (Wine Type)	Name property	Description property
1	Red (Dim Value 101)	Antinori Toscana Solaia	Dark ruby in color, with extremely ripe...
2	Red (Dim Value 101)	Chateau St. Jean	Dense, rich, and complex describes this California...
3	White (Dim Value 103)	Chateau Laville	Dense and vegetal, with celery, pear, and spice flavors...
4	Other (Dim Value 103)	Jose Maria da Fonseca	Big, ripe, and generous, layered with honey...

In this example, the following query:

```
<application>?N=0&Ntk=Description&Ntt=Ripe
```

returns records 1 and 4, because the navigation request is empty (`N=0`).

However, the following query:

```
<application>?N=101&Ntk=Description&Ntt=Ripe
```

returns only record 1, because the navigation request (`N=101`) is already filtering the record set to records 1 and 2.

The following query, which is missing a navigation request (`N`), is invalid:

```
<application>?Ntk=Description&Ntt=Ripe
```

Methods for using multiple search keys and terms

In a more advanced application, users can search against multiple properties with multiple terms. To do this, `Ntk` and `Ntt` are used together.

You can implement searching multiple properties using `AND` Boolean logic with `Ntk` and `Ntt` with the following query:

```
Ntk=<property_key1>|<property_key2>
Ntt=<search_term1>|<search_term2>
```

In this query, each term is evaluated against the corresponding property. The returned record set represents an intersection of the multiple searches.

Examples of searching multiple terms

For example, assume that a search for the term `cherry` returns 5,000 records while a search for `peach` returns 2,000 records.

However, a multiple search for both terms:

```
<application>?N=0&Ntk=Description|Description&Ntt=cherry|peach
```

returns only 10 records if those 10 records are the only records in which both terms exist in the Description property.

You can use any number of property keys, as long as it matches the number of search terms.

For example, consider the following records:

Rec ID	Dimension value (Wine Type)	Name property	Description property
1	Red (Dim Value 101)	Antinori Toscana Solaia	Dark ruby in color, with extremely ripe...
2	Red (Dim Value 101)	Chateau St. Jean	Dense, rich, and complex describes this California...
3	White (Dim Value 103)	Chateau Laville	Dense and vegetal, with celery, pear, and spice flavors...
4	Other (Dim Value 103)	Jose Maria da Fonseca	Big, ripe, and generous, layered with honey...

In this example, the following query:

```
<application>?N=0&Ntk=Description|Name&Ntt=Ripe|Solaia
```

returns only record 1.

The following query:

```
<application>?N=0&Ntk=Description|Name&Ntt=Ripe
```

is invalid, because the number of record search keys does not match the number of record search terms.

You can also use search interfaces to perform searches against multiple properties. For more information, see the section about search interfaces. For information on performing more complex Boolean queries, see topics about using Boolean search.

Related Links

Search interfaces and URL query parameters (Ntk) on page 175

Use the name of the search interface as the value for the `Ntk` parameter, just as you would use a normal property or dimension.

Methods for rendering results of record search requests

Rendering the results of a record search request is identical to rendering the results of a navigation request. This is because a record search request is a variation of a basic navigation request.

Specific objects and method calls exist that can be accessed from a `Navigation` object and return a list of valid record search keys. (This data is only available from a navigation request, not from a record or dimension search request.)

Java example

A Java code example for rendering results of record search is shown below:

```
ERecSearchKeyList keylist = nav.getERecSearchKeys();
for (int i=0; i < keylist.size(); i++) {
 ERecSearchKey key = keylist.getKey(i);
 String name = key.getName();
 boolean active = key.isActive();
}
```

The `ERecSearchKeyList` object is a vector containing `ERecSearchKey` objects. Each `ERecSearchKey` object contains the name of a property that has been enabled for record search, as well as a Boolean flag indicating whether that property is currently being used as a search key.

.NET example

A .NET code example for rendering results of record search is shown below:

```
ERecSearchKeyList keylist = nav.ERecSearchKeys;
for (int i=0; i < keylist.Count; i++) {
 ERecSearchKey key = (ERecSearchKey)keylist[i];
 String name = key.Name;
 Boolean active = key.IsActive();
}
```

The `ERecSearchKeyList` object is a vector containing `ERecSearchKey` objects. Each `ERecSearchKey` object contains the name of a property that has been enabled for record search, as well as a Boolean flag indicating whether that property is currently being used as a search key.

Search query processing order

This section summarizes how the MDEX Engine 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 MDEX Engine uses the following high-level steps to process record search queries:

1. Record filtering
2. Endeca Query Language (EQL) filtering
3. Tokenization
4. Auto correction (spelling correction and automatic phrasing)
5. Thesaurus expansion
6. Stemming
7. Primitive term and phrase lookup
8. Did you mean
9. Range filtering
10. Navigation filtering
11. Business rules and keyword redirects
12. Analytics
13. 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 MDEX Engine 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.

For more information about record filters, see the *Advanced Development Guide*.

Step 2: Endeca Query Language filters

The Endeca Query Language (EQL) contains a rich syntax that allows an application to build dynamic, complex filters that define arbitrary subsets of the total record set and restrict search and navigation results to those subsets. If used, this feature is applied after record filtering.

For details on this feature, see the *Advanced Development Guide*.

Step 3: Tokenization

Tokenization is the process by which the MDEX Engine analyzes the search query string, yielding a sequence of distinct query terms.

Step 4: Auto correction (spelling correction and automatic phrasing)

If spelling correction and automatic phrasing are enabled and triggered, the MDEX Engine implements them as part of the record search processing.

If the spelling correction feature is enabled and triggered, the MDEX Engine 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 MDEX Engine 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 *Advanced Development Guide*.

Step 5: Thesaurus expansion

The tokenized query, as well as each query alternative generated by spelling suggestion, is expanded by the MDEX Engine 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 `MatchAll`.

The other match modes (with the exception of MatchBoolean) 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 *Advanced Development Guide*.

Step 6: Stemming

Query terms, unless they are delimited with quotation marks to be treated as exact phrases, are expanded by the MDEX Engine 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 *Advanced Development Guide*.

Step 7: Primitive term and phrase lookup

Primitive term and phrase lookup is the lowest level of search processing performed by the MDEX Engine.

The MDEX Engine 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 indexes being searched. Search is never case-sensitive, even for phrases.

Step 8: Did You Mean

The MDEX Engine 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 *Advanced Development Guide*.

Step 9: Range filtering

Range filter functionality allows 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.

Because this step comes after "Did you mean?" processing, it reports the number of records before filtering.

For more details on range filtering, see Chapter 7.

Step 10: Navigation filtering

The MDEX Engine 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 11: Business rules and keyword redirects

Dynamic business rules employ a trigger and target mechanism to promote contextually relevant records to application users as they search and navigate within a data set.

Keyword redirects are similar to dynamic business rules also use trigger and target values. However, keyword redirects are used to redirect a user's search to a Web page (that is, a URL). These features are applied after navigation filtering.

For details on these features, see the *Advanced Development Guide*.

Step 12: Analytics

Endeca Analytics builds on the core capabilities of the Endeca MDEX Engine to enable applications that examine aggregate information such as trends, statistics, analytical visualizations, comparisons, and so on, all within the Guided Navigation interface. If Analytics is used, it is applied near the end of processing.

For more information about this feature, see the *Endeca Analytics Guide*.

Step 13: Relevance ranking

Relevance ranking is the last step in the MDEX Engine 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 *Advanced Development Guide*.

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 MDEX Engine 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. Even though there are objects and methods accessed from the `Navigation` object that enumerate search-enabled Endeca properties, these are normally used for debugging purposes that do not explicitly know this information for a given data set.

In production systems, these Endeca properties are typically hard-coded at the application level, because the application requires specific search keys to be used for specific functionality.

If an Endeca property is not enabled for record searching but an application attempts to perform a record search against this property, the MDEX Engine successfully returns a null result set.

The MDEX Engine error log, however, outputs the following message: `In fulltext search: [Wed Sep 3 12:28:02 2007] [Warning] Invalid fulltext search key "Description" requested.`

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

Finally, while implementing record search by enabling record properties for searching is the normal approach, dimension values can also be enabled for record searching. The dimension name then replaces the property key as the value for the `Ntk` parameter in the MDEX Engine query. The resulting navigation request contains any record that is tagged with a dimension value from the specified dimension that matches the search terms.

Performance impact of record search

Because record searching is an indexed feature, each property enabled for record searching increases the size of both the Dgidx process as well as the MDEX Engine process.

The specific size of the increase is related to the size of the unique word list generated by the specific property in the data set. Therefore, only properties that are specifically needed by an application for record searching should be configured as such.

Chapter 16

Working with Search Interfaces

A *search interface* is a named collection of properties and dimensions, each of which is enabled for record search in Developer Studio.

About search interfaces

A search interface allows you to control record search behavior for groups of one or more properties and dimensions.

A search interface may also contain:

- A number or 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 properties and dimensions
- Keyword in context results
- Partial match

You can use a search interface to control the behavior of search against a single property or dimension, or to simultaneously search across multiple properties and dimensions.

For example, if a data set contains both an `Actor` property and `Director` dimension, 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 property or dimension when performing record searches. By default, a record search query on a search interface returns results that match any of the properties or dimensions in the interface.

About implementing search interfaces

You implement search interfaces in Developer Studio's Search Interface editor.

Before implementing search interfaces, make sure that all the properties or dimensions that are going to be included in a search interface have already been enabled for record search.

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).

After indexing the data with the new search interface, the new key may be used for record searches.

Options for allowing cross-field matches

The **Allow Cross-field Matches** is one of the attributes in the **Search Interface editor** in Developer Studio. This attribute specifies when the MDEX Engine should try to match search queries across dimension or property boundaries.

The three settings for **Allow Cross-field Matches** are:

Setting	Description
Always	The MDEX Engine always looks for matches across dimension or property boundaries, in addition to matches within a dimension or property. If you choose to use cross-field matching, the **Always** setting is recommended and is the default. For example, in the `Sony camera` user query, if **Allow Cross-field Matches** is set to **Always**, the MDEX Engine returns all matches with `Brand = Sony` and `Prod¬uct_Type = camera`.
Never	The MDEX Engine does not look across boundaries for matches.
On Failure	The MDEX Engine only tries to match queries across dimension or property boundaries if it fails to find any matches within a single dimension or property.  **Note:** In most cases, the Always setting provides better results than the On Failure setting.

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` properties. Searching for `deniro` against this interface returns the union of records that results from searching for `deniro` against the `actor` property and against the `director` property.

Less frequently, you may wish to allow a match to span multiple properties and dimensions. For example, in the same `MoviePeople` search interface, a query for `clint eastwood` returns records where either an `actor` property or a `director` property 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 attribute in the **Search Interface** editor in Developer Studio, when you create a search interface. The **Allow Cross-field Matches** attribute specifies when the MDEX Engine should try to match search queries across dimension or property 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 dimension or property) is multi-assigned on a record, the multi-assigns are treated by the MDEX Engine as separate matches, just as if they were values from different properties. A search that matches two or more terms in separate multi-assign values for the same property is treated as a cross-field match by the MDEX Engine.

For example, assume a record has the following property 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 property (P_Tag).

Additional search interfaces options

You can configure additional features for the search interface by specifying other match-related options in the **Search Interface** editor in Developer Studio.

For example, you can specify the following options:

- A relevance ranking strategy that is associated with a search interface.
- Partial matching, which allows matches on subsets of the query.
- Complex Boolean search queries.

Search interfaces and URL query parameters (Ntk)

Use the name of the search interface as the value for the `Ntk` parameter, just as you would use a normal property or dimension.

No additional MDEX Engine URL query parameters are required to perform a record search using a search interface.

By default, using a search interface in a search performs a logical `OR` on the properties/dimensions in the interface.

For example, if a data set contains both an `Actor` property and `Director` dimension, a search interface can provide the user the ability to search for a person's name in both.

In this example, a search on the `MoviePeople` search interface returns records that match the `Actor` property OR the `Director` property.

The following two queries are not equivalent:

```
Ntk=actor|director&Ntt=deniro|deniro
Ntk=moviepeople&Ntt=deniro
```

- The first query performs a logical AND. This query only returns records where `actor` AND `director` contain `deniro`.
- The second query performs a logical `OR`.

Note: The `Nrk` URL parameter also requires a search interface.

Java examples of search interface methods

To obtain a list of valid search interfaces in Java, use the `Navigation.getERecCompoundSearchKeys()` method.

The following example shows how the `Navigation.getERecCompoundSearchKeys()` method can be used to obtain a list of search interface keys:

```
ERecCompoundSearchKeyList keylist =
 nav.getERecCompoundSearchKeys();
for (int i=0; i < keylist.size(); i++) {
// Get specific search interface key
  ERecCompoundSearchKey key = keylist.getKey(i);
  String name = key.getName();
  boolean active = key.isActive();
}
```

Note: Search interface keys are not returned in calls to the `Navigation.getERecSearchKeys()` method, which returns only basic record properties and dimensions.

.NET examples of search interface properties

To obtain a list of valid search interfaces in .NET, use the `Navigation.ERecCompoundSearchKeys` property.

The following example shows how the `Navigation.ERecCompoundSearchKeys` property can be used to obtain a list of search interface keys:

```
ERecCompoundSearchKeyList keylist = nav.ERecCompoundSearchKeys;
for (int i=0; i < keylist.Count; i++) {
// Get specific search interface key
 ERecCompoundSearchKey key =
 (ERecCompoundSearchKey) keylist.Key(i);
 String name = key.Name;
 boolean active = key.IsActive();
}
```

Note: Search interface keys are not returned in calls to the `Navigation.ERecSearchKeys` property, which returns only basic record properties and dimensions.

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 17

Using Dimension Search

There are two types of dimension search, default dimension search and compound dimension search.

About dimension search

Both default dimension search and compound dimension search allow users to perform keyword searches across dimensions for dimension values with matching names.

The result of a dimension search is a dimension search results object that contains dimension values.

The application can present these dimension values to the end-user, allowing the user to select them and create a new navigation request.

Depending on the type of dimension search you are using, those dimension values may be organized by:

- Dimension (default dimension search)
- Sets of dimension values (compound dimension search)

All configuration settings described for the dimension search are performed in the Developer Studio.

Default dimension search

Default dimension search returns a list of dimension values, organized by dimension, that match the user's search terms.

A dimension value must match all of a user's search terms to be considered a valid result when using default dimension search.

Example of default dimension search

For example, a default dimension search for `red` might return:

Dimension	Dimension values
`Wine_type`	`Red`
`Wineries`	`Green & Red, Red Hill, Red Rocks`

Dimension	Dimension values
Drinkability	Drink with red meat

Compound dimension search

Compound dimension search allows the MDEX Engine to return combinations of dimension values, called navigation states, that match a search query (in addition to single dimension values).

For example, the compound dimension search query:

```
1996 + merlot
```

could return a result such as:

```
{Year: 1996, Varietal: Merlot}
```

Note: Compound dimension search reduces to default dimension search for single-term queries, because any navigation state that minimally covers a single-term query will contain only one dimension value.

Compound dimension search results are navigation states that satisfy the following three properties:

- Validity. A navigation state is valid if it leads to actual records.

  For example, the navigation state `{Year: 1996, Varietal: Cabernet}` is valid if, and only if, there is at least one record that is assigned both dimension values.

- Coverage. A navigation state covers a query if the union of its dimension values accounts for all of the terms in the query, possibly by way of query expansion (such as stemming, thesaurus, or spelling correction).

  In other words, each dimension value in the navigation state must match at least one of the search terms. (We assume here that the query mode is **MatchAll**. The semantics for other match modes are discussed in other topics.)

  For example, the navigation state `{Year: 1996, Varietal: Cabernet}` is not a cover for the query `1996 + merlot`, because the query term `merlot` is not accounted for by any of its dimension values.

- Minimalism. A navigation state is a minimal cover of the query if removing any of its dimension values would cause it to no longer cover as many query terms.

  For example, the navigation state `{Year: 1996, Varietal: Merlot, Flavor: Oak}` is a cover, but it is not a minimal cover, because removing the dimension value `Flavor: Oak` leaves us with a cover.

Enabling dimensions for dimension search

The dimension values are enabled for the dimension search differently, depending on the type of the dimension search that you use.

In particular:

- Default dimension search.

  All dimensions are always enabled for the default dimension search. That is, all dimensions are searched by the MDEX Engine in the default dimension search.

  Unlike record search (which is disabled by default and therefore must be configured), there are no special configuration settings necessary to enable all dimensions for the default dimension search.

- Compound dimension search.

  If you use the `--compoundDimSearch` flag for Dgidx, all dimensions are enabled for the compound dimension search, that is they are searched by the MDEX Engine in the compound dimension search.

  In addition, you must set a Boolean flag on the `ENEQuery` object using these methods:

  - Java: `setDimSearchCompound()` method
  - .NET: `DimSearchCompound` property

Ordering of dimension search results

Dimension search results are ordered differently, depending on whether you have used the default dimension search or compound dimension search.

Ordering of results for default dimension search

The ordering of dimensions is determined by the statically defined dimension ranks.

Default dimension search results consist of dimension values grouped by dimension.

The ordering of dimension values, within each dimension, is based either on static dimension value ranks or on relevance ranking, if the latter is enabled.

Note: Relevance ranking must be explicitly requested (`Dk=1`) in order for the MDEX Engine to return ranked results rather than alphabetically sorted results. For more information, see the topic "Ranking results" later in this chapter.

Example of ordering results for default dimension search

In this example:

Dimension	Dimension values
`Wine_type`	`Red`
`Wineries`	`Green & Red, Red Hill, Red Rocks`
`Drinkability`	`Drink with red meat`

the `Wine_Type` dimension has a rank of 30, `Wineries` is ranked 20, and `Drinkability` is ranked 10.

The dimension values in the `Wineries` dimension are ranked as follows:

- `Green & Red` dimension value has a rank 3.
- `Red Hill` is ranked 2.
- `Red Rocks` is ranked 1.

Ordering of results for compound dimension search

This topic explains how compound dimension search results are ordered and contains examples of ordering.

Compound dimension search results are sets of dimension values that represent navigation states.

Technically, these groups are multisets, because a multiselect-AND dimension may be listed more than once in the set. For example, the navigation state `{Actor: Steve Martin, Actor: Goldie Hawn}` is listed in the `{Actor, Actor}` group.

The sets are ordered according to the following criteria:

- The primary sort is the number of dimensions represented in the navigation state. The fewer the number of dimensions, the higher the rank.

  For example, a result with dimension values from two dimensions would be returned before one that contained results from three.

- The secondary sort is lexicographical (alphanumeric), based on dimension ranks. The ordering of dimension values within each navigation state is based either on static dimension ranks (again lexicographic) or on relevance ranking, if the latter is enabled.

Example of ordering compound dimension search results

For example, consider a compound dimension search whose results are placed in the following groups:

```
{Actor}
{Director}
{Actor, Director}
{Actor, Director, Genre}
{Director, Genre}
{Title}
```

Assume that the static dimension ranks correspond to alphabetical order:

```
Actor < Director < Genre < Title
```

The compound dimension search result groups are ordered as follows:

```
{Actor}
{Director}
{Title}
{Actor, Director}
{Director, Genre}
{Actor, Director, Genre}
```

Filtering results that have no records

You can filter out unused dimension values from your dimension search results in the MDEX Engine at query time.

Dimension search can return dimension values that have no associated records. Depending on your application, you may not want your users to see such dimension search results. In such cases, you can filter out unused dimension values, using the dimension search ability to search within a navigation state.

You can do this in two ways:

- Call these method and property, passing in a `DimValIdList` consisting only of the value `0` (zero):
  - Java: the `ENEQuery.setDimSearchNavDescriptors()` method
  - .NET: the `ENEQuery.DimSearchNavDescriptors property`
- Use the `Dn` URL query parameter, setting the value to zero.

  In other words, instead of performing the query:

  ```
  D=Hampton+Bays
  ```

  use the query:

  ```
  D=Hampton+Bays&Dn=0
  ```

  You can code this into your application by adding `&Dn=0` any time you set the dimension search query. Because the work is done in the MDEX Engine, no UI modification to suppress results is required.

Advanced dimension search parameters

Advanced dimension search parameters give an application greater control over the matching dimension values returned. Standard dimension search returns all matching dimension values across all dimensions.

Advanced dimension search parameters allow the application to do the following:

- Request only the first n dimension values for each dimension. An additional parameter allows you to page through any additional matching dimension values after displaying the first n dimension values.
- Specify a single dimension within which to search.
- Restrict dimension search to searching within a given navigation state. The MDEX Engine returns only those matching dimension values that, when used to refine the specified navigation state, create a valid navigation request.

Disabling dimension search for synonyms

In some cases, you may decide that the text associated with a particular synonym is not appropriate for producing dimension search results.

Enabling hierarchical dimension search

By default, a dimension search considers only the text in individual dimension value synonyms when performing query matching. If you want dimension search to consider ancestor dimension values when matching a dimension search query, you must enable hierarchical dimension search in Developer Studio.

Returning the highest ancestor dimension

In the **Dimension Search Configuration** editor in Developer Studio you can specify that the results of a dimension search return only the highest ancestor dimension value.

For example, if both `red zinfandel` and `red wine` match a search query for `red` and you check **Return Highest Ancestor Dimension**, only the `red wine` dimension value is returned (assuming the `red wine` is the ancestor of `red zinfandel`). If the setting is not checked, then both dimension values are returned.

Searching inert dimension values

If **Include Inert Dimension Values** is checked in the **Dimension Search Configuration** editor in Developer Studio, then certain non-navigable dimension values (such as dimension roots) are also returned as the result of a dimension search query.

Collapsible dimension values (that is, dimension values that have their **COLLAPSIBLE** attribute set to `TRUE` within a `DVAL_REF` element) are never returned by dimension search.

Related Links

Adding search synonyms to dimension values on page 163
You can add synonyms to a dimension value so that users can search for other text strings and still get the same records as a search for the original dimension value name.

Dgidx flags for dimension search

Depending on the type of dimension search you use (default or compound dimension search), Dgidx requires different settings.

Dgidx flags for default dimension search

To make all dimension values available for the default dimension search, Dgidx does not require special flags. If a dimension value is properly created and used to classify a record in the data set, it is automatically indexed and enabled for the default dimension search.

Although all dimension values are enabled for the default dimension search, you can prevent certain dimension values from being added to the dimension search index, by filtering results with dimension values that have no associated records.

You can also limit the default dimension search to one dimension by using the `Di` parameter and specifying a single dimension for it.

Dgidx flags for compound dimension search

To make dimension values available for the compound dimension search, run the indexing using the `--compoundDimSearch` flag for Dgidx. Otherwise, compound dimension search will not be used by the MDEX Engine.

Although all dimension values are enabled for the compound dimension search if the `--compound¬DimSearch` flag is used for Dgidx, you can limit the compound dimension search to a list of dimensions, by using the `Di` parameter and specifying a list of dimension value IDs for it.

Note: Do not confuse indexing for dimension search with the Dgidx flags necessary to enable record search.

URL query parameters and dimension search

While a basic dimension search can be executed with a single parameter, an advanced dimension search query can have many different modifiers to control the resulting dimension values returned. This section contains examples of using these parameters.

As a rule of thumb, for any dimension that could contain more than 100 possible results, use one of the more advanced dimension search parameters to help control the results returned from the MDEX Engine. Without these controls, the size of the resulting object could cause slow response times between an application and the MDEX Engine.

Creating a default dimension search query

A default dimension search query contains a single parameter, `D` that specifies the keyword(s) to search with.

Each keyword can be plus- or space-delimited and should be URL encoded.

For example:

```
D=<string>+<string>…
```

Without any additional query modifiers, this dimension search is performed across all dimensions, and any/all matching dimension values in any/all dimensions (including hidden dimensions) are returned.

To create a default dimension search query:

Create a query of this type with the `D` parameter: `D=<string>+<string>….`
For example, create a query:

```
D=red
```

This query returns the following results, even if the `Wineries` dimension is hidden:

Dimension	Dimension values
`Wine_type`	`Red`
`Wineries`	`Green & Red, Red Hill, Red Rocks`
`Drinkability`	`Drink with red meat`

Creating a compound dimension search query

Compound search queries use the same dimension search URL parameters as default dimension search queries (`D`, `Dn`, `Di`, and so forth). Enabling and creating a compound dimension search query is a three-step process.

To enable and create a compound dimension search query:

1. Specify the `--compoundDimSearch` flag when running Dgidx.
2. Call the following method (Java) or property (.NET), before submitting the query:

Platform	Method or property
Java	`ENEQuery.setDimSearchCompound()`
.NET	`ENEQuery.DimSearchCompound`

3. Build the dimension search query using the same dimension search URL parameters as a default dimension search query (`D`, `Dn`, `Di`, and so forth).

Example query with a compound dimension search

The following is an example of a compound dimension search query (assuming the above three-step process is performed to enable this query).

This query:

```
D=red+1996
```

returns the following results:

Dimension	Dimension values
`Wine_Type, Year`	`[Red, 1996]`
`Wineries, Year`	`[Green & Red, 1996], [Red Hill, 1996]`

Note: Only valid navigation requests are returned as results. This example implies that there are `1996` wines from `Green & Red`, and from `Red Hill`, but not from `Red Rocks`.

Returning all possible dimension values in a dimension search

There may be limited use cases where you want create a query that returns all dimension values in all dimensions. In this case, you can specify `*` (an asterisk) as the string value to the Dimension (`D`) parameter, for example, `D=*`. This feature does not require you to select "Enable wildcard search" for the dimension values to be returned in the query. The parameter `D=*` is compatible with other additional dimension search parameters such as `Di` and `Dn`.

Limiting results of default dimension search and compound dimension search

Dimension search queries, either default dimension search or compound dimension search, could potentially contain many results. You can limit the number of returned results by using the Search Dimension (`Di`) parameter.

The `Di` parameter depends on the Dimension Search (`D`) parameter.

As a general rule, if a dimension could contain more than 100 possible results, use one of the more advanced dimension search parameters to help control the results returned from the MDEX Engine. Without these controls, the size of the result object could cause slow response times between an application and the MDEX Engine.

To limit the results of either a dimension search or compound dimension search:

In a query, specify a list of dimension IDs separated by plus signs (+) for the value of the `Di` parameter. The order of the IDs is unimportant.

For default dimension search, the results are limited to the specified dimension IDs. For compound dimension search, every result returned has exactly one value from each dimension ID specified in `Di`. This restricts a compound dimension search to the intersection of the specified dimensions (as opposed to the compound dimension search across all dimensions).

Example of a compound dimension search query

For example, the following compound dimension search query limits the number of returned results.

In this query, the `Winery` dimension has an ID of `11` and the `Year` dimension has an ID of `12`:

```
D=red+1996&Di=11+12
```

This query returns only the following results:

Dimension	Dimension values
Wineries, Year	[Green & Red, 1996], [Red Hill, 1996]

Setting the number of results

Another way to limit dimension search results (in addition to using the `Di` parameter only) is to limit the number of dimension values to return with each dimension, using the `Dp` parameter.

The `Dp` parameter depends on the Dimension Search (`D`) parameter.

To limit the number of dimension values to return with each dimension:

1. In a query, specify the `Dp` parameter and an integer value that represents the maximum dimension value count to return.

   This parameter takes an integer and when paired with the basic Dimension Search parameter (`D`), returns only the first n values from each dimension.

   For example, the following query:

   ```
   D=red&Dp=1
   ```

   returns only the following results:

Dimension	Dimension values
Wine_type	Red
Wineries	Green & Red
Drinkability	Drink with red meat

2. Optionally, use the Dimension Value Count parameter (`Dp`) with the Search Dimension parameter (`Di`), in which case only the first n dimension values for the specific dimension are returned.

   For example, the following query that contains a dimension search where the `Winery` dimension has an ID of `11`:

   ```
   D=red&Dp=1&Di=11
   ```

returns only the following results:

Dimension	Dimension values
Wineries	Green & Red

Note: If you are using default dimension search and specify more than one dimension ID for the Di parameter, then the Dp parameter is ignored.

Enabling result paging

To enable an application to page through dimension search results, use the Dimension Value Count parameter (Dp) in conjunction with the Search Results Offset parameter (Do).

To enable paging through the dimension search results:

1. Use the Do parameter, Do=int , where int is an integer.

   This allows an application to view n dimension search results at a time.

   For example, for n=5, the first query asks for only five results with no offset, the second query in the page set asks for five results with an offset of five, the third query asks for five results with an offset of ten, and so on.
2. (Optional but recommended). Use the Search Results Offset parameter (Do) in conjunction with both the Dp and Di parameters.

   Similar to other advanced dimension search parameters, the Search Results Offset parameter (Do) is fundamentally dependent on the Dimension Search parameter. Although it is not strictly enforced, the Search Results Offset parameter is most frequently used in conjunction with both the Dp and Di parameters.

   For example, the following dimension search query with these parameters:

   ```
   D=red&Dp=1&Di=11&Do=2
   ```

   returns only the following results:

Dimension	Dimension values
Wineries	Red Rocks

Ranking results

To rank the results of the default dimension search, use the Dk parameter.

To rank the results of the default dimension search:

Use the Dk parameter.

This simple ranking rule, when applied to the results of a default dimension search, enforces a dynamic order on the dimension values.

The dimension search ranking rule favors a combination of exact matches and frequency.

For example,

```
Dk=0 or 1
```

By default, matching dimension values are returned in the order that they would appear in the dimension for refining a navigation request.

It is important to note that this ranking rule is not the same as the more extensive ranking rules used to modify a record search request.

Note: Compound dimension search results cannot be dynamically ranked, so the `Dk` parameter is ignored for compound search results.

Searching within a navigation state

To limit a search to only valid dimension values within results of dimension search, use the Dimension Search Scope parameter, `Dn`.

The Dimension Search Scope parameter (`Dn`) is useful in conjunction with the other dimension search parameters to limit a search to only valid dimension values that can be combined with a specified navigation request to form a valid refinement request.

This is different from specifying a single dimension to search within. Think of this as a search within results for dimension search.

To search within a navigation state:

Use the Dimension Search Scope parameter (`Dn`).

For example:

```
Dn=<dimension value id>+<dimension value id>
```

For example, in this configuration:

Dimension	Dimension values
`Wine_type`	`Red`
`Wineries`	`Green & Red, Red Hill, Red Rocks`
`Drinkability`	`Drink with red meat`

if neither the `Red Rocks` nor the `Red Hill` winery dimension values are valid refinements for the `Wine Types: Red Wine` navigation query, then the following query:

```
D=red&Dn=40
```

where the `Red Wine` dimension value has an ID of `40`, returns only the dimension `Wineries` and the dimension values `Green & Red`.

Methods for accessing dimension search results

To access dimension search results, use`ENEQueryResults. containsDimensionSearch()` (Java) and `ENEQueryResults. ContainsDimensionSearch()`, as shown in examples in this topic.

If a valid dimension search request has been made, the following method calls for the query result object will evaluate to `true`:

- Java: `ENEQueryResults. containsDimensionSearch()` method call
- .NET: `ENEQueryResults. ContainsDimensionSearch()` method call

However, regardless of how the dimension search request is created to control the number of dimension value results returned, the same objects and methods are used to access those results.

Any matching dimension values are organized by dimension (or dimension list, in the compound dimension search case), and each specific match contains methods to access other values that describe the hierarchy of that dimension value within the dimension.

For this reason, the results are actually dimension locations instead of dimension values. Dimension locations contain a single dimension value, as well as a list of ancestor dimension values.

For example, if a resulting dimension value is `merlot`, it will not only be returned in the `Wine Types` dimension, but it will be contained in a dimension location that contains the dimension value `red`, because `red` is an ancestor of `merlot`.

Java example

The following code sample in Java shows how to access dimension search results:

```
ENEQuery usq = new ENEQuery(request.getQueryString(), "UTF-8");
// Set query so that compound dimension search is enabled
usq.setDimSearchCompound(true);
ENEQueryResults qr = nec.query(usq);
// If query results object contains dimension search results
if (qr.containsDimensionSearch()) {
 // Get dimension search results object
 DimensionSearchResult dsr = qr.getDimensionSearch();
 // Get results grouped by dimension groups
 DimensionSearchResultGroupList dsrgl = dsr.getResults();
 // Loop over result dimension groups
 for (int i=0; i < dsrgl.size(); i++) {
  // Get individual result dimension group
  DimensionSearchResultGroup dsrg =
    (DimensionSearchResultGroup)dsrgl.get(i);
  // Get roots for dimension group
  DimValList roots = dsrg.getRoots();
  // Loop over dimension group roots
  for (int j=0; j < roots.size(); j++) {
   // Get dimension root
   DimVal root = (DimVal)roots.get(j);
   // Display dimension root
   %><%= root.getName() %><%
  }
  // Loop over results in group
  for (int j=0; j< dsrg.getTotalNumResults(); j++) {
   // Get individual result
   DimLocationList dll = (DimLocationList)dsrg.get(j);
   // Loop over dimlocations in result
   for (int k=0; k<dll.size(); k++) {
    // Get individual dimlocation from result
    DimLocation dl = (DimLocation)dll.get(k);
    // Get ancestors list
    DimValList ancs = dl.getAncestors();
    // Loop over ancestors for results
    for (int l=0; l < ancs.size(); l++) {
     // Get ancestor and display its name
```

```
     DimVal anc = (DimVal)ancs.get(l);
     %><%= anc.getName() %> > <%
    }
     %><%= dl.getDimValue().getName() %><%
   }
  }
 }
}
```

.NET example

The following code sample in .NET shows how to access dimension search results:

```
ENEQuery usq = new ENEQuery(queryString, "UTF-8");
// Set query so that compound dimension search is enabled
usq.DimSearchCompound = true;
ENEQueryResults qr = nec.Query(usq);
// If query results object contains dimension search results
if (qr.ContainsDimensionSearch()) {
 // Get dimension search results object
 DimensionSearchResult dsr = qr.DimensionSearch;
 // Get results grouped by dimension groups
 DimensionSearchResultGroupList dsrgl = dsr.Results;
 // Loop over result dimension groups
 for (int i=0; i < dsrgl.Count; i++) {
  // Get individual result dimension group
  DimensionSearchResultGroup dsrg =
    (DimensionSearchResultGroup)dsrgl[i];
  // Get roots for dimension group
  DimValList roots = dsrg.Roots;
  // Loop over dimension group roots
  for (int j=0; j < roots.Count; j++) {
   // Get dimension root
   DimVal root = (DimVal)roots[j];
   // Display dimension root
   %><%= root.Name %><%
  }
  // Loop over results in group
  for (int k=0; k< dsrg.TotalNumResults; k++) {
   // Get individual result
   DimLocationList dll = (DimLocationList)dsrg[k];
   // Loop over dimlocations in result
   for (int m=0; m<dll.Count; m++) {
    // Get individual dimlocation from result
    DimLocation dl = (DimLocation)dll[m];
    // Get ancestors list
    DimValList ancs = dl.Ancestors;
    // Loop over ancestors for results
    for (int n=0; l < ancs.Count; n++) {
     // Get ancestor and display its name
     DimVal anc = (DimVal)ancs[n];
     %><%= anc.Name %> > <%
    }
     %><%= dl.DimValue.Name %><%
   }
  }
 }
}
```

Displaying refinement counts for dimension search

A front-end application can display refinement counts for dimension values returned by a dimension search. Refinement counts can provide more context in an Endeca application by providing the user with the number of records (or aggregated records) associated with a given dimension value.

Enabling refinement counts for dimension search

You enable refinement counts for a dimension search on a per-query basis using the `Drc` (Refinement Configuration for Dimension Search) query parameter. No Developer Studio configuration or Dgraph flags are required to enable this feature.

For details about using `Drc`, see *Drc (Refinement Configuration for Dimension Search)* on page 267.

Retrieving refinement counts for dimension search

Record counts are returned in two Dgraph properties.

To retrieve the counts for regular (non-aggregated) or aggregated records beneath a given refinement (dimension value), use these Dgraph properties:

- Counts for regular (non-aggregated) records are returned as a property on each dimension value. For regular records, this property is `DGraph.Bins`.
- Counts for aggregated records are also returned as a property on each dimension value. For aggregated records, this property is `DGraph.AggrBins`.

For a given `DimensionSearchResult` object, request all dimension search results with:

- Java: `DimensionSearchResult.getDimensionSearch()` method
- .NET: `DimensionSearchResult.DimensionSearch` property

The dimension search results for a given dimension are returned in a `DimensionSearchResultGroup` object.

For each dimension value in the group, you can return a `DimLocation` object from a `DimLocationList`. You can then return the `DimVal` object with:

- Java: `DimValList.getDimValue()` method
- .NET: `DimValList.Item` property

To get a list of properties (PropertyMap object) associated with the dimension value, use:

- Java: `DimVal.getProperties()` method
- .NET: `DimVal.Properties` property

Calling the `PropertyMap.get()` method (Java) or `PropertyMap` object (.NET) at this point, with the `DGraph.Bins` or `DGraph.AggrBins` argument will return a list of values associated with that property. This list should contain a single element, which is the count of non-aggregated or aggregated records beneath the given dimension value.

Java example

This example gets the refinement counts for all the dimension values in a dimension that has an id of 800000.

```
if (results.containsDimensionSearch()) {
   DimensionSearchResult result = results.getDimensionSearch();
   DimensionSearchResultGroup group = result.getDimensionSearchResult¬
```

```
Group(800000);
   for (DimLocationList l : (List<DimLocationList>)group) {
      DimLocation loc = (DimLocation)l.get(0);
      DimVal dimVal = loc.getDimValue();
      PropertyMap pmap = dimVal.getProperties();
      String dstats = "";
      if (pmap.get("DGraph.Bins") != null) {
         dstats = "(" + pmap.get("DGraph.Bins") + ")";
      }
   }
}
```

.NET example

This example gets the refinement counts for all the dimension values in a dimension that has an id of 800000.

```
if (results.ContainsDimensionSearch()) {
   DimensionSearchResult result = results.DimensionSearch;
   DimensionSearchResultGroup group = result.GetDimensionSearchResult¬
Group(800000);
   for (int i = 0; i < group.Count; i++) {
      DimLocationList l = (DimLocationList)group[i];
      DimLocation loc = (DimLocation)l[0];
      DimVal dimVal = loc.DimValue;
      PropertyMap pmap = dimVal.Properties;
      String dstats = "";
      if (pmap["DGraph.Bins"] != null) {
         dstats = " ("+pmap["DGraph.Bins"]+")";
      }
   }
}
```

Performance impact of refinement counts for dimension search

Dimension search is generally not an expensive feature, and adding counts to a dimension search query adds only modest costs to query processing. However, there can be feature interaction issues that increase performance costs.

In particular, Type Ahead search may impose small but potentially noticeable performance costs. Remember that Type Ahead search performs dimension search queries for each character that an application user types. Adding refinement counts to each dimension search query slows overall performance because of rapid query processing. You may have to experiment to determine whether this performance cost has any noticeable impact for your application.

When to use dimension and record search

Dimension search is sometimes confused with record search. This topic provides examples of when to use each type of search.

Being clear about the differences between the two basic types of keyword search (record search and dimension search) is important before attempting to create a solution for a specific business problem. Use the following recommendations:

Type of keyword search	When to use
Dimension search	In general, datasets with little descriptive text and extensive dimension values that represent the most frequently searched terms (for example, `autos`) are a good fit for dimension search. Keyword searches are usually oriented towards such keywords, as for example, `make`, `model`, `year`, and so on, which would probably be included in the list of dimensions. For example, searching for `Ford` would return a single dimension value from the `Make` dimension.
Record search	Datasets with descriptive text or names (such as news articles) are better suited for record search. This is because a reasonable set of dimension values for such a dataset cannot be expected to cover all the terms required to handle keyword search. In such cases, record search allows an application to search directly against record text (such as the body of an article).

Note: Read the rest of this topic for additional recommendations.

For many commerce applications, a combination of dimension search and record search is actually the best solution. In this case, separate dimension search and record search queries are executed simultaneously for the same keywords, as demonstrated in the reference implementation:

- If a dimension value matches, the user is given the opportunity to select that dimension value in place of the record search query to produce results that have actually been classified.
- If no dimension values match, the user is still left with the matching records for a record search query.

Keep in mind that navigation queries and dimension 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 dimension search does not, and visa-versa.

For example, the following query is valid but contains two completely independent types of results:

```
N=40&D=red
```

In this query, the `ENEQueryResults.containsDimensionSearch()` method (Java), and the `ENEQueryResults.ContainsDimensionSearch()` method (.NET), as well as the `ENEQueryResults.containsNavigation()` method (Java), and the `ENEQueryResults.ContainsNavigation()` method (.NET) evaluate to `true` for the query results object.

The `Navigation` object is the same as if the query were only `N=40`. The dimension search results object is the same as if the query were only `D=red`. By that reasoning, the following query also contains two independent types of results:

```
N=40&Ntk=Name&Ntt=red&D=red
```

One final consideration in selecting what type of search solution to implement: Unless compound dimension search is enabled, dimension search is only used for finding a single dimension value. Therefore, multiple keywords are still used to find a single dimension value.

For example, `red+1996` returns the `Red` dimension value, and the `1996` dimension value. It only returns a single dimension value that matches both of those terms, if one exists.

Refer to the "Using Boolean Search" section for details on performing Boolean queries with dimension search, for example, `red+or+1996`, which returns both the `red` dimension value and the `1996` dimension value.

Compound dimension search is most appropriate where multiple terms are used to search for combinations of concepts, such as `D=red+1996`. Record search may also be appropriate, and is described in the section about record search.

Related Links

About Record Search on page 161
This section discusses record search, which is an Endeca equivalent of full-text search, and is one of the fundamental building blocks of Endeca search capabilities.

Performance impact of dimension search

This topic discusses dimension search and its impact on MDEX Engine performance.

Creating the additional index structures for compound dimension search may result in a moderate increase in indexing time, particularly if there are a large number of dimensions.

The runtime performance of dimension search directly corresponds to the number of dimension values and the size of the resulting set of matching dimension values. But 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 dimension values is exceptionally large (greater than 1,000), thus creating a large results page. This is when the advanced dimension search parameters should be used to limit the number of results per request.

Compound dimension search requests are generally more expensive than non-compound requests, and are comparable in performance to record search requests:

- If you submit a default dimension search query, the query is generally very fast.
- If you submit a compound dimension search query, performance is not as fast as for the default dimension search.

In both cases, the query will be faster if you limit the results by using any of the advanced dimension search parameters. For example, you can use the `Di` parameter to specify the specific dimension (in the case of the default dimension search), or a list of dimension value IDs (in the case of compound dimension search) for which you expect matches returned by the MDEX Engine.

Chapter 18

Record and Dimension Search Reports

The record and dimension search reports provide API-level access to summary information about search queries. This information includes the number of results, spelling suggestions, and query expansion useful for highlighting.

Implementing search reports

The search reports do not require any work in Developer Studio, and no Dgidx or MDEX Engine configuration flags are necessary to enable this feature. Moreover, there are no URL query parameters to enable search reports.

Methods for search reports

The MDEX Engine returns search reports as `ESearchReport` objects.

- For a dimension search, a single `ESearchReport` object is returned.
- For a record search, one `ESearchReport` object is returned for each search key.

Retrieving search reports

To retrieve search reports, use `getESearchReports()` methods (Java) and `ESearchReports` properties (.NET) on the `DimensionSearchResult` and `Navigation` classes.

Both the `DimensionSearchResult` and `Navigation` classes have `getESearchReports()` methods (Java), and `ESearchReports` properties (.NET) that return a `Map` (Java), and an `IDic¬tionary` (.NET) of search keys to `ESearchReport` objects. In the dimension search case, the single search report is associated with the literal string `Dimension Search`.

If, however, you have performed a multiple search (that is, using the `Ntk` and `Ntt` parameters with two or more search keys and terms), you can use the `getESearchReportsComplete()` method (Java), and the `ESearchReportsComplete` property (.NET) in the `DimensionSearchResult` and `Navigation` classes.

These accessors return a `Map` (Java) and an `IDictionary` (.NET) of `List` (Java) and `IList` (.NET) objects that contain `ESearchReports` objects.

Encapsulating the ESearchReports objects in a List (Java), or an IList (.NET) prevents multiple ESearchReports with the same key from overwriting each other, which can happen with the getESearchReports() method (Java) and ESearchReports property (.NET).

Accessing information in search reports

An ESearchReport object provides access to summary information about the search through accessor methods (Java), and properties (.NET). This topic contains code examples for accessing summary information in search reports.

The report provides basic information about the search through the following ESearchReport methods (Java), and properties (.NET):

Method (Java) or property (.NET)	Description
Java: getKey() NET: Key	Returns the search key used in the current search.
Java: getTerms() .NET: Terms	Returns the search terms as a single String.
Java: getNumMatchingResults() .NET: NumMatchingResults	Returns the number of results that matched the search query. For record searches, this is the number of records. For dimension searches, this is the number of matching dimension values.

Match mode information is available through the following ESearchReport methods (Java), or properties (.NET):

Method (Java) or property (.NET)	Description
Java: getSearchMode() NET: SearchMode	Returns the requested match mode.
Java: getMatchedMode() .NET: MatchedMode	Returns the selected match mode. This is different than getSearchMode() (Java) and SearchMode (.NET) in that getMatchedMode() (Java) and MatchedMode (.NET) return the match mode that was actually selected by the MDEX Engine as opposed to the match mode that was requested in the query.
Java: getNumMatchedTerms() .NET: NumMatchedTerms	Returns the number of search terms that were successfully matched.

Word interpretation information, which is useful for highlighting or informing users about query expansion, is available through the ESearchReport.getWordInterps() method (Java), and

`ESearchReport.WordInterps` property (.NET). The method and property return a `PropertyMap` that associates words or phrases with their expansions.

Spelling correction information is available through two `ESearchReport` methods (Java), and properties (.NET):

Method (Java) or property (.NET)	Description
Java: `getAutoSuggestions` NET: `AutoSuggestions`	Is used for autosuggest (alternate spelling correction) results and returns a `List` (Java), and an `IList` (.NET) of `ESearchAutoSuggestion` objects.
Java: `getDYMSuggestions` .NET: `DYMSuggestions`	Is used for "Did You Mean" results and returns a List an IList of `ESearchDYMSuggestion` objects.

The `ESearchAutoSuggestion`, and `ESearchDYMSuggestion` classes have `getTerms()` method (Java), and `Terms` property (.NET) that return the suggestion as a string.

The `ESearchDYMSuggestion` class also includes a `getNumMatchingResults()` method (Java), and `NumMatchingResults` property (.NET) that return the number of results associated with the "Did You Mean" suggestion. For more information on these features, see the section on the "Did You Mean" feature.

Finally, the following `ESearchReport` calls report error or warning information:

- The `getTruncatedTerms()` method (Java) and `TruncatedTerms` property (.NET) return the truncated query terms (as a single string), if the query was truncated. If the number of search terms is too large, the MDEX Engine truncates the query for performance reasons. This method or property return the new set of search terms after the truncation.
- The `isValid()` method (Java and .NET) returns `true` if the search query is valid.

  If `false` is returned, use `getErrorMessage()` (Java), and `ErrorMessage` (.NET) to get the error message.
- The `getErrorMessage()` method (Java), and `ErrorMessage` property (.NET) return the error message for an invalid query.

Java example

The following code snippet in Java shows how to access information in an `ESearchReport` object:

```
// Get the Map of ESearchReport objects
Map recSrchRpts = nav.getESearchReports();
// Declare the search key being sought
String desiredKey = "my_search_interface";
if (recSrchRpts.containsKey(desiredKey)) {
// Get the ERecSearchReport for the desired search key
 ESearchReport srchReport =
 (ESearchReport) recSrchRpts.get(desiredKey);
 // Get the search term submitted for this search report
 String srchTerms = srchReport.getTerms();
 // Get the number of matching results
 long numMatchingResults = srchReport.getNumMatchingResults();
 // Get the match mode that was used for this search
 ESearchReport.Mode mode = srchReport.getMatchedMode();
 // Display a message if MatchAll mode was used
 // by the MDEX Engine
```

```
 String matchallMessage = "";
 if (mode == ESearchReport.MODE_ALL) {
  matchallMessage = "MatchAll mode was used";
 }
}
```

.NET Example

The following code snippet in .NET shows how to access information in an `ESearchReport` object:

```
// Get the Dictionary of ESearchReport objects
IDictionary recSrchRpts = nav.ESearchReports;
// Declare the search key being sought
String desiredKey = "my_search_interface";
if (recSrchRpts.Contains(desiredKey)) {
// Get the ERecSearchReport for the desired search key
 ESearchReport srchReport =
 (ESearchReport) recSrchRpts[desiredKey];
 // Get the search term submitted for this search report
 String srchTerms = srchReport.Terms;
 // Get the number of matching results
 long numMatchingResults = srchReport.NumMatchingResults;
 // Get the match mode that was used for this search
 ESearchReport.Mode mode = srchReport.MatchedMode;
 // Display a message if MatchAll mode was used by
 // Navigation Engine
 String matchallMessage = "";
 if (mode == ESearchReport.MODE_ALL) {
  matchallMessage = "MatchAll mode was used";
 }
}
```

Troubleshooting search reports

The tokenization used for substitutions depends on the configuration of search characters.

If word interpretation is to be used to facilitate highlighting variants of search keywords that appear in displayed search results, then the application should consider that words or phrases appearing in substitutions may not include white space, punctuation, or other configured search characters.

Note: Search reports have no impact on performance.

Chapter 19

Using Search Modes

By default, Endeca search operations return results that contain text matching all user search terms. In other words, search is conjunctive by default. However, in some cases a less restrictive matching is desirable, so that results are returned that contain fewer user search terms. This section describes how to enable the MatchAny and MatchPartial modes for record search and dimension 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 dimension search query.

Valid search modes are the following:

Search mode	Description
MatchAll	Match all user search terms (that is, perform a conjunctive search). This is the default mode.
MatchPartial	Match some user search terms.
MatchAny	Match at least one user search term.
MatchAllAny	Match all user search terms if possible, otherwise match at least one. MatchAllAny 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, MatchAllPartial is a better strategy.
MatchAllPartial	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, MatchAllPartial is generally a better choice than MatchAllAny.
MatchPartialMax	Match a maximal subset of user search terms.

Search mode	Description
MatchBoolean	Match using a Boolean query.

MatchAll mode

In MatchAll mode (the default mode), results must contain text matching each user search query term.

MatchPartial mode

In MatchPartial 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 MatchPartial mode, results must contain text matching search query terms, according to the following rules:

- The **Match at least** 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 **Omit at most** setting specifies the maximum number of user query terms that can be ignored in the user query. If **Omit at most** value is set to zero, any number of words can be ignored.

You can specify both of these settings in Developer Studio.

In MatchPartial mode, result sets always include all of the results that a MatchAll query have produced, and possibly additional results as well.

Interaction of MatchPartial 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 MatchPartial mode is used. The example in this topic explains the interaction between stop words and MatchPartial mode.

The Endeca MDEX Engine 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 MatchPartial 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.

MatchAny mode

In MatchAny mode, results need only match a single user search term.

A MatchAny result set always includes all of the results that a MatchAll or MatchPartial query have produced, and possibly additional results as well.

Note: MatchAny is not recommended for use with record search in typical catalog applications.

MatchAllPartial mode

In MatchAllPartial mode, the MDEX Engine first uses MatchAll mode to return results matching all search terms, if any are available.

If no such MatchAll results are available, the MDEX Engine returns the results that MatchPartial would have produced. This allows a more conservative matching policy than MatchPartial, because high-quality conjunctive results are returned if they exist and MatchPartial 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: MatchAllPartial 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.

MatchAllAny mode

In MatchAllAny mode, the MDEX Engine first uses MatchAll mode to return results matching all search terms, if any are available.

If no such MatchAll results are available, the MDEX Engine returns the results that MatchAny would have produced.

Note: MatchAllAny is useful for dimension search.

MatchPartialMax mode

MatchPartialMax mode is a variant of the MatchAllPartial mode: MatchAll results are returned if they exist.

If no such MatchAll results exist, then results matching all but one terms are returned; otherwise, results matching all but two terms are returned; and so forth.

MatchPartialMax mode is subject to the **Match at least** and **Omit at most** settings used in the MatchPartial mode. Hence, a MatchPartialMax result set includes results if (and only if) the corresponding MatchPartial result set includes results, and it contains a subset of the MatchPartial results (possibly the entire set).

MatchBoolean mode

The MatchBoolean 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

This topic summarizes options you can use to implement search modes.

No Forge or Dgidx configuration is required to enable the MatchAll, MatchAny, or MatchAnyAll search modes. MatchPartial, MatchAllPartial, and MatchPartialMax are configured as URL query parameters. In Developer Studio, you configure the minimum number of words for partial match modes and maximum number of words that may be omitted for partial match modes.

No MDEX Engine configuration flags are necessary to enable search modes.

URL query parameters for search modes

You can use `Ntx` and `Dx` parameters with search modes. This topic contains code examples.

By using the following syntax, the search mode can be specified independently for each record search operation contained in a navigation query:

```
Ntx=mode+matchmode-1|mode+matchmode-2|...
```

where matchmode is the name of one of the search modes (such as matchallpartial).

The syntax for a dimension search query is similar:

```
Dx=mode+matchmode
```

Using the syntax above, each search query can be enabled for any of the listed modes.

Two sample queries are:

```
<application>?N=0&Ntk=Brand&Ntt=Nike+Adidas
&Ntx=mode+matchallany

<application>?D=Nike+sneakers&Dx=mode+matchany
```

Query examples with search modes

The MatchAny mode can be used in combination with multiple record searches to achieve Boolean-query effects using a simplified interface.

For example, the following query:

```
Ntk=Brand|Color&Ntt=Polo+Sport|red+blue&Ntx=mode+
matchall|mode+matchany
```

could be used to search for items with a `Brand` property matching `Polo` AND `Sport`, and with a `Color` property matching either `red` OR `blue`.

In some cases, it is useful to contrast the MatchAny versus MatchAll mode for combined record search and dimension search operations. For example, the following query in a movie database:

```
N=0&Ntk=AllText&Ntt=Gere+Roberts&D=Gere+Roberts&Dx=
mode+matchany
```

would return records matching both `Gere` AND `Roberts` (such as `Pretty Woman`), but would return dimension values containing either `Gere` OR `Roberts` (such as `Richard Gere` and `Julia Roberts`).

The MatchPartial mode can be thought of as being the union of several conjunctive queries. For example, if **Match At Least** and **Omit At Most** both have the default value of two in Developer Studio, then the following query:

```
N=0&Ntk=AllText&Ntt=brown+leather+jacket&Ntx=mode+matchpartial
```

would return records matching either `brown` and `leather`, or `leather` and `jacket`, or `brown` and `jacket`.

On the other hand, if **Match At Least** is one and **Omit At Most** is two, then the same query would return records matching either `brown` or `leather` or `jacket`—the same behavior as MatchAny.

Search mode methods

There are no objects types or method calls associated with search queries that use a match mode. Results returned are the same as for default MatchAll search queries.

Chapter 20

Using Boolean Search

This section describes how to enable Boolean search for record search and dimension search.

About Boolean search

The MatchBoolean 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.

Endeca search operations use the MatchAll mode by default, which results in conjunctive searches. However, users often want more precise control over their exact search query.

For example, there is no way to formulate the query that expresses the request: `"Show me all records that match either red or blue and also match the word car."`

For example, the query (`red` OR `blue`) AND `car` would express the request described above. The OR in this query is a disjunctive operator and results in a hit on all records that match 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 MDEX Engine.

Unlike the MatchAll and MatchAny modes, Boolean search also lets users specify negation in their queries.

For example, the query `camcorder AND NOT digital`will search for all Endeca records 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 MDEX Engine 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 property or dimension regardless of whether or not these properties or dimensions are included in the same search interface.

Related Links

Example of Boolean query syntax on page 208
The complete grammar for expressing Boolean queries, in a BNF-like format, is included in this topic.

Examples of using the key restrict operator on page 209
This topic uses examples to explain how to use the key restrict operator (`:`) in queries that contain Boolean search.

Example of 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 properties, `Actor` and `Director`, you can issue a query which involves a Boolean expression consisting of both the `Actor` and `Director` properties (for example, `"Search for records where the director was DeNiro and the actor does not include Pacino."`). The two properties 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 properties 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 property or dimension named `maker`, and `martin` against the specified search interface.

If the intention was to search against the property or dimension 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 property or dimension `car maker`, while it searches for `martin` against the specified search interface.

- `"car maker" : (aston martin)`

  This query does a conjunctive (MatchAll) search for the words `aston martin` against the property or dimension `car maker`.

- `"car maker" : "aston martin"`

  This query searches for the phrase `aston martin` against the property or dimension `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 subexpressions

This topic contains examples of using proximity operators with nested subexpressions.

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)
```

For other support limitations, see the topic about interaction of Boolean search with other features.

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 bicy¬cle)"`. 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	
Geospatial filters and 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	
Why did it match	Yes	
Word interp	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.`	
`Manufac¬tur¬er:(Sony OR Item: Camera)`	`The key restrict operator may not be used within another key restrict expression.`	
`Manufac¬turer:`	`Unexpected end of expression. The key restrict operator must be followed by a word, a phrase, or a left parenthesis.`	
`Manufac¬turer:OR`	`The key restrict operator must be followed by a word, a phrase, or a left parenthesis.`	
`Foo:Sony`	`Unknown search index name "Foo" used for restrict operator`	The search index name must exactly match the search index name used in the data.

Sample query	Error message	Comments
`Sony AND OR Aiwa`	`Expecting a term or phrase.`	Repeated operators are an error.

Implementing Boolean search

Except for proximity search, no Forge or Dgidx configuration is required to enable Boolean search mode.

Properties and dimensions should be configured appropriately for record search and/or dimension search as described in the documentation for those features.

There are no MDEX Engine configuration flags necessary to enable Boolean search mode.

URL query parameters for Boolean search

To specify a Boolean search query, use the `Ntx` (for record search), and `Dx` (for dimension search) URL query parameters.

- Record search.

  To specify a Boolean search for each record search operation contained in a navigation query, use the following URL query syntax with `Ntx`:

  ```
  Ntx=mode+matchboolean|...
  ```

- Dimension search.

  To specify a Boolean search for a dimension search query, use the following URL query syntax with `Dx`:

  ```
  Dx=mode+matchboolean
  ```

You can specify the search mode independently for each record search operation contained in a navigation query, and for the dimension search query.

Using the syntax above, you can enable each search query for MatchAll mode (which is the default if no mode is specified), MatchAny mode, or MatchBoolean mode. These are the mode definitions:

- In MatchAll mode, results must contain text matching each user search query term in at least one location.
- In MatchAny mode, results need only match a single user search term.
- In MatchBoolean mode, the results must satisfy the specified Boolean expression.

Additional examples of queries with Boolean search

The following are example queries:

```
<application>?N=0&Ntk=Brand&Ntt=Nike+or+Adidas
&Ntx=mode+matchboolean
```

```
<application>?N=0&Ntk=Title&Ntt=Japan+or+UK+not+USA
&Ntx=mode+matchboolean

<application>?D=solid+not+mahogany&Dx=mode+matchboolean
```

Methods for Boolean search

This topic contains examples of code in Java and .NET for obtaining Boolean search information in the `ESearchReport` object.

There are no object types or method calls associated with MatchBoolean search query processing. Results are returned the same as for default MatchAll search queries.

However, results returned by the MDEX Engine for MatchBoolean URL query parameters contain the following information in the Record Search Report supplement (`ESearchReport` object):

- Whether or not the Boolean query is valid. Use the `ESearchReport.isValid()` method to determine this.
- If the query is invalid, an error message is returned. Use `ESearchReport.getErrorMessage()` (Java), and `ESearchReport.ErrorMessage` (.NET) to obtain an error message (in English) that is suitable for display directly to the user.

Java example

The following code snippet in Java shows how to obtain the information in the `ESearchReport` object:

```
// Get the Map of ESearchReport objects
Map recSrchRpts = nav.getESearchReports();
if (recSrchRpts.size() > 0) {
 // Get the user's search key
 String searchKey = request.getParameter("Ntk");
 if (searchKey != null) {
  if (recSrchRpts.containsKey(searchKey)) {
  // Get the ERecSearchReport for the search key
  ESearchReport srchRpt =
   (ESearchReport) recSrchRpts.get(searchKey);
  // Check if the search is valid
   if (! srchRpt.isValid()) {
    // If invalid search, get the error message
    String errorMessage = srchRpt.getErrorMessage();
    // Print or log the message
    ...
   }
  }
 }
}
```

.NET Example

The following code snippet in .NET shows how to obtain the information in the `ESearchReport` object:

```
// Get the Dictionary of ESearchReport objects
IDictionary recSrchRpts = nav.ESearchReports;
// Get the user's search key
```

```
String searchKey = Request.QueryString["Ntk"];
 if (searchKey != null) {
  if (recSrchRpts.Contains(searchKey)) {
  // Get the ERecSearchReport for the search key
   ESearchReport srchRpt = (ESearchReport)
    recSrchRpts[searchKey];
  // Check if the search is valid
   if (! srchRpt.isValid()) {
    // If invalid search, get the error message
    String errorMessage = srchRpt.ErrorMessage;
    // Print or log the message
    ...
   }
  }
 }
}
```

Troubleshooting Boolean search

If you encounter unexpected behavior while using Boolean search, use the Dgraph `-v` flag when starting the MDEX Engine. 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 21

Using Phrase Search

Phrase search allows users to specify a literal string to be searched. This section discusses how to use phrase search.

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 MDEX Engine 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 index data out of word positions. This is called positional indexing.

Dgidx creates a positional index for both properties and dimension values.

Phrase search is automatically enabled in the MDEX Engine at all times. However, the default operation of phrase search examines potential matching text to verify the presence of the requested phrase query string. This examination process can be slow if the text data is large (perhaps containing long description property values) or offline (in the case of document text).

The MDEX Engine uses positional index data to improve performance in these scenarios. Positional indexing improves the performance of multi-word phrase search, proximity search, and certain relevance ranking modules. The thesaurus uses phrase search, so positional indexing improves the performance of multi-word thesaurus expansions as well. Positional indexing is enabled by default for Endeca properties and dimensions and cannot be disabled with Developer Studio.

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.

URL query parameters for phrase search

You can request phrase matching by enclosing a set of one or more search terms in quotation marks (ASCII character decimal `34`, or hexadecimal `0x22`). You can include phrase search queries in either record search or dimension search operations and combine phrase search with non-phrase search terms or other phrase terms.

Examples of phrase search queries

The following are examples of phrase search queries:

- A record searh for phrase `cd player` is as follows:
  ```
  N=0&Ntk=All&Ntt=%22cd+player%22
  ```
- A record searh for records containing phrase `cd player` and the word `sony` is as follows:
  ```
  N=0&Ntk=All&Ntt=%22cd+player%22+sony
  ```
- A record search for records containing phrase `cd player` and also phrase `optical output` is as follows:
  ```
  N=0&Ntk=All&Ntt=%22cd+player%22+%22optical+output%22
  ```
- A dimension search for dimension values containing the phrase `Samuel Clemens` is as follows:
  ```
  D=%22Samuel+Clemens%22
  ```

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 MDEX Engine within a few seconds (depending on hardware), and possibly faster on moderate sized data sets. Obviously, 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).

You can use the `--phrase_max <num>` flag for the Dgraph to specify the maximum number of words in each phrase for text search. Using this flag improves performance of text search with phrases. The default number is 10. If the maximum number of words in a phrase is exceeded, the phrase is truncated to the maximum word count and a warning is logged.

Chapter 22

Using Snippeting in Record Searches

This section describes how to use snippeting. 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 (also referred to as keyword in context or KWIC) provides the ability to return an excerpt from a record—called a snippet—to an application user who performs a record search query.

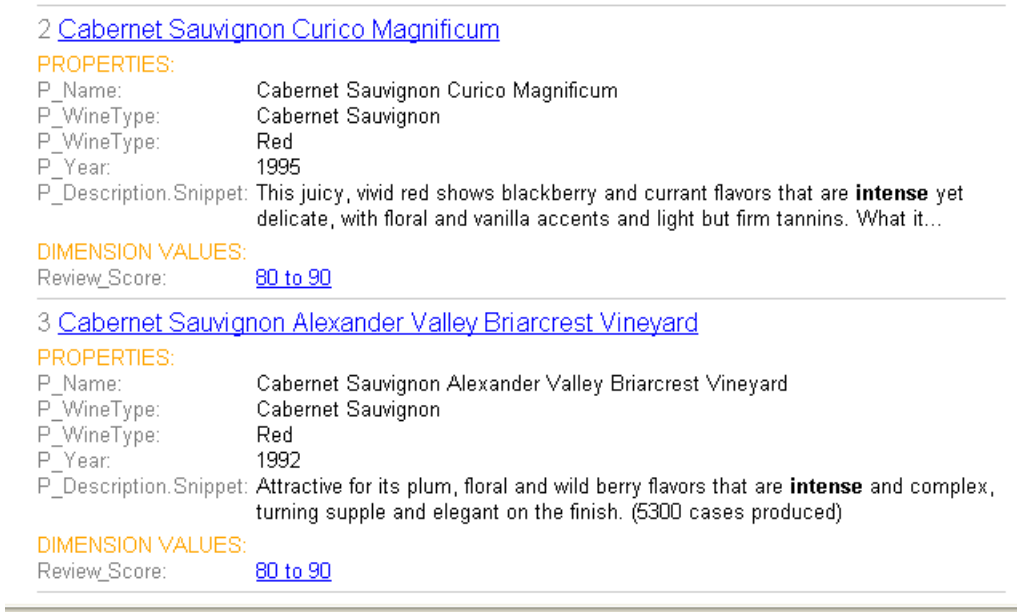
A snippet contains the search terms that the user provided along with a portion of the term's surrounding content to provide context. A Web application displays these snippets on the record list page of a query's results. With the added context, users can more quickly choose the individual records they 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.

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.

For example, if a user searches for `intense` in a wine catalog, the record list for this query has many records that match intense. A snippet for each matching record displays on a record list page:


2 Cabernet Sauvignon Curico Magnificum

PROPERTIES:

P_Name:	Cabernet Sauvignon Curico Magnificum
P_WineType:	Cabernet Sauvignon
P_WineType:	Red
P_Year:	1995
P_Description.Snippet:	This juicy, vivid red shows blackberry and currant flavors that are **intense** yet delicate, with floral and vanilla accents and light but firm tannins. What it...

DIMENSION VALUES:

Review_Score:	80 to 90

3 Cabernet Sauvignon Alexander Valley Briarcrest Vineyard

PROPERTIES:

P_Name:	Cabernet Sauvignon Alexander Valley Briarcrest Vineyard
P_WineType:	Cabernet Sauvignon
P_WineType:	Red
P_Year:	1992
P_Description.Snippet:	Attractive for its plum, floral and wild berry flavors that are **intense** and complex, turning supple and elegant on the finish. (5300 cases produced)

DIMENSION VALUES:

Review_Score:	80 to 90

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 `<endeca_term></endeca_term>` tags. The tags call out search terms and allow you to more easily reformat the terms for display in your Web application.

The snippet size is the total number of search terms and surrounding context words. 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 MDEX Engine 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 `<endeca_term>` tags.

```
...in Edinburgh <endeca_term>Scotland</endeca_term>, and has
been employed by Ford for 25 years...He first joined Ford's
<endeca_term>British</endeca_term> operation. Mazda motor...
```

Snippet property names

The MDEX Engine dynamically creates new snippet properties by appending `.Snippet` to the original name of the search interface members (fields) that you enabled for snippeting.

For example, if you enable snippeting for properties named Description and Reviews, the MDEX Engine creates new properties named `Description.Snippet` and `Reviews.Snippet` and returns these properties with the result set for a user's record search.

Snippets are dynamically generated properties

It is important to emphasize that the MDEX Engine dynamically generates snippet properties.

This means the snippet properties, unlike other Endeca properties, are not created, configured, or mapped using Developer Studio. A dynamically generated snippet property is not tagged to an Endeca record. The snippet property appears with a record only on a record list page.

About enabling and configuring snippeting

You enable the snippeting feature in the **Member Options** dialog box, which is accessed from the **Search Interface** editor in Developer Studio.

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 a dimension or property that has been enabled for search and that has been added to the Selected members pane of the Search Interface editor.

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** property for Search Interface A does not affect the **Description** property in Search Interface B.

URL query parameters for snippeting

You can configure snippeting on a per query basis by using the `Ntx` URL query parameter, the `snip` operator of `Ntx`, and key/value pairs that indicate which field to snippet and how many words to return in a snippet. This section contains examples of record search queries with snippeting.

Providing these values in a URL overrides any configuration options specified in a Developer Studio project file.

You can disable snippeting on a per query basis by using the `nosnip+true` operator of `Ntx`. The `nosnip+true` operator globally disables all snippets for any search interface member you enabled.

Examples of queries with snippeting

You can include snippeting only in record search operations. The following are examples of snippeting in queries:

- In a record search for records containing the word `blue`, snippet the `description` property with a maximum size of thirty words:

  ```
  N=0&Ntk=description&Ntt=blue&Ntx=snip+description:30
  ```

- In a record search for records containing the words `shirt` and `blue`, snippet the `title` property with a maximum size of ten words and the `description` property with a maximum size of thirty words:

  ```
  N=0&Ntk=title|description&Ntt=shirt|blue&Ntx=snip+title:10|snip+descrip¬
  tion:30
  ```

- In a record search for records containing the word `blue`, disable snippet results for the query:

  ```
  N=0&Ntk=description&Ntt=blue&Ntx=nosnip+true
  ```

Reformatting a snippet for display in your Web application

After the MDEX Engine returns a snippet property to your application, you can remove or replace the `<endeca_term>` tags from the snippet before displaying it in a record list page.

To reformat a snippet for display in a front-end Web application:

Add application code to replace the `<endeca_term>` tags in a snippet property with an HTML formatting tag, such as `<B>` (bold), to highlight search terms in a snippet.

Your Web application can display the snippet as a property on a record list page like other Endeca properties. For details, see the section about Displaying Endeca records.

Performance impact of snippeting

The snippeting feature does not have a performance impact during Data Foundry processing. However, enabling snippeting does affect query runtime performance.

There is no effect on Forge or Dgidx processing time or indexing space requirements on your hard disk.

You can minimize the performance impact on query runtime by limiting the number of words in a property that the MDEX Engine evaluates to identify the snippet. This approach is especially useful in cases where a snippet-enabled property stores large amounts of text.

Provide the `--snip_cutoff <num words>` flag to the Dgraph to restrict the number of words that the MDEX Engine evaluates in a property.

For example, `--snip_cutoff 300` evaluates the first 300 words of the property to identify the snippet.

Note: 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.

Tips and troubleshooting for snippeting

If a snippet is too short and you are not seeing enough context words in it, open the **Member Options** editor in Developer Studio and increase the value for **Maximum snippet size**. The default value is 25 words.

Chapter 23

Using Wildcard Search

Wildcard search allows users to match query terms to fragments of words in indexed text. This section discusses how to use wildcard search.

About wildcard search

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

Normally, Endeca search operations (such as record search and dimension 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 MDEX Engine 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	
Boolean search	Yes	
Snippeting	No	
Phrase search	No	
Why did it match	Yes	
Word interp	Yes	

Ways to configure wildcard search

You use Developer Studio to configure wildcard search in your application, using one of these dialogs: the **Dimension** and **Property** editors, the **Dimension Search Configuration** editor, and the **Search Interface** editor. The following topics provide details on these configuration options.

Configuring wildcard search with Dimension and Property editors

The **Dimension** and **Property** editors of Developer Studio allow you to enable wildcard search for any Endeca property or dimension.

Before you can enable wildcard search with **Dimension** and **Property** editors, you must first:

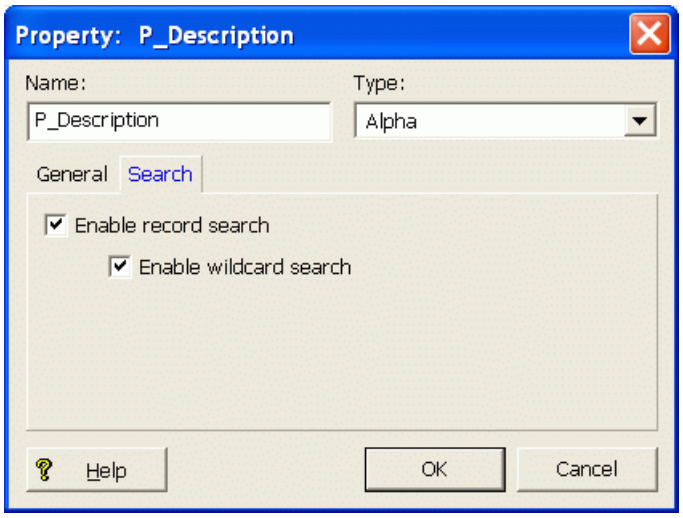
- Select the property or dimension for which you want to enable wildcard search.
- Check the **Enable Record Search** option in both editors for the specified Endeca property or dimension.

Note: If you use this method, you will only affect records enabled for search, but not dimensions enabled for search. (For dimensions enabled for search, you can enable wildcard search for ALL dimensions at once.)

To configure wildcard search in **Dimension** and **Property** editors:

1. In Developer Studio, go to **Dimension** or **Property** editor and select a **Search** tab.
2. In the **Search** tab, check **Enable Wildcard Search** option, as shown in the following example:

Note: This configuration affects only a single property or dimension that you have selected. For a dimension, it only affects record search for that dimension.



Configuring wildcard search with the Dimension Search Configuration editor

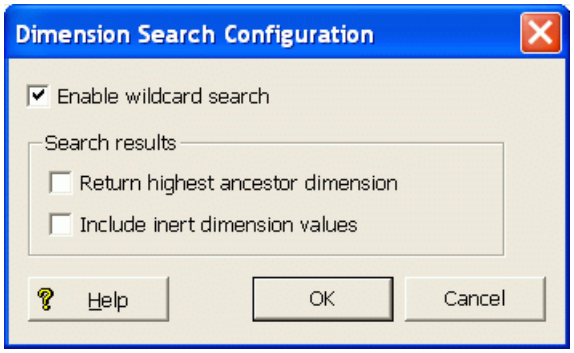
The **Dimension Search Configuration** editor in Developer Studio lets you configure wildcard search for all dimensions in your project.

Unlike the option for enabling wildcard search in the **Search** tab of the **Dimension** editor, which affects only a single dimension, the **Dimension Search Configuration** editor globally sets the options for all dimensions in a project.

Note: When you enable wildcard search for all dimensions in a project, this affects your results when you perform dimension search (that is, this does not apply to record search. For record search, you enable wildcard search per each property or dimension.)

To configure wildcard search with **Dimension Search Configuration** editor:

Check the **Enable Wildcard Search** option, as shown in the following example:



Configuring wildcard search with the Search Interface editor

You can enable wildcard matching for a search interface by adding one or more wildcard-enabled properties and dimensions to the search interface.

Use the **Search Interface** editor in Developer Studio to add the desired properties and dimensions. 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 property 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 with the **Search Interface** editor in Developer Studio:

1. Add the desired properties and dimensions to the search interface.
2. Enable wildcard search for members of the search interface.

   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.

Note: If you have a partially wildcard-enabled search interface, the MDEX Engine logs an informational message similar to the following example: `Search interface "MySearch" has some fields that have wildcard search enabled and others that do not. A wildcard search will behave differently when applied to wildcard enabled fields than when applied to other fields in this search interface (see the documentation for more details). Fields with wildcard indexing enabled: "Authors" "Titles" Fields with wildcard indexing disabled: "Price".` The message is only for informational purposes and does not affect the search operation.

MDEX Engine flags for wildcard search

There is no MDEX Engine configuration required to enable wildcard search. If wildcarding is enabled in Developer Studio, the MDEX Engine 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 MDEX Engine 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.

If a search query includes a wildcard expression that matches too many terms, the search returns results for the top frequent terms and the `is_valid` flag is set to `false` in the record search report.

To retrieve the error message, use the `ESearchReport.getErrorMessage()` method (Java), or `ESearchReport.ErrorMessage` property (.NET).

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. For more information on tuning this parameter, see the *Performance Tuning Guide*.

Other flags or attributes that existed in previous releases for tuning wildcard search are deprecated starting with the version 6.1.2 and ignored by the MDEX Engine.

Presentation API development for wildcard search

No specific Presentation API development is required to use wildcard search.

If wildcard search is enabled during indexing, users can enter search queries containing asterisk operators to request partial matching.

There are no special MDEX Engine URL parameters, method calls, or object types associated with wildcard search.

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 MDEX Engine (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 MDEX Engine 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, Endeca recommends using wildcard search queries with at least 2-3 non-wildcarded characters in them, such as `abc*` and

`ab*de`, and avoiding 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 `*.` or `* *`, are rejected by the MDEX Engine.

- **Analyze the format of your typical wildcard query cases**. This lets you be aware of performance implications associated with one specific wildcard search pattern.

  For example, it is useful to know that if search queries contain only wildcards and punctuation, such as `*.*`, the MDEX Engine rejects them for performance reasons and returns no results.

  Do you have queries that contain punctuation syntax in between strings of text, such as `ab*c.def*`?

  For strings with punctuation, the MDEX Engine generates lists of words that match each of the punctuation-separated wildcard expressions. Only in this case, the MDEX Engine 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. For more detailed information on this type of tuning, see the *Performance Tuning Guide*.

Note: You enable wildcard search in Developer Studio.

Chapter 24

Search Characters

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

Using search characters

The Endeca MDEX Engine 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 dimension values in dimension 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 Endeca MDEX Engine.
- Provides information about features supporting special handling for ISO-Latin1 and Windows CP1252 international characters during search indexing and query processing.

Note: Modifying search characters has no effect on Chinese, Japanese, or Korean language tokenization.

Query matching semantics

The semantics of matching search queries to text containing special non-alphanumeric characters in the Endeca MDEX Engine 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 Endeca system divides characters in indexed text into three categories:

- Alphanumeric characters including ASCII characters as well as non-punctuation characters in ISO-Latin1 and Windows CP1252.
- 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 properties for record search, dimension values for dimension search) is indexed based on the alternatives for handling characters from the three categories, which is described in subsequent topics.

Indexing alphanumeric characters

Alphanumeric characters are included in all forms.

Because Endeca 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 "Using Phrase Search" chapter.)
    - 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 MatchAll, a search for ice~cream would return non-contiguous results for [ice AND cream].

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 Search Characters editor in Developer Studio.

Note: Search characters are configured globally for all search operations. For example, adding the plus (+) character marks it as a search character for dimension search, record search, record search group, and navigation state search operations.

Dgidx flags for search characters

There are no Dgidx flags that are necessary to enable the search characters feature. Dgidx automatically detects the configured search characters.

Presentation API development for search characters

The search characters feature does not require any Presentation API development.

There are no relevant MDEX Engine parameters to control this feature, nor does this feature introduce any additional method calls or object types.

MDEX Engine flags for search characters

There are no MDEX Engine flags necessary to enable the search characters feature. The MDEX Engine automatically detects the additional search characters.

Chapter 25

Examples of Query Matching Interaction

The following examples of query matching interaction use record search, but the general matching concepts apply in all other search features supported by the MDEX Engine. The tables below illustrate the combined effects of various features by exposing text matches for given record search queries. In all cases we assume MatchAll search mode.

Record search without search characters enabled

In this example, the hyphen (-) is not specified as a search character.

In this table, 1 through 4 represent the text, while a through d represent the query.

	a) ice cream	b) ice-cream	c) icecream	d) "ice cream"
1. ice cream	Yes	Yes	If word-break analysis is used, this alternate form will be included for consideration as a spelling correction. It will be ranked for quality and considered alongside other results when the query is executed.	Yes
2. icecream	If word-break analysis is used, this alternate form will be included for consideration as a spelling correction. It will be ranked for quality and considered alongside other	If word-break analysis is used, this alternate form will be included for consideration as a spelling correction. It will be ranked for quality and considered alongside other	Yes	Yes

	results when the query is executed.	results when the query is executed.		
3. ice-cream	Yes	Yes	If word-break analysis is used, this alternate form will be included for consideration as a spelling correction. It will be ranked for quality and considered alongside other results when the query is executed.	Yes
4. cream ice	Yes. Note that by using Phrase relevance ranking, the priority of this text would be lowered.	No	No	No

Note: Keep in mind that although an alternate form is considered for spelling correction, the form will be discarded if the original terms return enough results.

Record search with search characters enabled

In this example, the hyphen (-) has been specified as a search character.

In this table, 1 through 4 represent the text, while a through d represent the query.

	a) ice cream	b) ice-cream	c) icecream	d) "ice cream"
1. ice cream	Yes	No	Yes, if word-break analysis is used.	Yes
2. icecream	Yes, if word-break analysis is used.	Yes, if espell is enabled and the --spellnum Dgidx option is enabled.	Yes	No
3. ice-cream	No	Yes	Yes, if espell is enabled and the --spellnum Dgidx option is enabled.	No

4. cream ice	Yes	No	No	No

Record search with wildcard search enabled but without search characters

In this example, the hyphen (-) has not been specified as a search character, and wildcards are used in the queries.

In this table, 1 through 4 represent the text, while a through e represent the query.

	a) ice crea*	b) ice-crea*	c) icecrea*	d) "ice crea*"	e) ic*rea*
1. ice cream	Yes	Yes	Yes, if word-break analysis is used.	No	No
2. icecream	Yes, if word-break analysis is used.	Yes, if word-break analysis is used.	Yes	No	Yes
3. ice-cream	Yes	Yes	Yes, if word-break analysis is used.	No	No
4. cream ice	Yes. Note that by using Phrase relevance ranking, the priority of this text would be lowered.	No	No	No	No

Record search with both wildcard search and search characters enabled

In this example, the hyphen (-) has been specified as a search character, and wildcards are used in the queries.

In this table, 1 through 4 represent the text, while a through e represent the query.

	a) ice crea*	b) ice-crea*	c) icecrea*	d) "ice crea*"	e) ic*rea*
1. ice cream	Yes	No	Yes, if word-break analysis is used.	No	No
2. icecream	Yes, if word-break analysis is used.	No	Yes	No	Yes
3. ice-cream	No	Yes	No	No	Yes
4. cream ice	Yes	No	No	No	No

Appendix A

Endeca URL Parameter Reference

This appendix provides a reference to the Endeca Presentation API's URL-based syntax for navigation, record, aggregated record, and dimension search queries.

About the Endeca URL query syntax

The Endeca query syntax defines how the client browser communicates with the Presentation API.

This appendix describes two methods:

- URL parameters
- ENEQuery setter methods (Java) and properies (.NET)

URL parameter description format

The tables in this appendix describe the Endeca query parameters, using the following characteristics:

Parameter	The query parameter, which is case-sensitive.
Name	The common name for the query parameter.
Java setter method	The corresponding `ENEQuery` Java setter method for the parameter.
.NET setter property	The corresponding `ENEQuery` .NET setter property for the parameter.
Type	The type of valid value for the query parameter.
Description	The basic MDEX result object that this parameter is associated with.
Object	A description of the query parameter, including information about its arguments.
Dependency	Additional query parameters that are required to give this parameter context.

In addition, an example of the query parameter use is given after the table.

About primary parameters

The following parameters are primary parameters:

- `N` (Navigation)
- `R` (Record)
- `A` (Aggregated Record)

- An (Aggregated Record Descriptors)
- Au (Aggregated Record Rollup Key)
- D (Dimension Search)

All other parameters are secondary. In order to use the secondary parameters in a query, you must include the primary parameters associated with that query type. For example, you cannot use a Dimension Search Scope (Dn) parameter without a Dimension Search (D) parameter

Note that the A, An, and Au parameters are mandatory for all aggregated record queries and must always be used together.

N (Navigation)

The N parameter sets the navigation field for a query.

Parameter	N
Name	Navigation
Java setter method	ENEQuery.setNavDescriptors()
.NET setter property	ENEQuery.NavDescriptors
Type	<dimension value id>+<dimension value id>+<dimension value id>...
Description	A unique combination of dimension value IDs that defines each navigation object. The root navigation object is indicated when zero is the only value in the parameter.
Object	Navigation
Dependency	none

Examples

```
/controller.php?N=0

/controller.php?N=132831+154283
```

Nao (Aggregated Record Offset)

The Nao parameter sets the navigation aggregated record list offset.

Parameter	Nao
Name	Aggregated Record Offset
Java setter method	ENEQuery.setNavAggrERecsOffset()
.NET setter property	ENEQuery.NavAggrERecsOffset
Type	int
Description	Specifies a number indicating the starting index of an aggregated record list. This parameter is similar to No (Record Offset) but for aggregated records.

Object	`Navigation`
Dependency	`N`, `Nu`

Examples

```
/controller.php?N=0&Nao=3&Nu=ssn

/controller.php?N=132831+154283&Nao=15&Nu=ssn
```

Ndr (Disabled Refinements)

The Ndr parameter lets you display disabled refinements.

Parameter	`Ndr`
Name	Disabled Refinements
Java setter method	`setNavDisabledRefinementsConfig`
.NET setter property	`NavDisabledRefinementsConfig`
Type	`<basedimid>+<textsearchesinbase>+<true/false>+<eqlfilter¬inbase>+<true/false><rangefiltersinbase>+<true/false>+...`
Description	Determines which dimension refinements are not available for navigation in the current navigation state but would have been available if the top-level navigation filters, such as previously chosen dimensions, range filters, EQL filters, text filters or text searches were to be removed from this navigation state. Configuration settings include: • `<basedimid>` — an ID of a dimension that is to be included in the base navigation state. • `<eqlfilterinbase>` — a true or false value indicating whether the EQL filter is part of the base navigation state. • `<textsearchesinbase>` — a true or false value indicating whether text searches are part of the base navigation state. • `<rangefiltersinbase>` — a true or false value indicating whether range filters are part of the base navigation state. When the `Ndr` parameter equals zero, no disabled refinement values are returned for any dimensions (which improves performance).
Object	`Navigation`
Dependency	`N`

Examples

The first example illustrates a query that enables disabled refinements to be returned. In this example, the `Ndr` portion of the UrlENEQuery URL indicates that:

- Text search should be included in the base navigation state.

- The navigation selections from the dimension with ID 100000 should be included in the base navigation state.

```
/graph?N=110001+210001&Ne=400000&Ntk=All&Ntt=television&Ndr=textsearchesin¬
base+true+basedimid+100000
```

In the second example of a query, in addition to text searches, the EQL filters and range filters are also listed (they are set to `false`):

```
N=134711+135689&Ntk=All&Ntt=television&Ndr=basedimid+100000+textsearchesin¬
base+true+eqlfilterinbase+false+rangefiltersinbase+false
```

Ne (Exposed Refinements)

The Ne parameter sets the dimension navigation refinements that will be exposed.

Parameter	`Ne`
Name	Exposed Refinements
Java setter method	`ENEQuery.setNavExposedRefinements()`
.NET setter property	`ENEQuery.NavExposedRefinements`
Type	<dimension value id>+<dimension value id>+<dimension value id>...
Description	Determines which dimension navigation refinements are exposed. When the `Ne` parameter equals zero, no refinement values are returned for any dimensions (which improves performance). When this parameter contains valid dimension value IDs, refinement values are only returned for that dimension.
Object	`Navigation`
Dependency	`N`

Examples

```
/controller.php?N=132831+154283&Ne=0

/controller.php?N=132831+154283&Ne=134711
```

Nf (Range Filter)

The `Nf` parameter sets the range filters for the navigation query.

Parameter	`Nf`
Name	Range Filter
Java setter method	`ENEQuery.setNavRangeFilters()`
.NET setter property	`ENEQuery.NavRangeFilters`

Type	<string>|[[LT|LTEQ|GT|GTEQ] <numeric value> |BTWN <numeric value> <numeric value>] <key>|[GCLT|GCGT|GCBTWN][+<geocode reference point>]+<value>[+<value>]
Description	Sets the range filters for the navigation query on properties, or for the navigation query on dimensions. Multiple filters are specified by a vertical pipe (|) delimiting each filter. Accepts property and dimension values of Numeric type (Integer, Floating point, DateTime), or Geocode type. For values of type Floating point, you can specify values using both decimal (0.00...68), and scientific notation (6.8e-10).
Object	`Navigation`
Dependency	`N`

Examples

```
/controller.php?N=0&Nf=Price|GT+15

/controller.php?N=0&Nf=Price|BTWN+9+13

/controller.php?N=0&Nf=Location|GCLT+42.365615,-71.075647+10
```

Nmpt (Merchandising Preview Time)

The `Nmpt` parameter sets a preview time for the application.

Parameter	`Nmpt`
Name	Merchandising Preview Time
Java setter method	`ENEQuery.setNavMerchPreviewTime()`
.NET setter property	`ENEQuery.NavMerchPreviewTime`
Type	<string> value of the form: `YYYY-MM-DDTHH:MM` The letter T is a separator between the day value and the hour value. Time zone information is omitted.
Description	Sets a preview time that overrides the clock of the MDEX Engine. Allows the user to preview the results of dynamic business rules that have time values associated with their triggers. This is a testing convenience for rules with time triggers.
Object	`Navigation`
Dependency	`N`

Example

```
/controller.php?N=0&Nmpt=2006-10-15T18:00&Ne=1000
```

Nmrf (Merchandising Rule Filter)

The `Nmrf` parameter sets a dynamic business rule filter for the navigation query.

Parameter	`Nmrf`
Name	Merchandising Rule Filter
Java setter method	`ENEQuery.setNavMerchRuleFilter()`
.NET setter property	`ENEQuery.NavMerchRuleFilter`
Type	This filter can include strings, integers, separator characters, Boolean operators, wildcard operators, and Endeca property values.
Description	This parameter can be used to specify a rule filter that restricts the results of a navigation query to only the records that can be promoted by rules that match the filter.
Object	`Navigation`
Dependency	`N`

Examples

```
/controller.php?N=0&Nmrf=or(state:pending,state:approved)

/controller.php?N=0&Nmrf=or(1,5,8)
```

No (Record Offset)

The `No` parameter sets the navigation record list offset.

Parameter	`No`
Name	Record Offset
Java setter method	`ENEQuery.setNavERecsOffset()`
.NET setter property	`ENEQuery.NavERecsOffset`
Type	int
Description	The offset defines the starting index for a navigation object's record list. If the No parameter is 20, the list of items returned in a navigation object's record list will begin with item 21. (Offset is a zero-based index.) This parameter allows users to page through a long result set, either directly or step by step. If an offset is greater than the number of items in a navigation object's record list, then the record list returned will be empty.

Object	`Navigation`
Dependency	`N`

Example

```
/controller.php?N=132831+154283&No=20
```

Np (Records per Aggregated Record)

The `Np` parameter sets the maximum number of records to be returned in each aggregated record.

Parameter	`Np`
Name	Records per Aggregated Record
Java setter method	`ENEQuery.setNavERecsPerAggrERec()`
.NET setter property	`ENEQuery.NavERecsPerAggrERec`
Type	0, 1, or 2
Description	Specifies the number of records to be returned with an aggregated record: • A value of 0 means that no records are returned with each aggregated record. • A value of 1 means that a single representative record is returned with each aggregate record. • A value of 2 means that all records are returned with each aggregated record. To improve performance, use 0 or 1.
Object	`Navigation`
Dependency	`N`, `Nu`

Example

```
/controller.php?N=0&Nu=ssn&Np=0
```

Nr (Record Filter)

The `Nr` parameter sets a record filter on a navigation query.

Parameter	`Nr`
Name	Record Filter
Java setter method	`ENEQuery.setNavRecordFilter()`
.NET setter property	`ENEQuery.NavRecordFilter`
Type	<string>

Description	This parameter can be used to specify a record filter expression that will restrict the results of a navigation query.
Object	`Navigation`
Dependency	`N`

Examples

```
/controller.php?N=0&Nr=FILTER(MyFilter)

/controller.php?N=0&Nr=OR(sku:123,OR(sku:456),OR(sku:789))
```

Nrc (Dynamic Refinement Ranking)

The `Nrc` parameter sets a dynamic refinement configuration for the navigation query.

Parameter	`Nrc`
Name	Dynamic Refinement Ranking
Java setter method	`ENEQuery.setNavRefinementConfigs()`
.NET setter property	`ENEQuery.NavRefinementConfigs`
Type	<string>+<string>+<string>...
Description	Sets one or more dynamic refinement configurations for the navigation query. Each dynamic refinement configuration is delimited by the pipe character and must have the `id` setting. The configuration settings are: • `id` indicates the dimension value ID • `exposed` either true if the dimension value's refinements are exposed or false if not • `showcounts` indicates whether to show counts for a dimension value's refinements. Valid values are `true` to indicate counts are shown and `false` to indicate counts are not shown. • `dynrank` whether the dimension value has Dynamic Ranking enabled: enabled, disabled, or default • `dyncount` maximum number of dimension values to return: either default or an integer >= 0 • `dynorder` sort order: static, dynamic, or default Omitting a setting or specifying `default` results in using the setting in Developer Studio.
Object	`Navigation`
Dependency	`N`

Example

```
/controller.php?N=0&Nrc=id+134711+exposed+true+dynrank+enabled+dyncount+de¬
fault+dynorder+dynamic+showcounts+true|id+132830+dyncount+7
```

Nrcs (Dimension Value Stratification)

The `Nrcs` parameter sets the list of stratified dimension values for use during refinement ranking by the MDEX Engine.

Parameter	`Nrcs`
Name	Dimension Value Stratification
Java setter method	`ENEQuery.setNavStratifiedDimVals()`
.NET setter property	`ENEQuery.NavStratifiedDimVals`
Type	int,int;int,int;...
Description	Sets the stratification configuration for a list of dimension values. The stratified dimension values are delimited by semi-colons (;) and each stratified dimension value is in the format: `stratumInt,dimvalID` where *dimvalID* is the ID of the dimension value and *stratumnt* is a signed integer that signifies that stratum into which the dimension value will be placed. For *stratumInt*, a positive integer will boost the dimension value while a negative integer will bury it. Dimension values that are not specified will be assigned the strata of 0.
Object	`Navigation`
Dependency	`N`

Example

```
/controller.php?N=0&Nrcs=2,4001;2,3429;1,4057;1,4806;1,4207;-1,5408;-1,4809
```

Nrk (Relevance Ranking Key)

The `Nrk` parameter sets the search interface to be used when using relevance ranking in a record search.

Parameter	`Nrk`
Name	Relevance Ranking Key
Java setter method	`ENEQuery.setNavRelRankERecRank()`
.NET setter property	`ENEQuery.NavRelRankERecRank`
Type	<search interface>

Description	Sets the search interface to be used when using relevance ranking in a record search. Note that the search interface is not required to have a relevance ranking strategy implemented. Dimension names or property names are not supported for this parameter, only search interfaces. In addition, this parameter does not support multiple search interfaces; therefore, the use of a pipe (|) is not allowed. Note that the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters take precedence over `Ntk`, `Ntt`, and `Ntx`.
Object	`Navigation`
Dependency	`N`, `Nrt`, `Nrr`

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sono¬
ma&Nrk=All&Nrt=pear&Nrr=field&Nrm=matchall
```

Nrm (Relevance Ranking Match Mode)

The `Nrm` parameter sets the relevance ranking match mode to be used to rank the results of the record search.

Parameter	`Nrm`
Name	Relevance Ranking Match Mode
Java setter method	`ENEQuery.setNavRelRankERecRank()`
.NET setter property	`ENEQuery.NavRelRankERecRank`
Type	<string>
Description	With the exception of MatchBoolean, all of the search modes are valid for use: MatchAll, MatchPartial, MatchAny, MatchAllAny, MatchAllPartial, and MatchPartialMax. Attempting to use MatchBoolean with this parameter will cause the record search results to be returned without relevance ranking. This parameter does not support multiple match modes; therefore, the use of a pipe (|) is not allowed. Note that the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters take precedence over `Ntk`, `Ntt`, and `Ntx`. This parameter is not supported for use with the Aggregated MDEX Engine (Agraph).
Object	`Navigation`
Dependency	`N`, `Nrk`, `Nrt`, `Nrr`

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sono¬
ma&Nrk=All&Nrt=pear&Nrr=field&Nrm=matchall
```

Nrr (Relevance Ranking Strategy)

The `Nrr` parameter sets the relevance ranking strategy to be used to rank the results of the record search.

Parameter	`Nrr`
Name	Relevance Ranking Strategy
Java setter method	`ENEQuery.setNavRelRankERecRank()`
.NET setter property	`ENEQuery.NavRelRankERecRank`
Type	<string>
Description	Sets the relevance ranking strategy to be used to rank the results of the record search. The valid id module names that can be used are: exact, field, first, freq, glom, interp, maxfield, nterms, numfields, phrase, proximity, spell, compound, stem, thesaurus, and static. This parameter does not support multiple relevance ranking strategies; therefore, the use of a pipe (|) is not allowed. Note that the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters take precedence over `Ntk`, `Ntt`, and `Ntx`. This parameter is not supported for use with the Aggregated MDEX Engine (Agraph).
Object	`Navigation`
Dependency	`N`, `Nrk`, `Nrt`

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sono¬
ma&Nrk=All&Nrt=pear&Nrr=field&Nrm=matchall
```

Nrs (Endeca Query Language Filter)

The `Nrs` parameter sets an EQL record filter on a navigation query.

Parameter	`Nrs`
Name	Endeca Query Language Filter
Java setter method	`ENEQuery.setNavRecordStructureExpr()`
.NET setter property	`ENEQuery.NavRecordStructureExpr`

Type	<string>
Description	Sets the Endeca Query Language expression for the navigation query. The expression will act as a filter to restrict the results of the query. The `Nrs` parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.
Object	`Navigation`
Dependency	`N`

Examples

```
/controller.php?N=0&Nrs=collection()/record[type="book"]
```

Nrt (Relevance Ranking Terms)

The `Nrt` parameter sets the terms by which the relevance ranking module will order the results of the record search.

Parameter	`Nrt`
Name	Relevance Ranking Terms
Java setter method	`ENEQuery.setNavRelRankERecRank()`
.NET setter property	`ENEQuery.NavRelRankERecRank`
Type	<string>+<string>+<string>...
Description	Sets the terms by which the relevance ranking module will order the records. Each term is delimited by a plus sign (+). Note that these terms can be different from the search terms used in the record search. This parameter does not support multiple sets of terms; therefore, the use of a pipe (|) is not allowed. The `Nrt` parameter must be used with the `Nrk` parameter (which sets the search interface) and the `Nrr` parameter (which indicates the relevance ranking strategy to use for ordering the record set). Note that the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters take precedence over `Ntk`, `Ntt`, and `Ntx`. This parameter is not supported for use with the Aggregated MDEX Engine (Agraph).
Object	`Navigation`
Dependency	`N`, `Nrk`, `Nrr`

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sono¬
ma&Nrk=All&Nrt=pear&Nrr=field&Nrm=matchall
```

Ns (Sort Key)

The Ns parameter sets the list of keys that will be used to sort records.

Parameter	Ns
Name	Sort Key
Java setter method	ENEQuery.setNavActiveSortKeys()
.NET setter property	ENEQuery.NavActiveSortKeys
Type	Ns=sort-key-names[(geocode)][|sort order][||…]
Description	Specifies a list of properties or dimensions (sort keys) by which to sort the records, and an optional list of directions in which to sort. In other words, in order to sort records returned for a navigation query, you must append a sort key parameter (Ns) to the query, using the following syntax: Ns=sort-key-names[(geocode)][|sort order][||…] A sort key is a dimension or property name enabled for sorting on the data set. Optionally, each sort key can specify a sort order of 0 (ascending sort, the default) or 1 (descending sort). The records are sorted by the first sort key, with ties being resolved by the second sort key, whose ties are resolved by the third sort key, and so on. Whether the values for the sort key are sorted alphabetically, numerically, or geospatially is specified in Developer Studio. To sort records by their geocode property, add the optional geocode argument to the sort key parameter (noting that the sort key parameter must be a geocode property). Records are sorted by the distance from the geocode reference point to the geocode point indicated by the property key. Sorting can only be performed when accompanying a navigation query. Therefore, the sort key (Ns) parameter must accompany a basic navigation value parameter (N).
Object	Navigation
Dependency	N

Examples

```
N=132831+154283&Ns=Price|1
```

```
N=0&Ns=Price
N=101&Ns=Price|1||Color
N=101&Ns=Price|1||Location(43,73)
```

Nso (Sort Order)

The Nso parameter sets the sort order for the record list of the navigation object.

Parameter	Nso
Name	Sort Order
Java setter method	ENEQuery.setNavSortOrder()
.NET setter property	ENEQuery.NavSortOrder
Type	0 or 1
Description	Specifies the sort order for a navigation object's record list: • A value of 0 indicates an ascending sort, which is the default if the Nso parameter is not present. • A value of 1 indicates a descending sort. Note that previously, a sort key was specified with the Ns=key parameter and a sort order was specified with Nso=1. The Nso parameter has been deprecated. Now, the preferred way of specifying the sort order is also through the Ns parameter, using Ns=key|1.
Object	Navigation
Dependency	N, Ns

Example

```
/controller.php?N=132831+154283&Ns=Price&Nso=1
```

Ntk (Record Search Key)

The Ntk parameter sets which dimension, property, or search interface will be evaluated when searching.

Parameter	Ntk
Name	Record Search Key
Java setter method	ENEQuery.setNavERecSearches()
.NET setter property	ENEQuery.NavERecSearches
Type	<search key>
Description	Sets the keys of the record search for the navigation query. The keys are delimited by a pipe (|). Search keys can be either valid dimension names or property names enabled for record search in the data set. The search key can also be a search interface. The Ntk parameter must be used with the Ntt parameter, which indicates the search terms for each key. In addition, Ntt should have the same number of term sets as Ntk has keys. Note that there is no explicit text search descriptor API object, so displays of text search descriptors need to be extracted from the current query.

Object	`Navigation`
Dependency	`N`, `Ntt`.

Examples

```
/controller.php?N=0&Ntk=DESCRIP&Ntt=merlot+1996

/controller.php?N=132831&Ntk=DESCRIP&Ntt=merlot+1996
```

Ntpc (Compute Phrasings)

The `Ntpc` parameter sets whether the MDEX Engine computes alternative phrasings for the current query.

Parameter	`Ntpc`
Name	Compute Phrasings
Java setter method	`ENEQuery.setNavERecSearchComputeAlternativePhrasings()`
.NET setter property	`ENEQuery.NavERecSearchComputeAlternativePhrasings`
Type	0 or 1
Description	Specifies whether to turn on the computed alternative phrasings feature for a record search (a value of 1) or to turn it off (a value of 0). 0 is the default.
Object	`Navigation`
Dependency	`N`, `Ntk`, `Ntt`. `Nty` is also a dependency if Did You Mean and automatic phrasing are being used.

Example

```
/controller.php?N=0&Ntk=All&Ntt=napa%20valley&Nty=1&Ntpc=1
```

Ntpr (Rewrite Query with an Alternative Phrasing)

The `Ntpc` parameter sets whether the MDEX Engine uses one of the alternative phrasings it has computed.

Parameter	`Ntpr`
Name	Rewrite Query with an Alternative Phrasing
Java setter method	`ENEQuery.setNavERecSearchRewriteQueryToAnAlternativePhrasing()`
.NET setter property	`ENEQuery.NavERecSearchRewriteQueryToAnAlternativePhrasing`
Type	0 or 1
Description	Sets whether the MDEX Engine uses one of the alternative phrasings it has computed instead of the end user's original query when computing the set of

	documents to return. 1 instructs the MDEX Engine to use a computed alternative phrasing, while 0 (the default) instructs it to use the user's original query.
Object	`Navigation`
Dependency	`N`, `Ntk`, `Ntt`, `Ntpc`. `Nty` is also a dependency if Did You Mean and automatic phrasing are being used.

Example

```
/controller.php?N=0&Ntk=All&Ntt=napa%20valley&Nty=1&Ntpc=1&Ntpr=1
```

Ntt (Record Search Terms)

The `Ntt` parameter sets the actual terms of a record search for a navigation query.

Parameter	`Ntt`
Name	Record Search Terms
Java setter method	`ENEQuery.setNavERecSearches()`
.NET setter property	`ENEQuery.NavERecSearches`
Type	<string>+<string>+<string>...
Description	Sets the terms of the record search for a navigation query. Each term is delimited by a plus sign (+). Each set of terms is delimited by a pipe (|). The `Ntt` parameter must be used with the `Ntk` parameter, which indicates which keys of the records to search. In addition, `Ntt` should have the same number of term sets as `Ntk` has keys. Note that there is no explicit text search descriptor API object, so displays of text search descriptors need to be extracted from the current query.
Object	`Navigation`
Dependency	`N`, `Ntk`.

Examples

```
/controller.php?N=0&Ntk=DESCRIP&Ntt=merlot+1996

/controller.php?N=132831&Ntk=DESCRIP&Ntt=merlot+1996
```

Ntx (Record Search Mode)

The `Ntx` parameter sets the options for record search in the navigation query.

Parameter	`Ntx`
Name	Record Search Mode

Java setter method	`ENEQuery.setNavERecSearches()`
.NET setter property	`ENEQuery.NavERecSearches`
Type	<string>+<string>+<string>...
Description	Sets the options for record search in the navigation query. The options include: • `mode` for specifying a search mode. • `rel` for specifying a relevance ranking module. • `spell+nospell` for disabling spelling correction and DYM suggestions on individual queries. • `snip` and `nosnip` operators for enabling or disabling the snippeting feature, specifying a field to snippet, and configuring how many words to return in a snippet.
Object	`Navigation`
Dependency	`N`, `Ntk`, `Ntt`

Examples

```
/controller.php?N=0&Ntk=Brand&Ntt=Nike+Adidas&Ntx=mode+matchallany+rel+MyS¬
trategy
```

```
/controller.php?N=0&Ntk=Brand&Ntt=Nike+Adidas&Ntx=mode+spell+nospell
```

Nty (Did You Mean)

The `Nty` parameter sets the Did You Mean feature for record search in the navigation query.

Parameter	`Nty`
Name	Did You Mean
Java setter method	`ENEQuery.setNavERecSearchDidYouMean()`
.NET setter property	`ENEQuery.NavERecSearchDidYouMean`
Type	0 or 1
Description	Sets whether the record search should turn on the "Did You Mean" feature. This parameter is only used if a full-text query is being made with the navigation. The default value is 0 (off).
Object	`Navigation`
Dependency	`N`, `Ntk`, `Ntt`

Example

```
/controller.php?N=0&Ntk=DESC&Ntt=merlot+1996&Nty=1
```

Nu (Rollup Key)

The Nu parameter sets the rollup key for aggregated records.

Parameter	Nu
Name	Rollup Key
Java setter method	ENEQuery.setNavRollupKey()
.NET setter property	ENEQuery.NavRollupKey
Type	<dimension or property key>
Description	Specifies the dimension or property by which records in a navigation object's record list should be aggregated. By setting a key with this parameter, aggregated Endeca records (AggERec objects) will be returned by the navigation query instead of Endeca records (ERec objects). Note that the rollup attribute of the property or dimension must be set in Developer Studio.
Object	Navigation
Dependency	N

Examples

```
/controller.php?N=0&Nu=ssn

/controller.php?N=13283&Nu=ssn
```

Nx (Navigation Search Options)

The Nx parameter sets the options that navigation search uses (excluding options such as record search).

Parameter	Nx
Name	Navigation Search Options
Java setter method	ENEQuery.setNavOpts()
.NET setter property	ENEQuery.NavOpts
Type	<string>+<string>+<string>...
Description	Sets the navigation search options used to enable Why Match, Why Rank, and Why Precedence Rule Fired. Valid string values include: • whymatch — a string indicating that Why Match is enabled for the query. • whyrank — a string indicating that Why Rank is enabled for the query. • whyprecedencerulefired — a string indicating that Why Precedence Rule Fired is enabled for the query.
Object	Navigation Search

Dependency	N

Examples

This simple example enables Why Did It Match:

```
/controller.php?N=0&Nx=whymatch
```

This simple example enables Why Rank:

```
/controller.php?N=0&Nx=whyrank
```

This simple example enables Why Precedence Rule Fired:

```
/controller.php?N=500&Nx=whyprecedencerulefired
```

R (Record)

The R parameter sets the ID of the record to be queried for.

Parameter	R
Name	Record
Java setter method	ENEQuery.setERecs()
.NET setter property	ENEQuery.ERecs
Type	<record ID>
Description	Query to obtain a single specific Endeca record.
Object	Record (ERec)
Dependency	none

Example

```
/controller.php?R=7
```

A (Aggregated Record)

The A parameter sets the ID of an aggregated record to be queried for.

Parameter	A
Name	Aggregated Record
Java setter method	ENEQuery.setAggrERecSpec()
.NET setter property	ENEQuery.AggrERecSpec
Type	<agg record ID>
Description	Query to obtain a single aggregated record from the MDEX Engine.
Object	Aggregated Record (AggrERec)

Dependency	An, Au (Note that A, An, and Au are all considered primary parameters and must be used together.)

Example

```
/controller.php?A=7&An=123&Au=ssn
```

Af (Aggregated Record Range Filter)

The Af parameter sets the aggregated record range filters for the navigation query..

Parameter	Af
Name	Aggregated Record Range Filter
Java setter method	ENEQuery.setAggERecNavRangeFilters()
.NET setter property	ENEQuery.AggERecNavRangeFilters
Type	<string>|[[LT|LTEQ|GT|GTEQ] <numeric value> |BTWN <numeric value> <numeric value>] <key>|[GCLT|GCGT|GCBTWN][+<geocode reference point>]+<value>[+<value>]
Description	Sets the aggregated record navigation range filters. Multiple filters are delimited by vertical pipes (|).
Object	Aggregated Record (AggrERec)
Dependency	A, An, Au

Example

```
/controller.php?A=7&An=123&Au=ssn&Af=Base|GT+100000
```

An (Aggregated Record Descriptors)

The An parameter sets the navigation values which the aggregated record will be aggregated in relation to.

Parameter	An
Name	Aggregated Record Descriptors
Java setter method	ENEQuery.setAggrERecNavDescriptors()
.NET setter property	ENEQuery.AggrERecNavDescriptors
Type	<dimension value id>+<dimension value id>+<dimension value id>...
Description	Sets the aggregated record navigation values for the query. An and Au define the record set from which the aggregated record was created.

Object	Aggregated Record (AggrERec)
Dependency	A, Au (Note that A, An, and Au are all considered primary parameters and must be used together.)

Example

```
/controller.php?A=7&An=123&Au=ssn
```

Ar (Aggregated Record Filter)

The An parameter sets the aggregated record navigation record filter.

Parameter	Ar
Name	Aggregated Record Filter
Java setter method	ENEQuery.setAggERecNavRecordFilter()
.NET setter property	ENEQuery.AggERecNavRecordFilter
Type	<string>
Description	Sets the aggregated record navigation record filter. This filter expression restricts the records contained in an aggregated record result returned by the MDEX Engine.
Object	Aggregated Record (AggrERec)
Dependency	A, An

Example

```
/controller.php?A=2496&An=0&Au=sku&Ar=OR(10001,20099)
```

Ars (Aggregated EQL Filter)

The Ars parameter sets an aggregated record EQL filter.

Parameter	Ars
Name	Aggregated EQL Filter
Java setter method	ENEQuery.setAggrERecStructureExpr()
.NET setter property	ENEQuery.AggrERecStructureExpr
Type	<string>
Description	Sets the Endeca Query Language expression for aggregated record query. The expression will act as a filter to restrict the results of the query. The Ars parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.

Object	Aggregated Record (`AggrERec`)
Dependency	A

Example

```
/controller.php?An=0&A=1&Au=author_nationality
&Ars=collection()/record[recordtype = "author" and not(author_name="kurt
vonnegut")]
```

As (Aggregated Record Sort Key)

The `As` parameter sets the list of keys that will be used to sort representative records in an aggregated record details query.

Parameter	`As`
Name	Aggregated Record Sort Key
Java setter method	`ENEQuery.setAggrERecActiveSortKeys()`
.NET setter property	`ENEQuery.AggrERecActiveSortKeys`
Type	`As=sort-key-names[(geocode)][|sort order][||…]`
Description	Specifies a list of properties or dimensions (sort keys) by which to sort the representative records, and an optional list of directions in which to sort. In other words, in order to sort representative records in aggregated records, you must append a sort key parameter (`As`) to the aggregated record query, using the following syntax: `As=sort-key-names[(geocode)][|sort order][||…]` A sort key is a dimension or property name enabled for sorting on the data set. Optionally, each sort key can specify a sort order of 0 (ascending sort, the default) or 1 (descending sort). The records are sorted by the first sort key, with ties being resolved by the second sort key, whose ties are resolved by the third sort key, and so on. Whether the values for the sort key are sorted alphabetically, numerically, or geospatially is specified in Developer Studio. To sort records by their geocode property, add the optional `geocode` argument to the sort key parameter (noting that the sort key parameter must be a geocode property). Records are sorted by the distance from the geocode reference point to the geocode point indicated by the property key.
Object	Aggregated Record (`AggrERec`)
Dependency	A, An

Example

```
/controller.php?A=7&An=123&Au=ssn&As=Price|1
```

Au (Aggregated Record Rollup Key)

The Au parameter sets the rollup key for aggregated records.

Parameter	Au
Name	Aggregated Record Rollup Key
Java setter method	ENEQuery.setAggrERecRollupKey()
.NET setter property	ENEQuery.AggrERecRollupKey
Type	<dimension or property key>
Description	Sets the aggregated record rollup key (a property or dimension) with which the aggregated record is derived. Note that the rollup attribute of the property or dimension must be set in Developer Studio.
Object	Aggregated Record (AggrERec)
Dependency	A, An

Example

```
/controller.php?A=7&An=123&Au=ssn
```

D (Dimension Search)

The D parameter sets the dimension search query terms.

Parameter	D
Name	Dimension Search
Java setter method	ENEQuery.setDimSearchTerms()
.NET setter property	ENEQuery.DimSearchTerms
Type	<string>+<string>+<string>...
Description	Query to obtain the set of dimension values whose names match the search term(s).
Object	DimensionSearchResult
Dependency	none

Examples

```
/controller.php?D=Merlot

/controller.php?D=Red+White
```

Df (Dimension Search Range Filter)

The `Df` parameter sets the navigation range filters that restrict the dimension search.

Parameter	`Df`
Name	Dimension Search Range Filter
Java setter method	`ENEQuery.setDimSearchNavRangeFilters()`
.NET setter property	`ENEQuery.DimSearchNavRangeFilters`
Type	<string>|[[LT|LTEQ|GT|GTEQ] <number> |BTWN <number> <number>] <key>|[GCLT|GCGT|GCBTWN][+<geocode reference point>]+<value>[+<value>]
Description	Sets the dimension search to be applied to dimension values for those records that passed the range filter used for this property. Multiple filters are vertical pipe (|) delimited.
Object	Dimension Value Search
Dependency	`D`

Example

```
/controller.php?D=Merlot&Df=Price|LT+11
```

Di (Search Dimension)

The `Di` parameter sets the dimensions for a dimension search to search against.

Parameter	`Di`
Name	Search Dimension
Java setter method	`ENEQuery.setDimSearchDimensions()`
.NET setter property	`ENEQuery.DimSearchDimensions`
Type	<dimension id> or <dimension id>+<dimension id>...
Description	The Di parameter can be used with two types of dimension search: • Default dimension search • Compound dimension search Note that by default, all dimensions are enabled for default dimension search. If you use Dgidx `--compoundDimSearch` flag, all dimensions are enabled for compound dimension search. If used for default dimension search, specify one or more dimension IDs for the `Di` parameter. The MDEX Engine returns matches only from the dimensions you specify (as opposed to the default behavior of searching across all dimensions).

	If used for the compound dimension search, specify a list of dimension IDs for the `Di` parameter. This way, you are requiring that every result returned has exactly one value from each dimension ID specified in `Di`. This restricts your compound dimension search to the intersection of the specified dimensions (as opposed to the compound dimension search across all dimensions).
Object	Dimension Value Search
Dependency	`D`

Examples

```
/controller.php?D=Merlot&Di=11378

/controller.php?D=red+1996&Di=11+12
```

Dk (Dimension Search Rank)

The `Dk` parameter sets how the dimension search results are sorted.

Parameter	`Dk`
Name	Dimension Search Rank
Java setter method	`ENEQuery.setDimSearchRankResults()`
.NET setter property	`ENEQuery.DimSearchRankResults`
Type	0 or 1
Description	Sets the dimension search behavior used to rank results: • If set to 0, default dimension value ranking (alpha, numeric or manual as set in Developer Studio) is used to order dimension search results. This is the default. • If set to 1, relevance ranking is used to sort dimension search results.
Object	Dimension Value Search
Dependency	`D`

Example

```
/controller.php?D=Merlot&Dk=1
```

Dn (Dimension Search Scope)

The `Dn` parameter sets a navigation state that reduces the scope of a dimension value search.

Parameter	`Dn`
Name	Dimension Search Scope

Java setter method	`ENEQuery.setDimSearchNavDescriptors()`
.NET setter property	`ENEQuery.DimSearchNavDescriptors`
Type	<dimension value id>+<dimension value id>+<dimension value id>...
Description	Specifies the navigation values that describe a navigation state that restrict the number of values that can be searched from. The `Dn` parameter takes a single dimension value for a given single-select dimension, and multiple dimension values for a given multiselect dimension. When the search query is combined with this parameter, the MDEX Engine returns dimension values that create valid navigation objects.
Object	Dimension Value Search
Dependency	`D`

Example

```
/controller.php?D=Merlot&Dn=132831
```

Do (Search Result Offset)

The `Do` parameter sets the dimension search results offset.

Parameter	`Do`
Name	Dimension Search Offset
Java setter method	`ENEQuery.setDimSearchResultsOffset()`
.NET setter property	`ENEQuery.DimSearchResultsOffset`
Type	int
Description	Specifies the offset with which the dimension search will begin returning results per dimension. For example, you could specify an offset of 5 to look at a single dimension five results at a time.
Object	Dimension Value Search
Dependency	`D`, `Di`, `Dp`

Example

```
/controller.php?D=Merlot&Di=11378&Dp=3&Do=3
```

Dp (Dimension Value Count)

The `Dp` parameter sets the number of dimension value matches to return per dimension.

Parameter	`Dp`

Name	Dimension Value Count
Java setter method	`ENEQuery.setDimSearchNumDimValues()`
.NET setter property	`ENEQuery.DimSearchNumDimValues`
Type	int
Description	Sets the number of dimension value matches to return per dimension. If you do a dimension search, you normally get all of the results back. If you only want to see the first three, for example, specify 3 for the `Dp` parameter.
Object	Dimension Value Search
Dependency	`D`, `Di`

Example

```
/controller.php?D=Merlot&Di=11378&Dp=3
```

Dr (Dimension Search Filter)

The `Dr` parameter sets the record filter for the dimension search navigation query.

Parameter	`Dr`
Name	Dimension Search Filter
Java setter method	`ENEQuery.setDimSearchNavRecordFilter()`
.NET setter property	`ENEQuery.DimSearchNavRecordFilter`
Type	<string>
Description	Sets the dimension search navigation record filter. This filter restricts the scope of the records that will be considered for a dimension search. Only dimension values represented on at least one record satisfying the specified filter are returned as search results.
Object	Dimension Value Search
Dependency	`D`

Example

```
/controller.php?D=Hawaii&Dn=0&Dr=NOT(Subject:Travel)
```

Drc (Refinement Configuration for Dimension Search)

The `Drc` parameter sets refinement configuration options for a dimension search query.

Parameter	`Drc`
Name	Refinement Configuration for Dimension Search

Java setter method	`ENEQuery.setDimSearchRefinementConfigs()`
.NET setter property	`ENEQuery.DimSearchRefinementConfigs`
Type	<string>+<string>+<string>...
Description	Sets one or more dynamic refinement configurations for a dimension search query. Each refinement configuration option is delimited by the pipe character and must have the `id` setting. The configuration settings are: • `id` indicates the dimension value ID • `showcounts` indicates whether to show counts for a dimension value's refinements. Valid values are `true` to indicate counts are shown and `false` to indicate counts are not shown.
Object	`DimensionSearchResult`
Dependency	`D`

Example

```
/controller.php?D=1*&Drc=id+134711+showcounts+true|id+132830+show¬
counts+false
```

Drs (Dimension Search EQL Filter)

The `Drs` parameter sets the dimension search EQL filter.

Parameter	`Drs`
Name	Dimension Search EQL Filter
Java setter method	`ENEQuery.setDimSearchNavRecordStructureExpr()`
.NET setter property	`ENEQuery.DimSearchNavRecordStructureExpr`
Type	<string>
Description	Sets the Endeca Query Language filter for a dimension search. This filter restricts the scope of the records that will be considered for a dimension search. Only dimension values represented on at least one record satisfying the specified filter are returned as search results. Note that the `Drs` parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.
Object	Dimension Value Search
Dependency	`D`

Example

```
/controller.php?D=classic&Drs=collection()/record
```

Dx (Dimension Search Options)

The Dx parameter sets the options for dimension search.

Parameter	Dx
Name	Dimension Search Options
Java setter method	ENEQuery.setDimSearchOpts()
.NET setter property	ENEQuery.DimSearchOpts
Type	<string>+<string>+<string>...
Description	Sets the dimension search options used in search mode and relevance ranking. The options include: • mode for specifying a search mode. • rel for specifying a relevance ranking module. • spell+nospell for disabling spelling correction and DYM suggestions on individual queries. • whyrank to indicate that Why Rank is enabled for the query.
Object	Dimension Value Search
Dependency	D, Dk

Examples

```
/controller.php?D=mark+twain&Dk=1&Dx=rel+exact,static(rank,descending)
```

This example shows how to disable spelling correction for a dimension search query for "blue suede shoes":

```
/controller.php?D=blue+suede+shoes&Dx=mode+matchallpartial+spell+nospell
```

Du (Rollup Key for Dimension Search)

The Du parameter sets the property or dimension to use as the rollup key for aggregated records in a dimension search query.

Parameter	Du
Name	Rollup Key for Dimension Search
Java setter method	ENEQuery.setDimSearchRollupKey()
.NET setter property	ENEQuery.DimSearchRollupKey
Type	<dimension or property key>
Description	Specifies the dimension or property by which records are rolled up. This parameter has no meaning unless counts are enabled with Drc. Note that the rollup attribute of the property or dimension must be set in Developer Studio.

Object	`DimensionSearchResult`
Dependency	`D` and `Drc`

Examples

```
/controller.php?D=Merlot&Drc=id+1000+showcounts+true&Du=P_Winery
```

Index

E

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

W