

Oracle Commerce Guided Search

MDEX Engine Developer's Guide

Version 6.5.1 • July 2014



Contents

Copyright and disclaimer.....	11
Preface.....	13
About this guide.....	13
Who should use this guide.....	13
Conventions used in this guide.....	13
Contacting Oracle Support.....	13
 Part I: Overview of Oracle Commerce Guided Search Applications.....	 15
Chapter 1: About the MDEX Engine.....	17
MDEX Engine overview.....	17
About the Information Transformation Layer.....	18
Chapter 2: Assembler functionality.....	19
How the MDEX Engine Communicates with the Assembler.....	19
 Part II: Presentation API Basics.....	 21
Chapter 3: Presentation API Overview.....	23
List of Guided Search APIs.....	23
Architecture of the Presentation API.....	23
One query, one page.....	25
About query result objects returned by the MDEX Engine.....	26
Chapter 4: Working with the Presentation API.....	31
Core classes of the Presentation API.....	31
Using the core objects to query the MDEX Engine.....	34
Four basic queries.....	35
Getting started with your own Web application.....	40
List of query exceptions.....	40
Chapter 5: Using the Reference Implementation.....	43
Reference implementation overview.....	43
The purpose of the reference implementation.....	43
Four primary modules.....	44
Non-MDEX Engine URL parameters.....	45
About JavaScript files.....	46
Module maps.....	46
Module descriptions.....	51
Tips on using the UI reference implementation modules.....	53
Chapter 6: Running the Reference Implementations.....	55
Running the JSP reference implementation.....	55
Running the ASP.NET reference implementation.....	58
 Part III: Record Features.....	 61
Chapter 7: Working with Oracle Commerce Records.....	63
Displaying Oracle Commerce records.....	63
Displaying record properties.....	65
Displaying dimension values for Oracle Commerce records.....	68

Paging through a record set.....	70
Chapter 8: Sorting Oracle Commerce Records.....	73
About record sorting.....	73
Configuring precomputed sort.....	73
Changing the sort order with Dgidx flags.....	75
URL parameters for sorting.....	75
Sort API methods.....	76
Troubleshooting application sort problems.....	77
Performance impact for sorting.....	77
Using geospatial sorting.....	78
Chapter 9: Using Range Filters.....	83
About range filters.....	83
Configuring properties and dimensions for range filtering.....	83
URL parameters for range filters.....	84
Using multiple range filters.....	86
Examples of range filter parameters.....	86
Rendering the range filter results.....	87
Troubleshooting range filter problems.....	87
Performance impact for range filters.....	88
Chapter 10: Creating Aggregated Records.....	89
About aggregated records.....	89
Enabling record aggregation.....	89
Generating and displaying aggregated records.....	90
Aggregated record behavior.....	97
Refinement ranking of aggregated records.....	97
Chapter 11: Controlling Record Values with the Select Feature.....	99
About the Select feature.....	99
Configuring the Select feature.....	99
URL query parameters for Select.....	100
Selecting keys in the application.....	100
Chapter 12: Using the Oracle Commerce Query Language.....	103
About the Oracle Commerce Query Language.....	103
Oracle Commerce Query Language syntax.....	104
Making Oracle Commerce Query Language requests.....	107
Record Relationship Navigation queries.....	108
Dimension value queries.....	112
Record search queries.....	114
Range filter queries.....	117
Dimension search queries.....	119
Oracle Commerce Query Language interaction with other features.....	120
Oracle Commerce Query Language per-query statistics log.....	124
Creating an Oracle Commerce Query Language pipeline.....	127
Chapter 13: Record Filters.....	131
About record filters.....	131
Record filter syntax.....	131
Enabling properties for use in record filters.....	134
Data configuration for file-based filters.....	134
Record filter result caching.....	135
URL query parameters for record filters.....	135
Record filter performance impact.....	136
Chapter 14: Bulk Export of Records.....	139
About the bulk export feature.....	139
Configuring the bulk export feature.....	139

Using URL query parameters for bulk export.....	139
Setting the number of bulk records to return.....	140
Retrieving the bulk-format records.....	141
Performance impact for bulk export records.....	142
Part IV: Dimension and Property Features.....	145
Chapter 15: Property Types.....	147
Formats used for property types.....	147
Temporal properties.....	148
Chapter 16: Working with Dimensions.....	151
Displaying dimension groups.....	151
Displaying refinements.....	154
Displaying disabled refinements.....	162
Implementing dynamic refinement ranking.....	167
Displaying descriptors.....	173
Displaying refinement statistics.....	178
Displaying multiselect dimensions.....	182
Using hidden dimensions.....	186
Using inert dimension values.....	187
Displaying dimension value properties.....	189
Working with external dimensions.....	192
Chapter 17: Dimension Value Boost and Bury.....	193
About the dimension value boost and bury feature.....	193
Nracs parameter.....	193
Stratification API methods.....	194
Retrieving the DGraph.Strata property.....	195
Interaction with disabled refinements.....	196
Chapter 18: Using Derived Properties.....	199
About derived properties.....	199
Configuring derived properties.....	199
Displaying derived properties.....	200
Chapter 19: Configuring Key Properties.....	203
About key properties.....	203
Defining key properties.....	204
Automatic key properties.....	205
Key property API.....	205
Part V: Basic Search Features.....	207
Chapter 20: Record Searches.....	209
Keyword search overview.....	209
Making properties or dimensions searchable.....	210
Hierarchical record searches.....	210
Adding search synonyms to dimension values.....	211
Features for controlling record search.....	211
Search query processing order.....	214
Tips for troubleshooting record search.....	217
Performance impact of record search.....	218
Chapter 21: Search Interfaces.....	219
About search interfaces.....	219
About implementing search interfaces.....	219
Options for enabling cross-field matches.....	220

Additional search interfaces options.....	221
Search interfaces and URL query parameters (Ntk).....	221
Java examples of search interface methods.....	222
.NET examples of search interface properties.....	222
Tips for troubleshooting search interfaces.....	222
Chapter 22: Dimension Searches.....	223
About dimension search.....	223
Default dimension search.....	223
Compound dimension search.....	224
Enabling dimensions for dimension search.....	224
Ordering of dimension search results.....	225
Advanced dimension search parameters.....	227
Dgidx flags for dimension search.....	228
URL query parameters and dimension search.....	228
Methods for accessing dimension search results.....	233
Displaying refinement counts for dimension search.....	235
When to use dimension and record search.....	238
Performance impact of dimension search.....	239
Chapter 23: Record and Dimension Search Reports.....	241
Implementing search reports.....	241
Methods for search reports.....	241
Troubleshooting search reports.....	244
Chapter 24: Using Search Modes.....	247
List of search modes.....	247
Configuring search modes.....	249
URL query parameters for search modes.....	250
Search mode methods.....	251
Chapter 25: Using Boolean Search.....	253
About Boolean search.....	253
Example of Boolean query syntax.....	254
Examples of using the key restrict operator.....	255
About proximity search.....	255
Proximity operators and nested subexpressions.....	256
Boolean query semantics.....	257
Operator precedence.....	258
Interaction of Boolean search with other features.....	258
Error messages for Boolean search.....	259
Implementing Boolean search.....	260
URL query parameters for Boolean search.....	260
Methods for Boolean search.....	261
Troubleshooting Boolean search.....	262
Performance impact of Boolean search.....	262
Chapter 26: Using Phrase Search.....	263
About phrase search.....	263
About positional indexing.....	264
How punctuation is handled in phrase search.....	264
URL query parameters for phrase search.....	264
Performance impact of phrase search.....	265
Chapter 27: Using Snippeting in Record Searches.....	267
About snippeting.....	267
Snippet formatting and size.....	268
Snippet property names.....	269
About enabling and configuring snippeting.....	269
URL query parameters for snippeting.....	269
Reformatting a snippet for display in your Web application.....	270

Performance impact of snippeting.....	270
Tips and troubleshooting for snippeting.....	270
Chapter 28: Using Wildcard Search.....	271
About wildcard search.....	271
Interaction of wildcard search with other features.....	271
Ways to configure wildcard search.....	272
MDEX Engine flags for wildcard search.....	274
Presentation API development for wildcard search.....	275
Performance impact of wildcard search.....	275
Chapter 29: Search Characters.....	277
Using search characters.....	277
Query matching semantics.....	277
Categories of characters in indexed text.....	277
Search query processing.....	278
Implementing search characters.....	279
Dgidx flags for search characters.....	279
Presentation API development for search characters.....	279
MDEX Engine flags for search characters.....	280
Chapter 30: Examples of Query Matching Interaction.....	281
Record search without search characters enabled.....	281
Record search with search characters enabled.....	282
Record search with wildcard search enabled but without search characters.....	283
Record search with both wildcard search and search characters enabled.....	283
Chapter 31: Spelling Correction and Did You Mean.....	285
About Spelling Correction and Did You Mean.....	285
Spelling modes.....	286
Disabling spelling correction on individual queries.....	286
Spelling dictionaries created by Dgidx.....	287
Configuring spelling in Developer Studio.....	288
Modifying the dictionary file	289
About the admin?op=updateaspell operation.....	289
Enabling language-specific spelling correction.....	290
Dgidx flags for Spelling Correction.....	290
dgraph flags for enabling Spelling Correction and DYM.....	290
URL query parameters for Spelling Correction and DYM.....	291
Spelling Correction and DYM API methods.....	292
dgraph tuning flags for Spelling Correction and Did You Mean.....	295
How dimension search treats number of results.....	298
Troubleshooting Spelling Correction and Did You Mean.....	298
Performance impact for Spelling Correction and Did You Mean.....	299
Compiling the Aspell dictionary.....	300
About word-break analysis.....	302
Chapter 32: Stemming and Thesaurus.....	303
Overview of Stemming and Thesaurus.....	303
About the Stemming feature.....	303
About the Thesaurus feature.....	309
Dgidx and dgraph flags for the Thesaurus.....	311
Interactions with other search features.....	312
Performance impact of Stemming and Thesaurus.....	313
Chapter 33: Automatic Phrasing.....	315
About Automatic Phrasing.....	315
Using Automatic Phrasing with Spelling Correction and DYM.....	316
Adding phrases to a project.....	316
Presentation API development for Automatic Phrasing.....	319
Tips and troubleshooting for Automatic Phrasing.....	323

Chapter 34: Stop Words.....	325
About stop words.....	325
Adding a sample list of stop words to an application.....	326
Chapter 35: Relevance Ranking.....	327
About the Relevance Ranking feature.....	327
Relevance Ranking modules.....	327
Relevance Ranking strategies.....	337
Implementing relevance ranking.....	338
Controlling relevance ranking at the query level.....	341
Relevance Ranking sample scenarios.....	345
Recommended strategies.....	347
Performance impact of Relevance Ranking.....	349
Chapter 36: Record Boost and Bury.....	351
About the record boost and bury feature.....	351
Enabling properties for filtering.....	351
The stratify relevance ranking module.....	352
Record boost/bury queries.....	353
Boost/bury sorting for Oracle Commerce records.....	354
Part VI: Content Spotlighting and Merchandizing.....	357
Chapter 38: Implementing User Profiles.....	359
About user profiles.....	359
Profile-based trigger scenario.....	359
User profile query parameters.....	360
API objects and method calls.....	360
Performance impact of user profiles.....	361
Part VII: Understanding and Debugging Query Results.....	363
Chapter 39: Using Why Match.....	365
About the Why Match feature.....	365
Enabling Why Match.....	365
Why Match API.....	365
Why Match property format.....	366
Why Match performance impact.....	366
Chapter 40: Using Word Interpretation.....	369
About the Word Interpretation feature.....	369
Implementing Word Interpretation.....	369
Word Interpretation API methods.....	369
Troubleshooting Word Interpretation.....	371
Chapter 41: Using Why Rank.....	373
About the Why Rank feature.....	373
Enabling Why Rank.....	373
Why Rank API.....	373
Why Rank property format.....	374
Result information for relevance ranking modules.....	375
Why Rank performance impact.....	376
Chapter 42: Using Why Precedence Rule Fired.....	377
About the Why Precedence Rule Fired feature.....	377
Enabling Why Precedence Rule Fired.....	377
Why Precedence Rule Fired API.....	377

Why Precedence Rule Fired property format.....	378
Performance impact of Why Precedence Rule Fired.....	380
Appendix A: Oracle Commerce URL Parameter Reference.....	381
About the Oracle Commerce URL query syntax.....	381
N (Navigation).....	382
Nao (Aggregated Record Offset).....	382
Ndr (Disabled Refinements).....	383
Ne (Exposed Refinements).....	384
Nf (Range Filter).....	384
Nmpt (Merchandising Preview Time).....	385
Nmrf (Merchandising Rule Filter).....	386
No (Record Offset).....	386
Np (Records per Aggregated Record).....	387
Nr (Record Filter).....	387
Nrc (Dynamic Refinement Ranking).....	388
Nrcs (Dimension Value Stratification).....	389
Nrk (Relevance Ranking Key).....	389
Nrm (Relevance Ranking Match Mode).....	390
Nrr (Relevance Ranking Strategy).....	391
Nrs (Oracle Commerce Query Language Filter).....	391
Nrt (Relevance Ranking Terms).....	392
Ns (Sort Key).....	392
Nso (Sort Order).....	393
Ntk (Record Search Key).....	394
Ntpc (Compute Phrasings).....	395
Ntpr (Rewrite Query with an Alternative Phrasing).....	395
Ntt (Record Search Terms).....	396
Ntx (Record Search Mode).....	396
Nty (Did You Mean).....	397
Nu (Rollup Key).....	397
Nx (Navigation Search Options).....	398
R (Record).....	398
A (Aggregated Record).....	399
Af (Aggregated Record Range Filter).....	399
An (Aggregated Record Descriptors).....	400
Ar (Aggregated Record Filter).....	400
Ars (Aggregated EQL Filter).....	401
As (Aggregated Record Sort Key).....	401
Au (Aggregated Record Rollup Key).....	402
D (Dimension Search).....	403
Df (Dimension Search Range Filter).....	403
Di (Search Dimension).....	404
Dk (Dimension Search Rank).....	404
Dn (Dimension Search Scope).....	405
Do (Search Result Offset).....	405
Dp (Dimension Value Count).....	406
Dr (Dimension Search Filter).....	406
Drc (Refinement Configuration for Dimension Search).....	407
Drs (Dimension Search EQL Filter).....	408
Dx (Dimension Search Options).....	409
Du (Rollup Key for Dimension Search).....	410
Appendix B: MDEX Engine Logging Variables.....	411
About MDEX Engine logging variables.....	411
Logging variable operation syntax.....	411
Supported logging variables.....	412
Appendix C: Diacritical Character to ASCII Character Mapping.....	413
Mapping table.....	413

Copyright and disclaimer

Copyright © 2003, 2014, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Preface

Oracle Commerce Guided Search is the most effective way for your customers to dynamically explore your storefront and find relevant and desired items quickly. An industry-leading faceted search and Guided Navigation solution, Guided Search enables businesses to influence customers in each step of their search experience. At the core of Guided Search is the MDEX Engine™, a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Oracle Commerce Content Acquisition System provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. The Oracle Commerce Assembler dynamically assembles content from any resource and seamlessly combines it into results that can be rendered for display.

Oracle Commerce Experience Manager enables non-technical users to create, manage, and deliver targeted, relevant content to customers. With Experience Manager, you can combine unlimited variations of virtual product and customer data into personalized assortments of relevant products, promotions, and other content and display it to buyers in response to any search or facet refinement. Out-of-the-box templates and experience cartridges are provided for the most common use cases; technical teams can also use a software developer's kit to create custom cartridges.

About this guide

This guide describes how to create an Oracle Commerce Guided Search application.

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

Who should use this guide

This guide is intended for developers who are writing applications using Oracle Commerce Guided Search.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Support

Oracle Support provides registered users with answers to implementation questions, product and solution help, and important news and updates about Guided Search software.

You can contact Oracle Support through the My Oracle Support site at <https://support.oracle.com>.

Part 1

Overview of Oracle Commerce Guided Search Applications

- [*About the MDEX Engine*](#)
- [*Assembler functionality*](#)

Chapter 1

About the MDEX Engine

This section provides an overview of the functions that the MDEX Engine performs in an Oracle Commerce Guided Search application.

MDEX Engine overview

The MDEX Engine is the indexing and query module used by Guided Search applications to retrieve information that customers request.

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

The MDEX Engine is 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.

Adding replicas of MDEX Engines on additional servers provides redundancy and improves query response time. That is, if one server goes down, a replica of an MDEX Engine provides redundancy by enabling other servers in the implementation to continue to reply to queries. In addition, load balancers can improve total response time by distributing 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 Guide Search application includes one or more dgraphs.
dgidx	dgidx is the indexing program that reads the tagged records that were prepared by Forge and creates the proprietary indexes for the MDEX Engine.
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.

MDEX Engine Component	Description
enecerts	The <code>enecerts</code> utility creates SSL certificates.

About the Information Transformation Layer

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

The transformation process does not change the content of your source data -- only its representation within your application.

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.

For more information about the Information Transformation Layer, refer to the *Platform Services Forge Guide*.

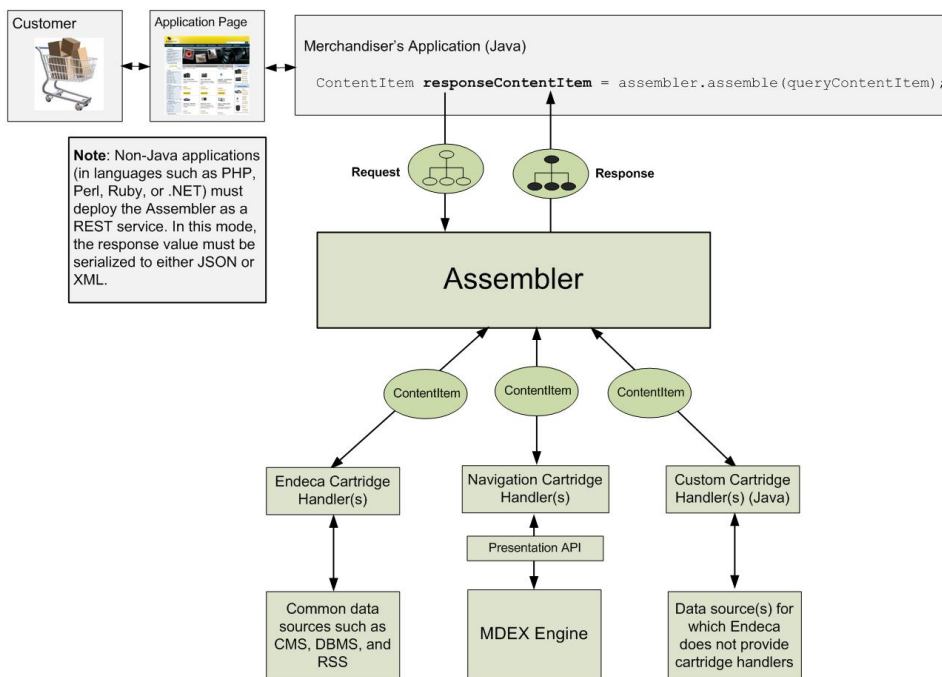
Chapter 2

Assembler functionality

The Assembler includes code for querying an MDEX Engine and returning information in a format that a front-end application can render for display on application pages. The Assembler acts as the central source for information that appears on the application page, apart from labels and other static elements of the application page templates.

How the MDEX Engine Communicates with the Assembler

The following figure illustrates the place of the Assembler in a Guided Search application:



As shown in the preceding diagram, the following things happen when the Assembler queries the MDEX Engine to obtain the information that a customer requests through Guided Search:

1. A customer makes a request for information through your application page.

2. To pass the customer's request to the Assembler, your application invokes the `assemble()` method as follows:

```
ContentItem result = assembler.assemble(contentItem)
```

3. The Assembler passes the request to the navigation cartridge handler.
4. The navigation cartridge handler invokes the Presentation API to pass the request to the MDEX Engine.
5. The MDEX Engine finds the requested information, filters and sorts it, and returns it to the navigation cartridge handler.
6. The navigation cartridge handler returns to the Assembler a content item containing the requested information.
7. The Assembler combines this content item and other content items into a result content item. The other content items may contain the results of other queries to the navigation cartridge handlers, or of queries to the cartridge handlers for other sources of information.
8. The Assembler returns the result content item to your application.
9. In your application, the rendering code for each cartridge handler identifies the information in the result content item that is to be displayed in the corresponding cartridge. The rendering code then converts this information ("renders" it) into a form that can be displayed.
10. Your application returns the rendered information to the application page to be displayed.

An application page typically requires several queries to the MDEX engine to retrieve all the content that it will display; these queries are made through a single call to the Assembler. The Assembler issues the individual queries and assembles the individual responses into a single result content item.

The result content item is a tree of content items, each of which contains information to be displayed in a particular cartridge. The content item tree can also contain content items from sources of information other than the MDEX Engine, such as RSS feeds, Database Management Systems (DBMS), Data Asset Managers (DAMs) and Content Management Systems (CMSs). For information about how to use the Assembler to manage information from such sources, refer to the *Assembler Developer's Guide*.

For information about the Assembler API, refer to the Assembler API help (javadoc). The Assembler API can be invoked directly in a Java environment, but it can also run as a RESTful service accessible by any language.

Part 2

Presentation API Basics

- [*Presentation API Overview*](#)
- [*Working with the Presentation API*](#)
- [*Using the Reference Implementation*](#)
- [*Running the Reference Implementations*](#)

Chapter 3

Presentation API Overview

The information about the Presentation API provided in this section is for the use of developers who must maintain existing applications that invoke the Presentation API directly, or who want to create functionality that is not included in the product by default. Oracle recommends that all new front-end application development use the Assembler API instead of the Presentation API.

List of Guided Search APIs

Depending on the packages you installed, your Guided Search installation includes one or more sets of APIs.

The Guided Search software packages contain the following API sets:

- The Presentation API. Although it is possible to invoke this API directly, Oracle recommends that you use the Assembler API instead. For information about how to use the Assembler, see the *Assembler Developer's Guide*.
- The Logging API that is used by the Logging and Reporting System. This API is part of Platform Services. For more information, see the *Platform Services Log Server and Report Generator Guide*.
- Security-related methods that implement secure Guided Search implementations. For more information, see the *Guided Search Security Guide*.

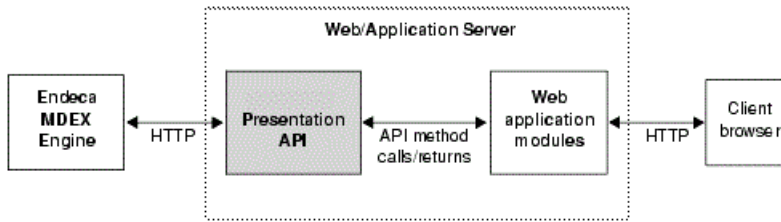
Architecture of the Presentation API

The Presentation API enables Web based Guided Search applications to communicate with the MDEX Engine.

The online portion of a typical Guided Search application has the following components:

- The MDEX Engine, which receives and processes query requests.
- The 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 Guided Search application that uses the 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 MDEX Engine, by means of 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, through the Web application server.



Note: For security reasons, you should never enable 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, by means of the Presentation API.

You build custom application modules for each Guided Search application. These modules can take many forms, depending on your application's requirements.

The Guided Search software package 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 must perform the following functions:

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

Methods for transforming requests into queries

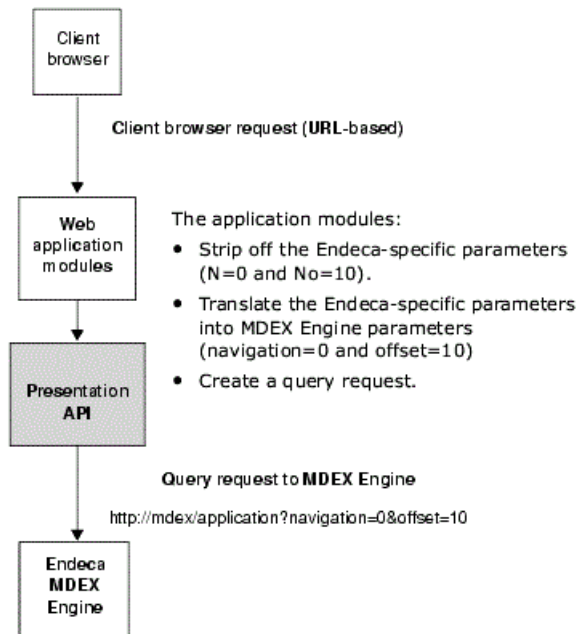
Application modules transform client browser requests into MDEX Engine queries.

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

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

The following diagram illustrates how a client browser request is transformed into an MDEX Engine query:

Transforming a Browser Request into an MDEX Engine Query



Techniques for passing request parameters

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

You can use one of the following techniques:

- 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 Oracle Commerce Platform Services package, client request parameters are embedded directly in the URL. This technique eases development and ensures load balancing, redundancy and statelessness.

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 need to build an entire page for a typical application.

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

- Indexed 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, and thus to improve performance. The performance improvement is gained by using 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.)

The Presentation API for Java and .NET

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

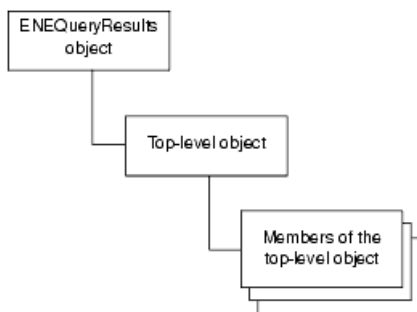
The Presentation API is managed by a Web application server of your choice. One or the other of the two forms of the Presentation API will be suitable for your chosen environment:

- 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.

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:



About top-level object types

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

You use 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.

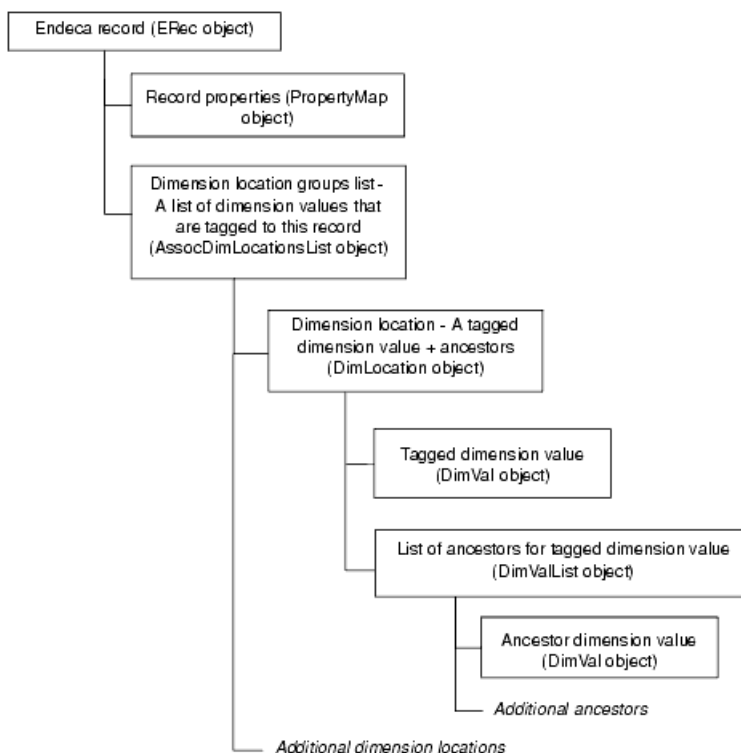
- *Record object* contain full information about individual records in the data set. This information includes the record's properties, as well as its tagged dimension values.

- *Aggregated record objects* contain information about aggregated records. An aggregated record is a collection of individual records that have been rolled up based on a rollup key (a property or dimension name).
- *Dimension search objects* contain the results of a dimension search.

Example of a top-level object

The Oracle Commerce record object returned by the MDEX Engine encapsulates information about dimensions that have been tagged to the Oracle Commerce record.

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



This diagram illustrates that Oracle Commerce record objects contain all the information associated with an Oracle Commerce 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.

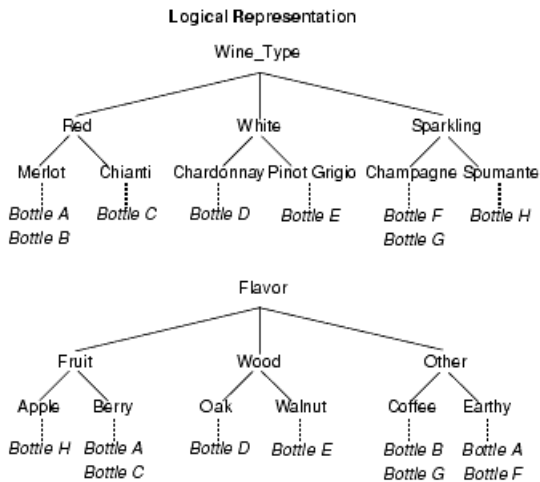
You can use the dimension hierarchy information in an Oracle Commerce record object to build follow-on navigation queries. For example, you can incorporate Find Similar functionality into your application by building a navigation query from the tagged dimension values for the current record.

Example of a record object for the wine data

To better understand a record object returned by the MDEX Engine, we can look at an example of a 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 wine data can be represented logically in a hierarchy of dimension values, as follows:



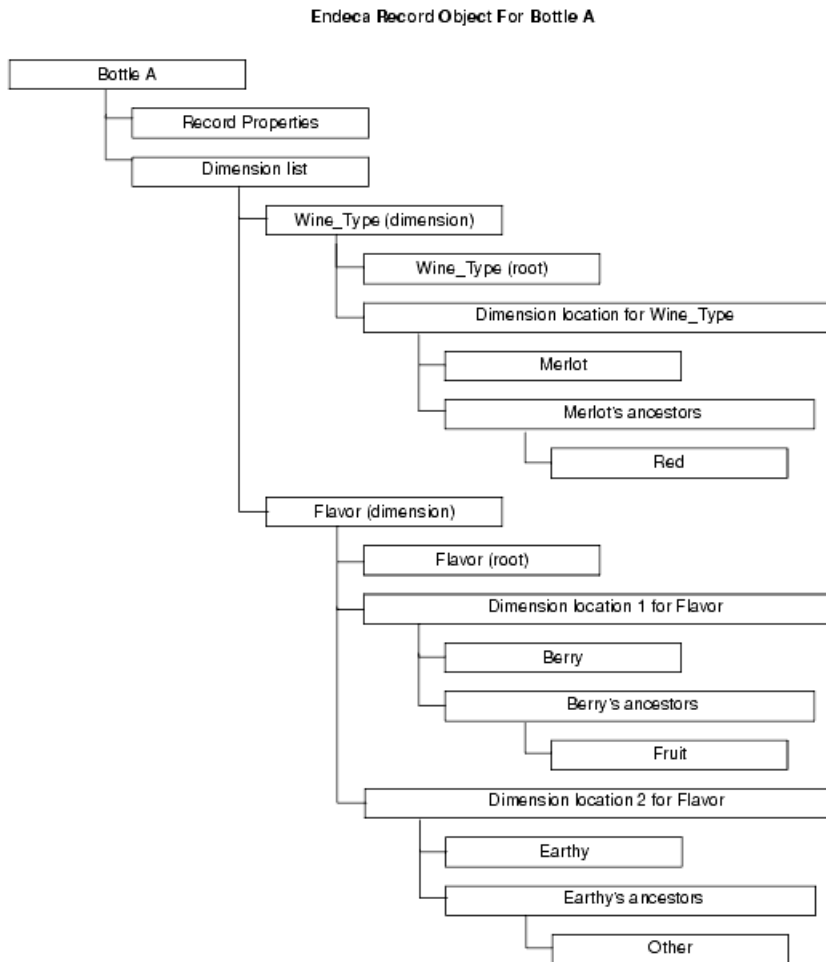
The wine data can be represented as records that correspond to physical bottles of wine, as follows:

Physical Representation

Bottle A Wine_Type: Merlot Flavor: Berry Flavor: Earthy	Bottle E Wine_Type: Pinot Grigio Flavor: Walnut
Bottle B Wine_Type: Merlot Flavor: Coffee	Bottle F Wine_Type: Champagne Flavor: Earthy
Bottle C Wine_Type: Chianti Flavor: Berry	Bottle G Wine_Type: Champagne Flavor: Coffee
Bottle D Wine_Type: Chardonnay Flavor: Oak	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 record object for Bottle A:



Obtaining additional object information

Understanding the contents of 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 about the top-level objects and all their members:

- [Presentation API for Java Reference \(Javadoc\)](#)
- [Presentation API for .NET \(HTML Help\)](#)

Chapter 4

Working with the Presentation API

This section describes how to use the Presentation API classes in a Web application module.

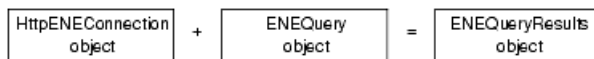
Core classes of the Presentation API

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

The Presentation API is based on three core classes:

- The `HttpENEConnection` class establishes 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 that 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 UriENEQuery

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

Creating the query with `UriENEQuery`

You use the `UriENEQuery` 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 UriENEQuery(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 UriENEQuery(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 `UriENEQuery` 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-Oracle Commerce 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.

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 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
...

```

Four basic queries

While the queries you send to an 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
Record query	ERec
Dimension search query	DimensionSearchResult
Aggregated 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: <code>controller.jsp?N=0 controller.jsp?N=123+456</code> .NET: <code>controller.aspx?N=0 controller.aspx?N=123+456</code>
Record search	R	The specifier (string-based ID) of the record to be returned.	Java: <code>controller.jsp?R=12345</code> .NET: <code>controller.aspx?R=12345</code>

Basic query type	URL param	Parameter definition	URL query string example
Dimension search	D	The dimension search terms.	Java: controller.jsp?D=red+wine .NET: controller.aspx?D=red+wine
Aggregated record search	A,An ,Au	<p>A: The specifier (string-based ID) of the aggregated Oracle Commerce record to be returned.</p> <p>An: The navigation descriptors that describe the record set from which the aggregated record is created.</p> <p>Au: The rollup key used to create the aggregated record.</p>	<p>Java: controller.jsp?A=123&An=456+789&Au=Name</p> <p>.NET: controller.aspx?A=123&An=456+789&Au=Name</p>


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	<p>Java: <code>setNavDescriptors(DimValIdList descriptors)</code></p> <p>.NET: <code>NavDescriptors</code></p>
Record search	<p>Java: <code>setERecSpec(String recordSpec)</code></p> <p>.NET: <code>ERecSpec</code></p> <p> Note: A <code>recordSpec</code>, or record specifier, is a string-based identifier.</p>
Dimension search	<p>Java: <code>setDimSearchTerms(String terms)</code></p> <p>.NET: <code>DimSearchTerms</code></p>
Aggregated record	Java:

Basic query type	Required methods (Java) or properties (.NET)
	<code>setAggrERecSpec(String aggregatedRecordSpec), setAggrERecNavDescriptors(DimValIdList descriptors), setAggrERecRollupKey(String key)</code> .NET: <code>AggrERecSpec, AggrERecNavDescriptors, AggrERecRollupKey</code>

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 and properties, their corresponding returned object types, and examples showing usage in Java and .NET.



Note: See the *Presentation API for Java Reference (Javadoc)* and *API Reference for .NET (HTML Help)* for complete information about all Presentation API classes, method (Java), and properties (.NET).

Method (Java) or property (.NET) convention	Returned object (type)	Examples
Java: <code>setERec...()</code> .NET: <code>ERec...</code>	<code>ERec</code>	Java: <code>setERecs(), setERecSpec()</code> .NET: <code>ERecs, ERecSpec</code>
Java: <code>setNav...()</code> .NET: <code>Nav...</code>	<code>Navigation</code>	Java: <code>setNavNumERecs()</code> .NET: <code>NavNumERecs</code>
Java: <code>setDimSearch...()</code> .NET: <code>DimSearch...</code>	 Note: This object has been deprecated.	Java: <code>setDimSearchTerms()</code> .NET: <code>DimSearchTerms</code>
Java: <code>setAggrERec...()</code> .NET: <code>AggrERec...</code>	<code>AggrERec</code>	Java: <code>setAggrERecRollupKey()</code> .NET: <code>AggrERecRollupKey</code>

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 <code>ENEQueryResults</code> methods or properties:
<code>N</code> or Java: <code>setNavDescriptors()</code> .NET: <code>NavDescriptors</code>	Java: <code>getNavigation()</code> .NET: <code>Navigation</code>
<code>R</code> or Java: <code>setERecSpec()</code> .NET: <code>ERecSpec</code>	Java: <code>getERecSpec()</code> .NET: <code>ERecSpec</code>
<code>D</code> or Java: <code>setDimSearchTerms()</code> .NET: <code>DimSearchTerms</code>	Java: <code>getDimensionSearch()</code> .NET: <code>DimensionSearch</code>
<code>A</code> , <code>An</code> , <code>Au</code> or Java: <code>setAggrERecSpec()</code> <code>setAggrERecNavDescriptors()</code> <code>setAggrERecRollupKey()</code> .NET: <code>AggrERecSpec</code> <code>AggrERecNavDescriptors</code> <code>AggrERecRollupKey</code>	Java: <code>getAggrERecSpec()</code> .NET: <code>AggrERecSpec</code>

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: <code>ENEQuery</code> object: <code>containsNavQuery()</code> <code>ENEQueryResults</code> object: <code>containsNavigation()</code> .NET:	<code>N</code> or Java: <code>setNavDescriptors()</code> .NET: <code>NavDescriptors</code>

If these methods evaluate to true:	Your query uses:
ENEQuery object: containsNavQuery() ENEQueryResults object: ENEQueryResults object:ContainsNavigation()	
Java: ENEQuery object: containsERecQuery() ENEQueryResults object: containsERec() .NET: ENEQuery object: ContainsERecQuery() ENEQueryResults object: ContainsERec()	R or Java: setERecSpec() .NET: ERecSpec
Java: ENEQuery object: ENEQuery object:containsDimSearchQuery() ENEQueryResults object: ENEQueryResults object:containsDimensionSearch() .NET: ENEQuery object: ENEQuery object:ContainsDimSearchQuery() ENEQueryResults object: ENEQueryResults object:ContainsDimensionSearch()	D or Java: setDimSearchTerms() .NET: DimSearchTerms
Java: ENEQuery object: containsAggrERecQuery() ENEQueryResults object: containsAggrERec() .NET: ENEQuery object: ContainsAggrERecQuery() ENEQueryResults object: ContainsAggrERec()	A, An, Au or Java: setAggrERecSpec() setAggrERecNavDescriptors() setAggrERecRollupKey() .NET: AggrERecSpec AggrERecNavDescriptors AggrERecRollupKey

Getting started with your own Web application

This section provides basic information about how to create a Guided Search application.

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

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.

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
<code>ENEException</code>	Indicates an exception from the MDEX Engine. This means that <code>ENEConnection</code> was able to contact the MDEX Engine but the MDEX Engine responded with an error.

Exception	Description
<code>ENEAuthenticationException</code>	Indicates an authentication exception from the MDEX Engine. This means that <code>ENEConnection</code> was able to contact the MDEX Engine but the MDEX Engine responded with an authentication error.
<code>ENEQueryException</code>	Indicates any connection problems in this method.
<code>ENEConnectionException</code>	Indicates a communication error in the <code>ENEConnection</code> with the MDEX Engine.
<code>EmptyENEQueryException</code>	Indicates that the <code>query()</code> method in Java, and the <code>Query()</code> method in .NET were called using an empty <code>ENEQuery</code> object. This exception occurs because the <code>ENEQuery</code> object did not express any requests to the MDEX Engine.
<code>PartialENEQueryException</code>	Indicates that the <code>ENEQuery</code> object does not contain all the necessary query parameters.
<code>UrlENEQueryParseException</code>	Indicates an error while parsing a browser request query string into individual MDEX Engine query parameters.
<code>VersionMismatchException</code>	Indicates the presence of incompatible modules in the Guided Search 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.

Chapter 5

Using the Reference Implementation

This section describes the reference implementation, its components, and what you need to know to use it.

Reference implementation overview

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

The 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 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.

The purpose of the reference implementation

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

The 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 Guided Search prototype.

The Java version

The Java version of the 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 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 record.

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
<code>eneHost</code>	<p>Used by the <code>misc_ene_switch</code> module to dynamically set the MDEX Engine hostname with each request.</p> <p>This parameter is particularly useful during development, but should be removed from a production deployment.</p>
<code>enePort</code>	<p>Used by the <code>misc_ene_switch</code> module to dynamically set the MDEX Engine port with each request.</p> <p>As with <code>eneHost</code>, this parameter is particularly useful during development, but should be removed from a production deployment.</p>
<code>displayKey</code>	<p>Used by <code>nav_records</code> and <code>nav_supplemental</code> to identify the property key that should be used to represent the name of a record.</p> <p>This parameter is useful for data inspection where different data sets may require different property keys to name the records.</p> <p>You should remove the <code>displayKey</code> parameter from a production deployment as the record names should never change.</p>
<code>hideProps</code>	<p>Provides a simple means of hiding properties for each record in the <code>nav_records</code> module.</p>

Parameter	Description
hideSups	Provides a simple means of hiding the data that is returned with each supplemental object in the <code>nav_supplemental</code> module.
hideMerch	Provides a simple means of hiding the data that is returned with each supplemental merchandising object in the <code>nav_merch</code> module.

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, Oracle recommends that you use only 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.

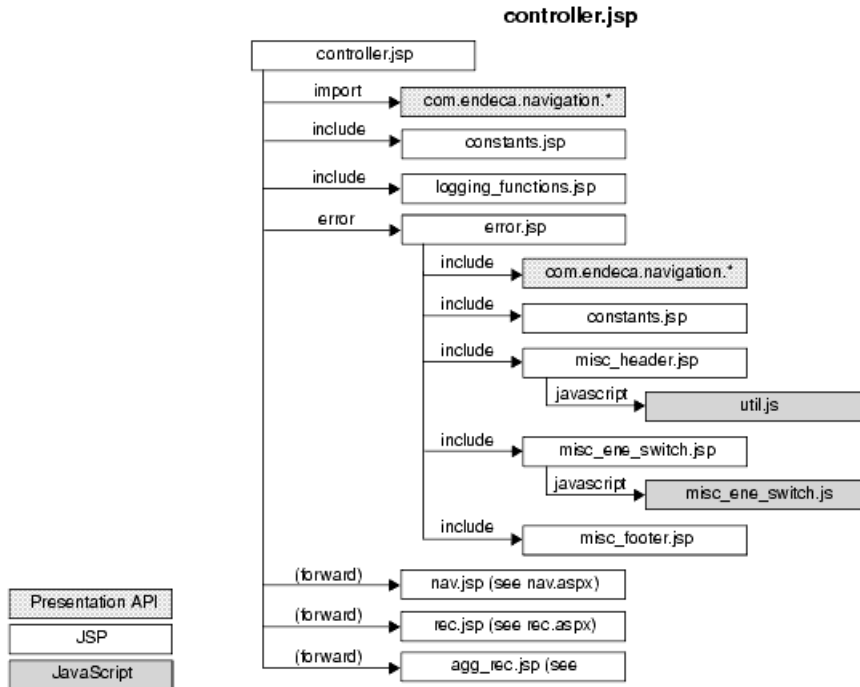
Module maps

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

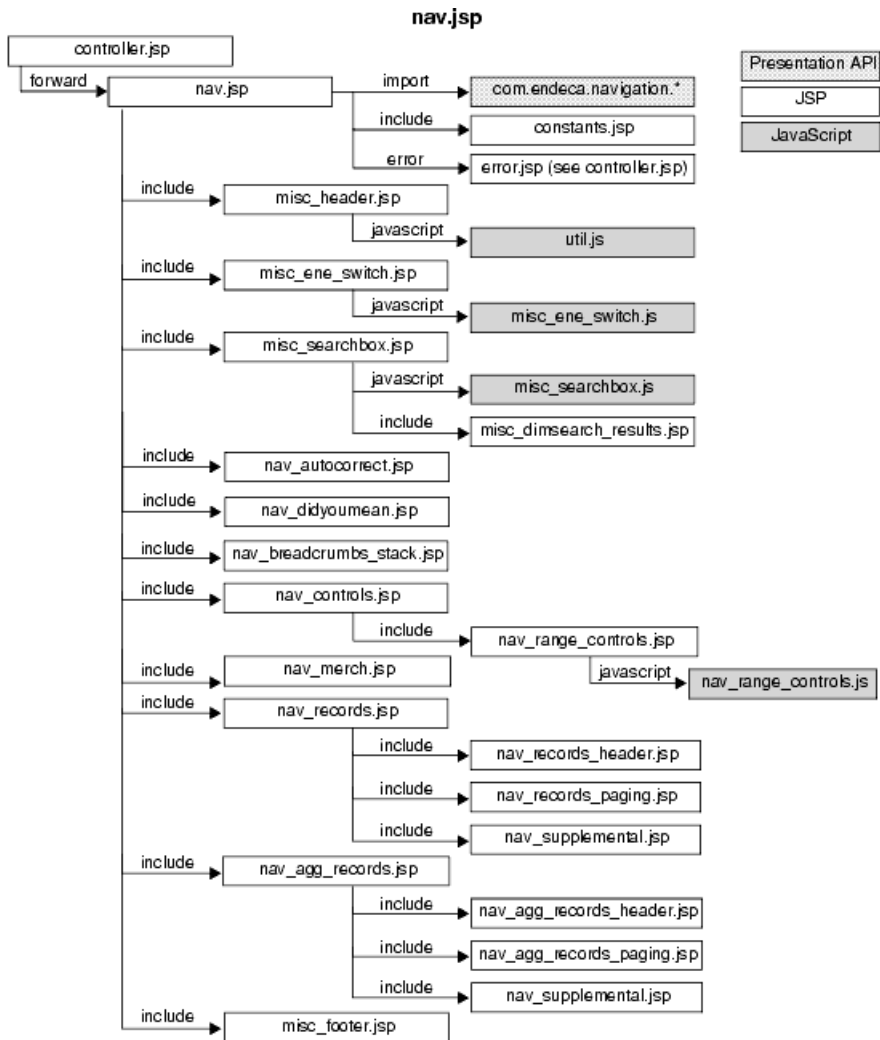
The `controller.jsp` (Java) module is the entry point into the UI reference implementation. It receives the browser request from the application server, formulates the MDEX Engine query, establishes a connection with the MDEX Engine and sends the query. Based on the contents of the query results, the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



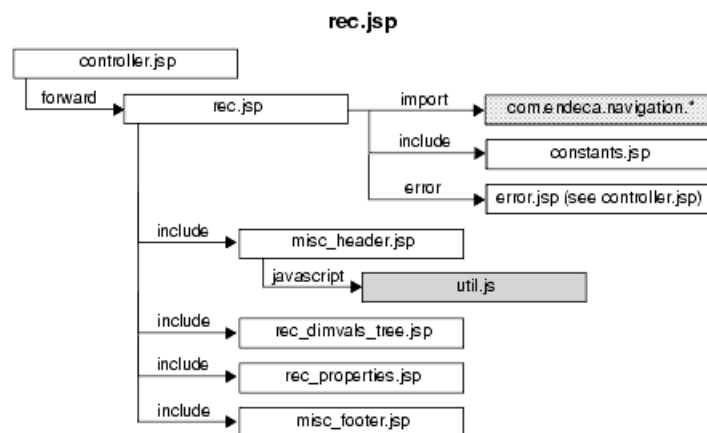
The `nav.jsp` (Java), using other included `nav` modules, renders the main navigation page, including the navigation controls, navigation descriptors, and a record set.

The following diagram shows the `nav` module map:



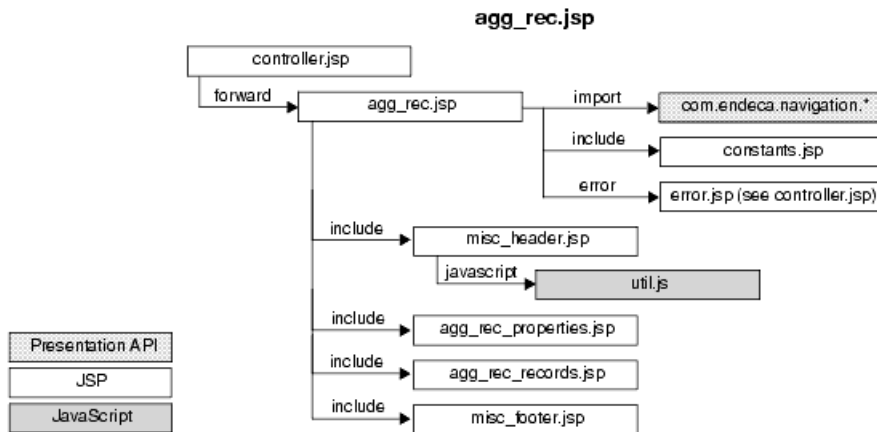
For record requests, **controller** forwards the request to the **rec.jsp** (Java) module which, along with its child **rec_*** modules, is responsible for rendering a record page for a single record.

The following diagram shows the **rec** module map:



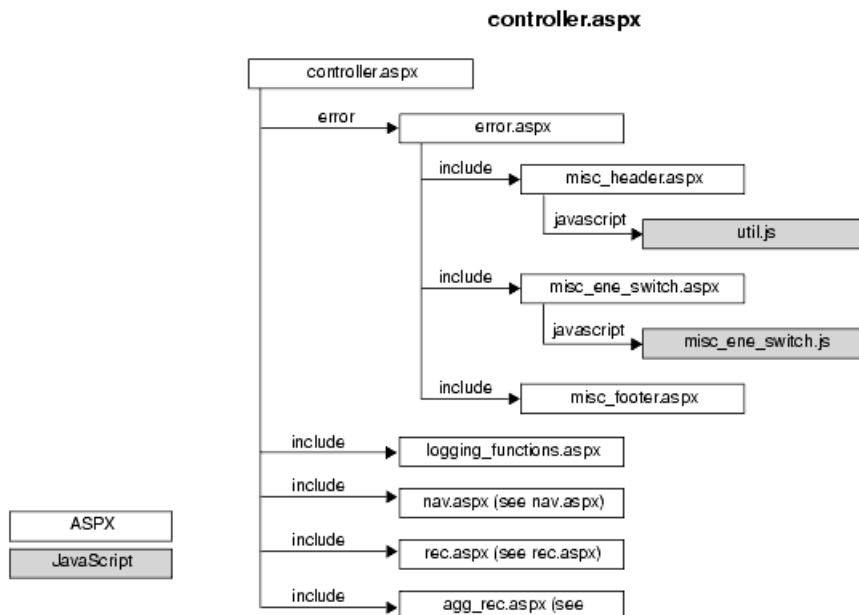
For aggregated record requests, `controller` forwards the request to the `agg_rec.jsp` (Java) module which, again, along with its child `agg_rec_*` modules, renders a page for an aggregated record.

The following diagram shows the `agg_rec` module map:

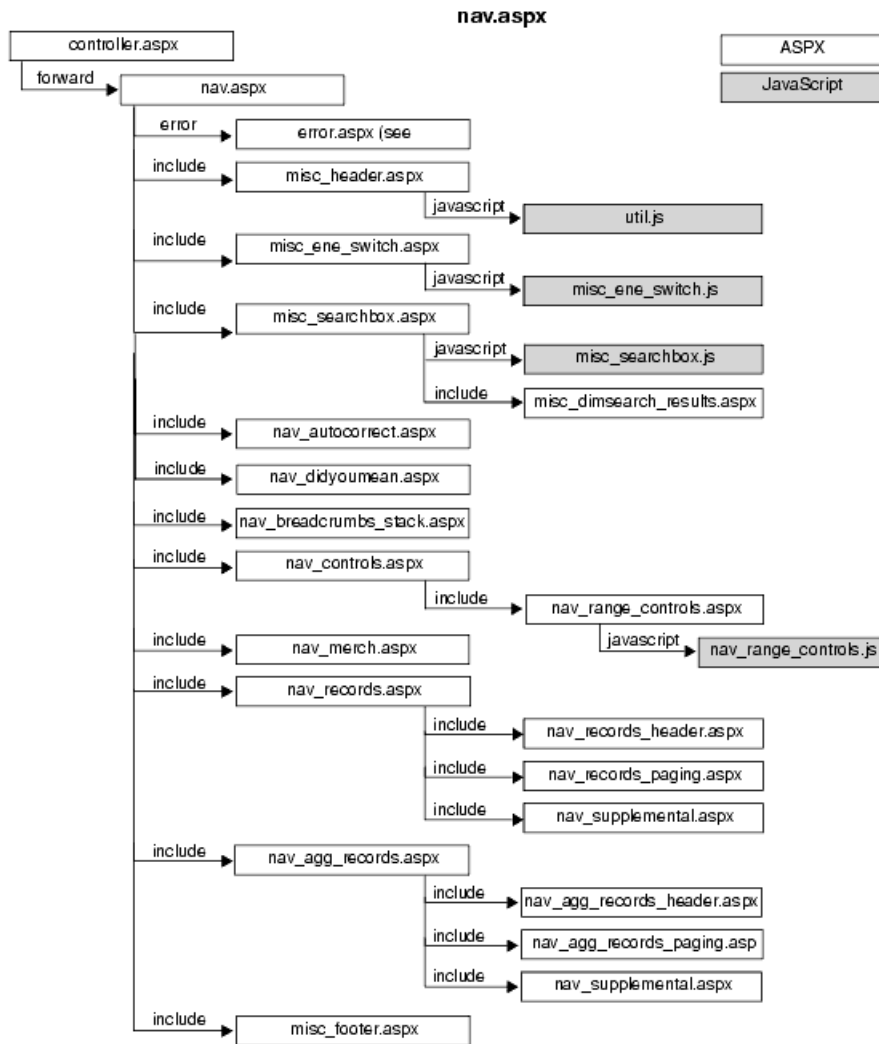


.NET module maps

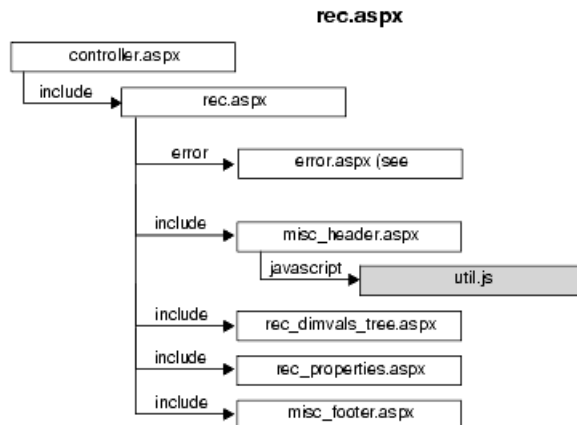
The following diagram shows the `controller` module map:



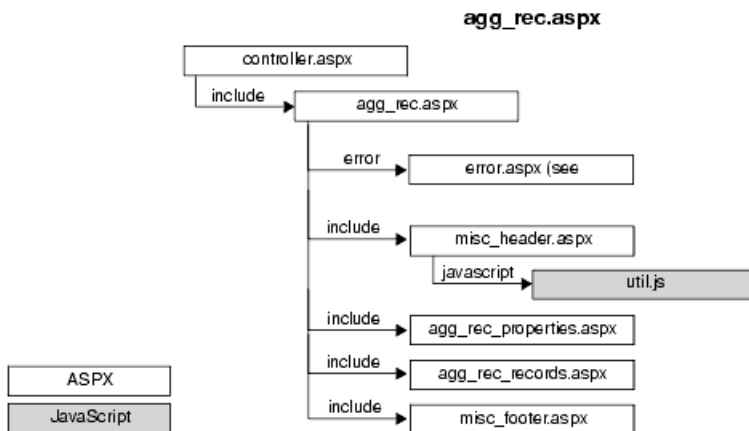
The following diagram shows the `nav` module map:



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 is 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.aspx	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 Oracle Commerce report element.
misc_ene_switch and misc_ene_switch.js	Render the MDEX Engine switching widget that enables 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 <code>nav_breadcrumbs_stack</code>
nav_range_controls and nav_range_controls.js	Renders a set of controls that enable 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
<code>nav_records_header</code>	Displays a record count and other controls to handle the record set display. Also displays an aggregated record count when records have been aggregated.
<code>nav_records_paging</code>	Displays controls for paging through the record set, when applicable.
<code>nav_agg_records</code>	Renders a list of records that have been aggregated based on a rollup key.
<code>nav_agg_records_header</code>	Displays a record count and other controls to handle the record set display along with an aggregated record count.
<code>nav_agg_records_paging</code>	Displays controls for paging through a list of aggregated records, when applicable.
<code>misc_searchbox</code> and <code>misc_searchbox.js</code>	Render a basic searchbox widget.
<code>misc_dimsearch_results</code>	Displays the results of a dimension search.
<code>rec</code>	Displays a record page for an individual record.
<code>rec_dimvals_trees</code>	Displays the dimension values that have been tagged to the current record.
<code>rec_properties</code>	Displays the properties for the current record.
<code>agg_rec</code>	Displays an aggregated record page for one aggregated record.
<code>agg_rec_properties</code>	Displays the properties associated with an aggregated record's representative record. Displays properties derived from performing calculations on the aggregated record's constituent records.
<code>agg_rec_records</code>	Displays the constituent records associated with the current aggregated record.
<code>coremetrics</code>	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.

Chapter 6

Running the Reference Implementations

You can use the sample reference implementation (a Web application) to verify that your Guided Search components are installed and working properly. The reference applications are included as part of the Platform Services package. Updated APIs for the reference applications are distributed with the Presentation API.

Running the JSP reference implementation

The JSP reference application can be installed in an application server with J2EE support such as Apache Tomcat. This section differs from the "Verifying your installation with the JSP reference application" section in that here we assume that you are installing the JSP reference implementation to a standalone version of the Tomcat Web server.

Setting up the JSP reference implementation on Windows

While this section assumes that you use the Tomcat server, you can use other application servers.

The JSP reference implementation depends on several paths related to the Tomcat Web server and Java SDK. This section assumes the following paths in your environment:

The location of the Tomcat installation	<code>C:\jakarta-tomcat-version</code>
The location of the Java SDK installation	<code>C:\j2sdk-version</code>

In the following procedures, adjust the paths as needed for your environment.

To set up the JSP reference implementation:

1. Copy the reference implementation user interface directory `%ENDECA_REFERENCE_DIR%\endeca_jspref` into the `C:\jakarta-tomcat-version\webapps` directory.
The `%ENDECA_REFERENCE_DIR%` variable is set as part of the Platform Services installation.
2. (Optional.) Navigate to `C:\jakarta-tomcat-version\conf` and open the `server.xml` file in a text editor. You can modify the file as follows:
 - a) Change the port that Tomcat listens on for a shutdown command from its default of 8005:

```
<Server port="8005" shutdown="SHUTDOWN">
```

- b) Change the Tomcat HTTP listening port from its default of 8080:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->  
<Connector port="8080" ...
```

- c) Save and close the `server.xml` file.
3. If your version of Java requires it, make sure that the `JAVA_HOME` environment variable is set to the location of the Java SDK directory. For example, the location might be `C:\j2sdk-version`.



Note: See the Tomcat documentation for more information about your version of the Tomcat server to check if it requires a `JAVA_HOME` environment variable.

To set the `JAVA_HOME` environment variable:

- a) From the Windows Control Panel, select **System**.
- b) Go to the **Advanced** tab and select **Environment Variables**.
- c) In the **System Properties** section, locate and select `JAVA_HOME`.
 - If `JAVA_HOME` does not exist, select **New**, and then in the **Variable Name** field, enter `JAVA_HOME`
- d) In the **Variable Value** field, enter the path of the Java SDK directory and click **OK**.
- e) Click **OK** to close the **Environment Variables** window.
- f) Click **OK** to close the **System Properties** window.
4. Copy the following file from the `PresentationAPI\<version>\java\lib` directory to `C:\jakarta-tomcat-version\webapps\endeca_jspref\WEB-INF\lib`:
 - `endeca_navigation.jar` (Presentation API)
5. Copy the following file from the `$ENDECA_ROOT/lib/java/` directory to `C:\jakarta-tomcat-version\webapps\endeca_jspref\WEB-INF\lib`:
 - `endeca_logging.jar` (Logging API)
6. Copy the following Report Generator file from the `%ENDECA_ROOT%\lib\java` directory to `C:\jakarta-tomcat-version\webapps\endeca_jspref\WEB-INF\lib`:
 - `rg.jar`
7. Start the Tomcat server. See the Tomcat documentation for specific instructions.

The JSP reference implementation is set up and you can now test your Guided Search installation with it.

Setting up the JSP reference implementation on UNIX

While this section assumes that you use the Tomcat server, you can use other application servers.

The JSP reference implementation depends on several paths related to the Tomcat Web server and Java SDK. This section assumes the following path names:

The location of the Tomcat installation	<code>/usr/local/tomcat-version</code>
The location of the Java SDK installation	<code>/usr/local/j2sdk-version</code>



Note: The Java SDK installation must consist of the entire JDK, and not just the location of a copied or linked Java binary.

To set up the JSP reference implementation:

1. Copy the reference implementation from `$ENDECA_REFERENCE_DIR/endeca_jspref` to the Tomcat `/webapps` directory (for example, `/usr/local/tomcat-version/webapps`).

The `$ENDECA_REFERENCE_DIR` variable is set as part of the Platform Services installation.

2. (Optional.) Go to the `/usr/local/tomcat-version/conf` directory and open the `server.xml` file in a text editor. You can modify the file as follows:

- a) Change the port that Tomcat listens on for a shutdown command from its default of 8005:

```
<Server port="8005" shutdown="SHUTDOWN">
```

- b) Change the Tomcat HTTP listening port from its default of 8080:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" ...
```

- c) Save and close the `server.xml` file.

3. Set the appropriate Tomcat environment variables.

- For `csh` and similar shells, set:

```
setenv JAVA_HOME /usr/local/j2sdk-version
setenv CATALINA_BASE /usr/local/tomcat-version
```

- For `bash`, set:

```
export JAVA_HOME=/usr/local/j2sdk-version
export CATALINA_BASE=/usr/local/tomcat-version
```

Generally these commands should be placed in a script run at the startup of the shell so that the variables are set for future use.

4. Copy the following file from the `PresentationAPI/<version>/java/lib` directory to `/usr/local/tomcat-version/webapps/endecca_jspref/WEB-INF/lib`:

- `endecca_navigation.jar` (Presentation API)

5. Copy the following file from the `$ENDECA_ROOT/lib/java/` directory to `/usr/local/tomcat-version/webapps/endecca_jspref/WEB-INF/lib`:

- `endecca_logging.jar` (Logging API)

6. Copy the following Report Generator file from the `$ENDECA_ROOT/lib/java` directory to `/usr/local/tomcat-version/webapps/endecca_jspref/WEB-INF/lib`:

- `rg.jar`

7. Start the Tomcat server.

The JSP reference implementation is set up and you can now test your Guided Search installation with it.

Enabling the Analytics controls in the JSP reference implementation

The JSP reference implementation includes a set of Analytics controls that are not displayed by default. These controls are useful for learning about, developing, and debugging Analytics statements.

These instructions pertain to the JSP reference implementation that runs under the Tools Service. If your JSP reference is running on a standalone Tomcat, use the same instructions, substituting the path names in your Tomcat installation for the ones below

To enable the Analytics controls in the JSP reference implementation:

1. After installing the Workbench package, place `CordaEmbedder.jar` in this directory:

- Windows: %ENDECA_TOOLS_ROOT%\server\webapps\endeca_jspref\WEB-INF\lib
- UNIX: \$ENDECA_TOOLS_ROOT/server/webapps/endeca_jspref/WEB-INF/lib



Note: This file is available as part of the Corda Server installation package and is required by the reference implementation even if you do not intend to use charts.

2. Edit the `web.xml` file (which is in the WEB-INF directory from step 1) and add the definition of the `eneAnalyticsEnabled` parameter, as in this example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- This file identifies these directories as containing
a Web application. -->
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <context-param>
    <param-name>eneAnalyticsEnabled</param-name>
    <param-value>1</param-value>
    <description>Flag to enable Oracle Commerce Analytics controls</description>
  </context-param>
</web-app>
```

3. Restart the Tools Service.
4. In a Web browser, navigate to the JSP reference implementation. The Analytics controls will be visible.

Verifying your installation with the JSP reference application

After you have successfully run a baseline update and started the Guided Search components, you can use the JSP reference implementation to navigate and search your data.

The JSP reference application is installed as part of Workbench installation and runs in the Tools Service.

To verify a Guided Search setup with the JSP reference application:

1. Open a Web browser.
2. In the Address box, enter the following URL:
`http://WorkbenchHost:8006/endeca_jspref`
 Replace *WorkbenchHost* with the name of the machine that is running Workbench. If you used a different port when you configured Workbench, substitute that port for 8006.
 This URL brings you to a page with a link called **ENDECA-JSP Reference Implementation**.
3. Click the **ENDECA-JSP Reference Implementation** link.
4. Enter the host name and port of the machine that the MDEX Engine is running on. For example, enter `localhost` and `15000`. Click **Go**.

You should see the reference implementation displaying application data.

Running the ASP.NET reference implementation

The ASP.NET reference implementation runs in IIS 6.0 on Windows Server 2003 64-bit systems, and requires some configuration before you deploy the application.

Configuring the 64-bit version of ASP.NET

Before you set up the reference application, make sure you have enabled the 64-bit version of ASP.NET.

The ASP.NET reference implementation supports versions 2.0 SP1, 3.0, and 3.5 of ASP.NET.

To install the 64-bit version of ASP.NET:

1. From a command prompt, issue the following command to disable 32-bit mode:

```
cscript %SYSTEMDRIVE%\inetpub\adminscripts\adsutil.vbs SET W3SVC/AppPools/Enable32bitAppOnWin64 0
```
2. Issue the following command to install the 64-bit version of ASP.NET 2.0 and to install the script maps at the IIS root:

```
%SYSTEMROOT%\Microsoft.NET\Framework64\v2.0.50727\aspnet_regiis.exe -i
```



Note: The .NET DLLs packaged with this release are compiled using the 64-bit version of the .NET Framework. They should be compatible with .NET Frameworks 2.0 SP1, 3.0, and 3.5.

Setting up the ASP.NET reference implementation

In this section we assume that you are using IIS 6.0 and .NET 2.0. The reference implementation supports versions 2.0 SP1, 3.0, and 3.5 of ASP.NET.

You must make sure that the 64-bit version of ASP.NET is configured and that you have enabled the ASP pages as an extension in the Microsoft IIS before proceeding with setup of the ASP.NET reference implementation.

To set up the ASP.NET reference implementation:

1. Copy all the `Endeca.Navigation.*.dll` files from `PresentationAPI\<version>\dotNet\lib` to: `C:\Endeca\PlatformServices\reference\endeca_ASP.NETref\bin`.
2. Copy the `Endeca.Logging.dll` file from `$ENDECA_ROOT/lib/Endeca.NET/` to: `C:\Endeca\PlatformServices\reference\endeca_ASP.NETref\bin`.
3. Modify the following IIS settings:
 - a) From the Windows Control Panel, select **Administrative Tools > Internet Information Services**.
 - b) In the **Internet Information Services** tree pane, expand the machine icon for the local machine.
 - c) Right-click **Default Website**.
 - d) Select **New > Virtual Directory**.



Note: If you are using IIS 7, you should create an **Application** rather than a **Virtual Directory**.

- e) Fill in the following fields in the **Virtual Directory Creation** wizard as follows:

Field	Value
Virtual Directory Alias	<code>endeca_ASP.NETref</code>
Website Content Directory	Browse to the location of the ASP.NET reference implementation. The default location is: <code>c:\Endeca\PlatformServices\reference\endeca_ASP.NETref</code>
Access Permissions	Leave the default settings in place.

The Virtual Directory Creation wizard opens.

- f) Click **Next**, then click **Finish**.
- g) In the IIS Manager MMC snap-in, to set the virtual directory name as an application name, right-click the virtual directory, and select **Virtual Directory > Application settings > Create**. The application name can be set to any name, and you can use the alias you used for the virtual directory as an example. Set **Execute Permissions** to **Scripts Only**.
- h) Close the Internet Information Services window.

The ASP.NET reference implementation is set up and you can now test your Guided Search installation with it.

Testing your Guided Search installation with the ASP.NET reference implementation

After you have set up the ASP.NET reference implementation, you can test your Guided Search installation with it.

To test the Guided Search installation with the ASP.NET reference implementation:

1. Open Internet Explorer.
2. Navigate to the following location: `http://EndecaServerNameorIP/endeca_ASP.NETref`
`EndecaServerNameorIP` refers to the machine on which you set up the reference application.
For example, assuming that you use the default IIS port of 80: `http://localhost/endeca_ASP.NETref`
3. From here, click **Oracle Commerce .NET Reference Implementation** to launch the ASP.NET Reference Implementation.
The ASP.NET Reference Implementation asks you for a host and port of the MDEX Engine server.
4. Enter the host name as the server name or IP of the machine on which you installed the MDEX Engine.
5. Enter the port number you specified for the MDEX server in the Deployment Template `AppConfig.xml` or in the `remote_index.script` control script. This is the port on which the MDEX Engine accepts queries.
6. Click **Go**.
The ASP.NET reference implementation opens.

Record Features

- *[Working with Oracle Commerce Records](#)*
- *[Sorting Oracle Commerce Records](#)*
- *[Using Range Filters](#)*
- *[Creating Aggregated Records](#)*
- *[Controlling Record Values with the Select Feature](#)*
- *[Using the Oracle Commerce Query Language](#)*
- *[Record Filters](#)*
- *[Bulk Export of Records](#)*

Working with Oracle Commerce Records

This section describes how to manage Oracle Commerce records in your Web application.

Displaying Oracle Commerce records

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

Displaying a list of Oracle Commerce records

Displaying a list of Oracle Commerce records is a common task in any Oracle Commerce application.

A typical Oracle Commerce application displays a list of records in the user's current navigation state. That state initially matches the user's search query. The user may subsequently refine the navigation state by clicking named links to subsets of this list of records. The subsets are known as refinements.

For example, a search for "35mm film" might return, in addition to a list of brands of 35mm film, a list of refinements such as "color" and "black and white". By clicking the refinement "color", the user refines the navigation state and limits the list of film brands to color films. The user might then click the refinements "print film" or "slide film" to limit the list of film brands to color films for either prints or slides.

The record list can be 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 by every query to the MDEX Engine. Your application can iterate through this list, extract the identifying information for each record, and display a table that contains the results.

Displaying each record in the ERecList object

An MDEX Engine query returns a list of records in an `ERecList` (Oracle Commerce records) or `AggrERecList` (aggregated Oracle Commerce records) object.

The lists of records are returned in a `Navigation` object. To retrieve lists from a `Navigation` object, use these methods:

- 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 because the Java versions of `ERecList` and `AggrERecList` inherit from `java.util.AbstractList`, all the iterator and indexing methods are available for these objects.

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 Oracle Commerce 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 the MDEX Engine returns will affect performance.

To reduce the number of records returned by the MDEX engine – and thus, to reduce the amount of time that the MDEX engine requires to process requests – write your requests to return only the subset of records that you are interested in displaying to the user.

To do this, use the `setNavNumERecs()` method (Java) or the `NavNumERecs` property (.NET) and the offset specified in the `setNavERecsOffset()` method (Java) or the `NavERecsOffset` property (.NET).

Displaying record properties

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

Properties are key/value pairs associated with Oracle Commerce records. Property values provide detailed information about a record that your application can display when a user accesses a record by searching for or navigating to it. Properties generally contain more detail about a record than dimension values, which are used for navigation.

For example, a record can have properties named Product Description, Price, and Part Number, each with an appropriate value. The values of these properties can be displayed when a user finds the record, but they are too specific to be useful for navigation. A dimension named Price Range, however, can help a user navigate to records with specific prices (stored as property values), which are displayed when the user finds the record.



Note: 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 (dimension values) 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 processes the properties of the source data records that are read by the pipeline. You can configure the property mapper to process source record properties in any of the following ways:

- Map the source data property to an existing Oracle Commerce dimension or a newly-created Oracle Commerce dimension.
- Map the source data property to an existing Oracle Commerce property or a newly-created Oracle Commerce property. You can also specify how the property is displayed.
- Ignore the source data property.

The property mapper is part of Developer Studio. See the Design Studio online help for information about how to use the property mapper.

For information about how to add and configure Oracle Commerce properties, see the *Platform Services 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 indexes. If you use the Dgidx `--nostrictattrs` flag, every property found on a record will be indexed.

The MDEX Engine dgraph program does not have configuration flags to control how properties are displayed.

Accessing properties from records

Properties can be accessed from any Oracle Commerce 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];
        }
    }
}
```

```

    }
  }
}

```

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() %>:&nbsp;</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 = propsMap.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 %>:&nbsp;</td>
        <td><%= prop.Value %></td>
        </tr><%=
    }
}
}

```

Displaying dimension values for Oracle Commerce records

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

Dimensions are the hierarchical, navigable concepts applied to Oracle Commerce 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 enable 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. Users can then check the boxes for the dimension values in which they are interested; for example:

```

Sleeve=short
Style=Oxford
Size=L

```

When the user checks these dimension values, a record set is returned 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 create and configure dimensions using Developer Studio.

Dimensions and their hierarchy of values are created in Developer Studio Dimensions view, and are referenced in a dimension adapter component. See the *Platform Services Forge Guide* for more information about 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 Oracle Commerce 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 when a query returns a large number of records.

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:

Page 2 of 27: << [Prev](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [Next](#) >>

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
```



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 8

Sorting Oracle Commerce Records

The sorting functionality enables the user to define the order of Oracle Commerce 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 about implementing this feature can be found in the Developer Studio online help.

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 Oracle Commerce 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 Oracle Commerce record does not include a value for the specified sort key, that record is sorted to the bottom of the list, regardless of the sort order.

For example, the following record set is sorted by 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 Oracle Commerce 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.

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)
```

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 indexing whether the current request is being sorted by the given sort key, and an integer indexing 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 `ENQuery.getNavActiveSortKeys()` method
- .NET: the `ENQuery.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();
}
```

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.

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.

Using geospatial sorting

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

Geocode properties represent latitude and longitude pairs to Oracle Commerce 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 Oracle Commerce 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, Oracle Commerce'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 Oracle Commerce'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 9

Using Range Filters

You can use range filters for navigation queries.

About range filters

Range filter functionality enables 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, GCBWTN). 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: *lat* 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.
- GCBWTN – 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 Oracle Commerce'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...
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 permitted 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 enables 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 10

Creating Aggregated Records

The Oracle Commerce aggregated records feature enables the end user to group records by dimension or property values.

About aggregated records

You can eliminate duplicate displays by combining, into a single aggregated record, the records that contain the information in each separate display.

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 Oracle Commerce record. When querying the MDEX Engine, you may want to treat these instances as a single record. This is accomplished by combining the records for the different formats into an Oracle Commerce aggregated record. The MDEX Engine handles this aggregated record as though it were a single record.

An aggregated record is based on a rollup key. A rollup key can be any property or dimension that has its rollup attribute enabled.

From a performance perspective, aggregated Oracle Commerce 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 Oracle Commerce record creation by enabling 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 about 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 Oracle Commerce 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 Oracle Commerce 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",aggrc.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", aggreg.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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 `--stat-abins` 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 Oracle Commerce record.

You display the Oracle Commerce records (ERec objects) in an aggregated record with the same procedures described in Chapter 5 ("Working with Oracle Commerce 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() %>
        </td>
        </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 %>
        </td>
        </tr>
        <%
    }
}
```



```
}
}
```

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.

Controlling Record Values with the Select Feature

This section describes how to use the Select feature for selecting specific keys (Oracle Commerce properties and/or dimensions) from the data so that only a subset of values is returned for Oracle Commerce records in a query result set.

About the Select feature

Your application can return record sets based on specific keys.

A set of Oracle Commerce records is returned with every navigation query result. By default, each record includes the values from all the keys (properties and dimensions) that have record page and record list attributes. These attributes are set with the **Show with Record** (for record page) and **Show with Record List** (for record list) checkboxes, as configured in Developer Studio.

However, if you do not want all the key values, you can control the characteristics of the records returned by navigation queries by using the Select feature.

The Select feature enables you to select specific keys (Oracle Commerce properties and/or dimensions) from the data so that only a subset of values will be transferred for Oracle Commerce records in a query result set. The Select functionality allows the application developer to determine these keys dynamically, instead of at dgraph startup. This selection overrides the default record page and record list fields.

A Web application that does not make use of all of the properties and dimension values on a record can be more efficient by only requesting the values that it will use. The ability to limit what fields are returned is useful for exporting bulk-format records and other scenarios. For example, if a record has properties that correspond to the same data in a number of languages, the application can retrieve only the properties that correspond to the current language. Or, the application may render the record list using tabs to display different sets of data columns (e.g., one tab to view customer details and another to view order details without always returning the data needed to populate both tabs).

This functionality prevents the transferring of unneeded properties and dimension values when they will not be used by the front-end Web application. It therefore makes the application more efficient because the unneeded data does not take up network bandwidth and memory on the application server.

The Select feature can also be used to specifically request fields that are not transferred by default.

Configuring the Select feature

No system configuration is required for the Select feature.

In other words, no instance configuration is required in Developer Studio and no Dgidx or dgraph flags are required to enable selection of properties and dimensions. Any existing property or dimension can be selected.

URL query parameters for Select

There is no Select-specific URL query parameter.

A query for selected fields is the same as any valid navigation query. Therefore, the Navigation parameter (N) is required for the request

Selecting keys in the application

With the Select feature, the Web application can specify which properties and dimensions should be returned for the result record set from the navigation query.

The specific selection method used by the application depends on whether you have a Java or .NET implementation.

Java selection method

Use the `ENEQuery.setSelection()` method for Java implementations.

For Java-based implementations, you specify the selection list by calling the `setSelection()` method of the `ENEQuery` object. Use the following syntax:

```
ENEQuery.setSelection(FieldList selectFields)
```

where *selectFields* is a list of property or dimension names that should be returned with each record. You can populate the `FieldList` object with string names (such as "P_WineType") or with Property or Dimension objects. In the case of objects, the `FieldList.addField()` method automatically extracts the string name from the object and adds it to the `FieldList` object.

During development, you can use the `ENEQuery.getSelection()` method (which returns a `FieldList` object) to check which fields are set.

The `FieldList` object contains a list of Oracle Commerce property and/or dimension names for the query. For details about the methods of the `FieldList` class, see the Oracle Commerce Javadocs for the Presentation API.



Note: The `setSelection()` and `getSelection()` methods are also available in the `UrlENEQuery` class.

Java Select example

The following is a simple Java example of setting an Oracle Commerce property and dimension for a navigation query. When the `ENEQueryResults` object is returned, it will have a list of records that have been tagged with the `P_WineType` property and the `Designation` dimension. You extract the records as with any record query.

```
// Create a query
ENEQuery usq = new UrlENEQuery(request.getQueryString(), "UTF-8");
// Create an empty selection list
FieldList fList = new FieldList();
```

```
// Add an Oracle Commerce property to the list
fList.addField("P_WineType");
// Add an Oracle Commerce dimension to the list
fList.addField("Designation");
// Add the selection list to the query
usq.setSelection(fList);
// Make the MDEX Engine query
ENEQueryResults qr = nec.query(usq);
```

.NET selection property

Use the `ENEQuery.Selection()` property for Java implementations.

In a .NET application, the `ENEQuery.Selection` property is used to get and set the `FieldList` object. You can add properties or dimensions to the `FieldList` object with the `FieldList.AddField` property.



Note: The `Selection` property is also available in the `UrlENEQuery` class.

.NET selection example

The following is a C# example of setting an Oracle Commerce property and dimension for a navigation query.

```
// Create a query
ENEQuery usq = new UrlENEQuery(queryString, "UTF-8");
// Create an empty selection list
FieldList fList = new FieldList();
// Add an Oracle Commerce property to the list
int i = fList.AddField("P_WineType");
// Add an Oracle Commerce dimension to the list
i = fList.AddField("Designation");
// Add the selection list to the query
usq.Selection = fList;
// Make the MDEX Engine query
ENEQueryResults qr = nec.query(usq);
```


Chapter 12

Using the Oracle Commerce Query Language

This section describes how to use the Oracle Commerce Query Language, which allows you to create various types of record filters when making navigation queries for record search.

About the Oracle Commerce Query Language

The Oracle Commerce 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.

Besides record search, these filters can also be used for dimension search. EQL is available as a core feature of Oracle Commerce Guided Search with the capabilities listed in the next section, “Basic filtering capabilities”. In addition, Record Relationship Navigation (RRN) (described in the topic “Record Relationship Navigation module”) is available as an optional module that extends the MDEX Engine capability.

Basic filtering capabilities

You can use EQL to create an expression that can filter on different features.

These include:

- Dimension values
- Specific property values
- A defined range of property values (range filtering)
- A defined range of geocode property values (geospatial filtering)
- Text entered by the user (record search)

The language also supports standard Boolean operators (`and`, `or`, and `not`) to compose complex expressions. In addition, EQL requests can be combined with other Oracle Commerce features, such as spelling auto-correction, Did You Mean suggestions, and the sorting parameters (`Ns` and `Nrk`). details about these interactions are provided in “Oracle Commerce Query Language and other features.”

Record Relationship Navigation module

The Record Relationship Navigation (RRN) module is an optional module that is intended for use with complex relational data.

Only customers entitled to the new module can pose queries that join records at query time and navigate based on the connected relationships.

This module is intended for sites that have different types of records, in which properties in one record type have values that ultimately refer to properties in another record type. For example, an Author record type can have an `author_bookref` property with the ID of a Book record type. In this case, you can leave the records uncombined (when the pipeline is run) and then have the MDEX Engine apply a relationship filter among the record types with an RRN request.

Among the benefits of query-time relationship filters are:

- Reduced memory footprint: With no need to combine different types of records in the pipeline, this will reduce the memory footprint of the MDEX Engine, allowing more data to fit into a single engine.
- Reduced application complexity: With the MDEX Engine handling the data relationships, custom application logic will be greatly simplified.
- Improved performance: RRN improves query performance by removing the need to query the MDEX Engine multiple times, thereby reducing the data being transferred over the network.

For details about constructing these types of requests, see “Record Relationship Navigation queries.”

Oracle Commerce Query Language syntax

The following EBNF grammar describes the syntax for EQL filter expressions.

```
RecordPath ::= Collection "(" ")" "/" RecordStep
Collection ::= FnPrefix? "collection"
FnPrefix ::= "fn" ":"
RecordStep ::= "record" Predicate?
Predicate ::= "[" Expr "]"
Expr ::= OrExpr
OrExpr ::= AndExpr ("or" AndExpr)*
AndExpr ::= NotExpr ("and" NotExpr)*
NotExpr ::= PrimaryExpr | (FnPrefix? "not" "(" Expr ")")
PrimaryExpr ::= ParenExpr | TestExpr
ParenExpr ::= "(" Expr ")"
TestExpr ::= ComparisonExpr | FunctionCall
FunctionCall ::= TrueFunction | FalseFunction | MatchesFunction
TrueFunction ::= FnPrefix? "true" "(" ")"
FalseFunction ::= FnPrefix? "false" "(" ")"
MatchesFunction ::= "endeca" ":" "matches" "(" "." ","
    StringLiteral "," StringLiteral ( "," StringLiteral ( ","
    StringLiteral ( "," (TrueFunction | FalseFunction) )? )? )?
    ")"
ComparisonExpr ::= LiteralComparison | JoinComparison
    | RangeComparison | GeospatialComparison
    | DimensionComparison
EqualityOperator ::= "=" | "!="
LiteralComparison ::= PropertyKey EqualityOperator Literal
JoinComparison ::= PropertyKey "=" PropertyPath
RangeComparison ::= PropertyKey RangeOperator NumericLiteral
GeospatialComparison ::= "endeca" ":" "distance" "("
    PropertyKey "," "endeca" ":" "geocode" "(" NumericLiteral ","
    NumericLiteral ")" ">" | "<" NumericLiteral
DimensionComparison ::= DimensionKey EqualityOperator
    (DimValById | DimValPath) "/" "id"
DimValById ::= "endeca" ":" "dval-by-id" "(" IntegerLiteral ")"
DimValPath ::= Collection "(" "" "dimensions" "" ")"
    ("/" DValStep)*
```



```

DValStep ::= ("*" | "dval") "[" "name" "=" StringLiteral "]"
DimensionKey ::= NCName
PropertyPath ::= RecordPath "/" PropertyKey
PropertyKey ::= NCName
RangeOperator ::= "<" | "<=" | ">" | ">="
Literal ::= NumericLiteral | StringLiteral
NumericLiteral ::= IntegerLiteral | DecimalLiteral
StringLiteral ::= "'" ('"' | [^"])* "'"
IntegerLiteral ::= [0-9]+
DecimalLiteral ::= ([0-9]+ "." [0-9]*) | ("." [0-9]+)

```

The EBNF uses these notations:

- + means 1 or more instances of a component
- ? means 0 or 1 instances
- * means 0 or more instances

The EBNF uses the same Basic EBNF notation as the W3C specification of XML, located at this URL:

<http://www.w3.org/TR/xml/#sec-notation>

Also, note these important items about the syntax:

- Keywords are case sensitive. For example, "endeca:matches" must be specified in lower case, as must the and and or operators.
- The names of keywords are not reserved words across the Oracle Commerce namespace. For example, if you have a property named collection, its name will not conflict with the name of the collection() function.
- To use the double-quote character as a literal character (that is, for inclusion in a string literal), it must be escaped by prepending it with a double-quote character.

These and other aspects of EQL will be discussed further in later sections of this section.

Negation operators

EQL provides two negation operators.

As the EBNF grammar shows, EQL provides two negation operators:

- The **not** operator
- The **!=** operator

An example of the **not** operator is:

```
collection()/record[not(Recordtype = "author")]
```

An example of the **!=** operator is:

```
collection()/record[Recordtype != "author"]
```

Although both operators look like they work the same, each in fact may return a different record set. Using the above two sample queries:

- The **not** operator example returns any record which does not have a Recordtype property with value "author" (including records which have no Recordtype properties at all).
- The **!=** operator returns only records which have non-"author" Recordtype property values. This operator excludes records which have no Recordtype properties.

The small (but noticeable) difference in the result sets may be a useful distinction for your application.

Using negation on properties

EQL supports filtering by the absence of assignments on records. By using the **not** operator, you can filter down to the set of records which do *not* have a specific property assignment.

For example:

```
collection()/record[author_id]
```

returns all records *with* the "author_id" property, while:

```
collection()/record[not (author_id)]
```

returns all records *without* the "author_id" property.

NCName format for properties and dimensions

With a few exceptions (noted when applicable), the names of Oracle Commerce properties and dimensions used in EQL requests must be in an NCName format.

(This restriction does not apply to the names of non-root dimension values or to the names of search interfaces.) The names are also case sensitive when used in EQL requests.

The NCName format is defined in the W3C document Namespaces in XML 1.0 (Second Edition), located at this URL: <http://www.w3.org/TR/REC-xml-names/#NT-NCName>

As defined in the W3C document, an NCName must start with either a letter or an underscore (but keep in mind that the W3C definition of Letter includes many non-Latin characters). If the name has more than one character, it must be followed by any combination of letters, digits, periods, dashes, underscores, combining characters, and extenders. (See the W3C document for definitions of combining characters and extenders.) The NCName cannot have colons or white space.

Take care when creating property names in Developer Studio, because that tool allows you to create names that do not follow the NCName rules. For example, you can create property names that begin with digits and contain colons and white space. Any names which do not comply with NCName formatting will generate a warning when running your pipeline.

The property must also be explicitly enabled for use with record filters (not required for record search queries). Dimension values are automatically enabled for use in record filtering expressions, and therefore do not require any special configuration.

URL query parameters for the Oracle Commerce Query Language

The MDEX Engine URL query parameters listed in this topic are available to control the use of EQL requests.

- **Nrs** - The **Nrs** parameter specifies an EQL request that restricts the results of a navigation query. This parameter links to the Java `ENEQuery.setNavRecordStructureExpr()` method and the `.NET ENEQuery.NavRecordStructureExpr` property. The **Nrs** parameter has a dependency on the **N** parameter, because a navigation query is being performed.
- **Ars** - The **Ars** parameter specifies an EQL request that restricts the results of an aggregated record query. This parameter links to the Java `ENEQuery.setAggrERecStructureExpr()` method and the `.NET ENEQuery.AggrERecStructureExpr` property. The **Ars** parameter has a dependency on the **A** parameter, because an aggregated record query is being performed.
- **Drs** - The **Drs** parameter specifies an EQL request that restricts the set of records considered for a dimension search. Only dimension values represented on at least one record satisfying the filter are returned as search results. This parameter links to the Java

`ENEQuery.setDimSearchNavRecordStructureExpr()` method and the `.NET ENEQuery.DimSearchNavRecordStructureExpr` property. The `Drs` parameter has a dependency on the `D` parameter.

These parameters (including the EQL expression) must be URL-encoded. For example, this query:

```
collection()/record[Recordtype = "author"]
```

should be issued in this URL-encoded format:

```
collection%28%29/record%5BRecordtype%20%3D%20%22author%22%5D
```

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

Making Oracle Commerce Query Language requests

The `collection()` function is used to query the MDEX Engine for a set (that is, a collection) of Oracle Commerce records, based on an expression that defines the records you want.

EQL allows you to make the following types of requests, all of which begin with the `collection()` function:

- Property value query
- Record Relationship Navigation query
- Dimension value query
- Record search query
- Range filter query

The basic syntax for the `collection()` function is:

```
fn:collection()/record[expression]
```

The `fn:` prefix is optional, and for the sake of brevity will not be used in the examples in this chapter. The `/record` step indicates that Oracle Commerce records are being selected. The `expression` argument (which is called the predicate) is an EQL expression that filters the total record set to the subset that you want. The predicate can contain one or more `collection()` functions (multiple functions are nested).

Issuing the `collection()` function without a predicate (that is, without an expression) returns the total record set because the query is not filtering the records. This query is therefore the same as issuing only an `N=0` navigation query, which is a root navigation request.

The following sample query illustrates the use of the `collection()` function with the `Nrs` parameter:

```
controller.jsp?N=0&Nrs=collection()/record[book_id = 8492]
```

Because EQL is a filtering language, it does not have a built-in sorting option. Therefore, an EQL request returns the record set using the MDEX Engine default sort order. You can, however, append a URL sorting parameter, such as the `Ns` parameter or the `Nrk`, `Nrt`, `Nrr`, and `Nrm` set of parameters. For more information about the interaction with other Oracle Commerce features, see “Oracle Commerce Query Language and other features.”

Property value queries

Property value queries (also called literal comparison queries) return those records that have a property whose value on the records is equal to a specified literal value.

The syntax for this type of query is:

```
collection()/record[propertyname = literalValue]
```

where:

- `propertyName` is the NCName of an Oracle Commerce property that is enabled for record filters. Dimension names are not supported for this type of query.
- `literalValue` is a number (either integer or floating point) or a quoted string literal. Numbers are not quoted. For a record to be returned, the value of `literalValue` must exactly match the value of `propertyName`, including the case of the value for quoted string literals. Wildcards are not supported, even if the property has been enabled for wildcard search.

Because it is a predicate, the expression must be enclosed within square brackets. Expressions can be nested.

Note that you can use one of the negation operators described in the "Negation operators" topic.

Examples

The first example illustrates a simple property comparison query:

```
collection()/record[Recordtype = "author"]
```

This query returns all records that have a property named `Recordtype` whose value is "author". If a `Recordtype` property on a record has another value (such as "editor"), then that record is filtered out and not returned.

The second example illustrates how to use the `and` operator:

```
collection()/record[author_nationality = "english"
and author_deceased = "true"]
```

This query returns all `Author` records for English writers who are deceased.

Record Relationship Navigation queries

EQL allows you to issue a request against normalized records, using record-to-record relationship filter expressions. These types of requests are called Record Relationship Navigation (RRN) queries.

If you have different record types in your source data, you can keep the records uncombined by using a Switch join in your pipeline. Then, by issuing an RRN query, the MDEX Engine can apply a relationship filter to the records at query time. Depending on how you have tagged the properties on the records, an RRN query can return records of only one type or of multiple types.

For example, assume that you want to have three record types (`Author` records, `Book` records, and `Editor` records). To define the record type, all the records have a property named `Recordtype` (the actual name does not matter). `Author` records have this property set to "author", `Book` records have it set to "book", and `Editor` records use a value of "editor". In your pipeline, you use a Switch join to leave those records uncombined. You can then filter `Book` records via relationship filters with `Author` and `Editor` records, but the returned records (and their dimension refinements) will be `Book` records only. This means that any other query parameters apply only to the record type that is returned.



Note: You must configure the MDEX Engine in order to enable RRN. This capability is an optional module that extends the MDEX Engine. Oracle Commerce customers who are entitled by their license to use RRN can find instructions on the Oracle Commerce Support site. Contact your Oracle Commerce representative if you need to obtain an RRN license.

Record Relationship Navigation query syntax

This topic describes the syntax for RRN queries.

The basic syntax for an RRN query is:

```
collection()/record[propertyKey1 = recordPath/propertyKey2]
```

where:

- `propertyKey1` is the NCName of an Oracle Commerce property on a record type to be filtered. The resulting records will have this property.
- `recordPath` is one or more `collection()/record` functions.
- `propertyKey2` is the NCName of an Oracle Commerce property on another record type that will be compared to `propertyKey1`. Records that satisfy the comparison will be added to the returned set of records.

The forward slash (/) character is required between `recordPath` and `propertyKey2` because `propertyKey2` is a property step.

There are two ways to differentiate RRN queries from other types of EQL requests:

- RRN queries have a `collection()/record` function on the right side of the comparison operator in the predicate.
- They include a property step.

The following example illustrates a basic relationship filter query:

```
collection()/record[author_bookref =
collection()/record[book_year = "1843"]/book_id]
```

In this example, the `author_bookref` is a property of `Author` records, which means that `Author` records are returned. These records are filtered by the `book_year` and `book_id` properties of the `Book` records. The `author_bookref` property is a reference to the `book_id` property (which is being used as the property step). Therefore, the query returns `Author` records for authors who wrote books that were published in 1843. There is an inner `collection()/record` function (which uses `book_year` as its property key) on the right side of the comparison expression.

The above query example is shown in a linear format. The query can also be made in a structured format, such as the following:

```
collection()/record
[
  author_bookref = collection()/record
  [
    book_year = "1843"
  ]
  /book_id
]
```

This structured format will be used for most of the following examples, as it makes it easier to parse the query.

Relationship filter expressions work from the inside out (that is, the inner expressions are performed before the outer ones). In this example, the MDEX Engine first processes the `Book` records to come up with a set of `Book` records that have the `book_year` property set to "1843". Then the `book_id` property values of the `Book` records provide a list of IDs that are used as indexes to filter the `Author` records (that is, as comparisons to the `author_bookref` property).

Record Relationship Navigation query examples

This topic contains examples of RRN queries.

The following examples assume that you have three record types in your source data: `Author` records, `Book` records, and `Editor` records. While all records have several properties, the `Author` and `Book` records have these properties that establish a relationship between the record types:

- `Author` records have an `author_bookref` property that references the `book_id` property of `Book` records. In addition, `Author` records have an `author_editorref` property that references the `editor_id` property of an `Editor` record.
- `Book` records have a `book_authorref` property that references the `author_id` property of `Author` records.

Using these cross-record reference properties, an RRN query can apply relationship filters between the record types.

RRN relationship filter examples

These examples illustrate how to build relationship filter queries.

The user may first issue a query for `Editor` records for an editor named Jane Smith who works in the city of Boston:

```
collection()/record
[
  editor_name = "Jane Smith"
  and
  editor_city = "Boston"
]
```

The query is then modified for `Author` records:

```
collection()/record
[
  author_editorref = collection()/record
  [
    editor_name = "Jane Smith"
    and
    editor_city = "Boston"
  ]
  /editor_id
]
```

The query returns all `Author` records filtered by the results of the `Editor` records. That is, the `Author` records are filtered by the `editor_id` property of the `Editor` records (which are referenced by the `author_editorref` property in the `Author` records).

The next example query returns books by American authors:

```
collection()/record
[
  book_authorref = collection()/record
  [
    author_nationality = "american"
  ]
  /author_id
]
```

The next example query returns all books by authors who are still alive:

```
collection()/record
[
  book_authorref = collection()/record
  [
    author_is_alive = true
  ]
  /author_id
]
```

```

    author_deceased = "false"
  ]
  /author_id
]

```

The next example query combines the two previous examples, and also illustrates the use of the `or` operator. Both inner `collection()/record` functions use the `author_id` property value results as indexes for the Book records.

```

collection()/record
[
  book_authorref = collection()/record
  [
    author_nationality = "american"
  ]
  /author_id
  or
  book_authorref = collection()/record
  [
    author_deceased = "false"
  ]
  /author_id
]

```

The next example query returns the books written by authors who have had those books published in a hard-cover format.

```

collection()/record
[
  book_authorref=collection()/record
  [
    author_bookref=collection()/record
    [
      book_cover = "hard"
    ]
    /book_id
  ]
  /author_id
]

```

The next query example extends the previous one by returning the books written by authors who have published hard-cover books and have worked with an editor named "Jane Smith". The query also shows how to apply relationship filters among all three record types.

```

collection()/record
[
  book_authorref=collection()/record
  [
    author_bookref=collection()/record
    [
      book_cover="hard"
    ]
    /book_id
    and
    author_editorref=collection()/record
    [
      editor_name="Jane Smith"
    ]
    /editor_id
  ]
  /author_id
]

```

In the final example, this powerful query returns all books written by the author of a play titled "The Island Princess" (which was written by English playwright John Fletcher) and also all books that were written by authors who co-wrote books with Fletcher. The result set will include plays that were written either by Fletcher or by anyone who has ever co-authored a play with Fletcher.

```
collection()/record
[
  book_authorref = collection()/record
  [
    author_bookref = collection()/record
    [
      book_authorref = collection()/record
      [
        author_bookref = collection()/record
        [
          book_title = "The Island Princess"
        ]
      ]/book_id
    ]/author_id
  ]/book_id
]/author_id
]
```

Dimension value queries

Dimension value queries allow you to filter records by dimension values. The dimension value used for filtering can be any dimension value in a flat dimension or in a dimension hierarchy.

Rules for the naming format of the dimension value are as follows:

- For a root dimension value (which has the same name as the dimension), the name must be in the NCName format.
- For a non-root dimension value (such as a leaf), the name does not have to be in the NCName format.

There are two syntaxes for using dimension values to filter records, depending on whether you are specifying a dimension value path or an explicit dimension value node.

Querying with dimension value paths

The syntax described in this topic specifies a dimension value path to the `collection()/record` function.

The path can specify just the root dimension value, or it can traverse part or all of a dimension hierarchy. The query will return all records that are tagged with the specified dimension value and with descendants (if any) of that dimension value.

Use the following steps to construct a dimension value path:

1. The path must start with the NCName of the dimension from which the dimension values will be filtered. The dimension name is not quoted and is case sensitive: `[dimName = collection("dimensions")`
2. It must then be followed by a slash-separated step specifier that uses the `dval` keyword (or the `*` keyword, both are interchangeable) and the name of the root dimension value, which is the same name as the dimension name. The name is case sensitive and must be within double quotes: `/dval[name = "dval-Name"]` or `/*[name = "dvalName"]`.

3. Optionally, you can use one or more slash-separated step specifiers to specify a path to a dimension value descendant. These step specifiers use the same syntax as described in the previous step. Names of descendant dimension values do not have to be in the NCName format.
4. The dimension value path must be terminated with `//id`. The `//id` path terminator specifies that the path be extended to any descendants of the last specified `dvalName` dimension value. The resulting syntax is:
`collection()/record[dimName = collection("dimensions")/dval[name = "dval-Name"]//id`.

Note that you can use one of the negation operators described in the "Negation operators" topic.

Query examples using dimension value paths

The examples in this topic illustrate how to construct dimension value paths using EQL syntax rules.

The examples use the Genre dimension, which has this hierarchy:

```
Genre
  Fiction
    Classic
    Literature
    Science-fiction
  Non-fiction
```

The Fiction dimension value has two descendants (Classic and Science-fiction), while the Non-fiction dimension value has no descendants (that is, it is a leaf dimension value).

The first example query is made against the dimension named Genre (the `dimName` argument). It uses one step specifier for the root dimension value (also named Genre). The query returns all records that are tagged with the Genre dimension value, including all its descendants (such as the Classic dimension value).

```
collection()/record
[
  Genre = collection("dimensions")/dval[name="Genre"]//id
]
```

The next example query uses two step specifiers in the predicate. The dimension value path begins with the dimension name (Genre), followed by the root dimension value name (also Genre), and finally the Fiction child dimension value. The query returns all records that are tagged with the Fiction dimension value, including its three descendants (Classic, Literature, and Science-fiction). Records tagged only with the Non-fiction dimension value are not returned because it is not a descendant of Fiction.

```
collection()/record
[
  Genre = collection("dimensions")/dval[name="Genre"]
    /dval[name="Fiction"]//id
]
```

The next example query uses three step specifiers to drill down to the Classic dimension value, which is a descendant of Fiction. The query returns all records that are tagged with the Classic dimension value or its Literature descendant. The example also shows the use of `*` (instead of `dval`) in the step specifier.

```
collection()/record
[
  Genre =collection("dimensions")/*[name="Genre"]
    /*[name="Fiction"]/*[name="Classic"]//id
]
```

The final example shows how you can use the `or` operator to specify two dimension value paths. The query returns records tagged with either the Science-fiction or Non-fiction dimension values. Using the `and` operator

in place of the `or` operator here would return records tagged with both the Science-fiction and Non-fiction dimension values.

```
collection()/record
[
  Genre = collection("dimensions")/dval[name="Genre"]
    /dval[name="Fiction"]/dval[name="Science-fiction"]//id
  or
  Genre = collection("dimensions")/dval[name="Genre"]
    /dval[name="Non-fiction"]//id
]
```

Querying with dimension value IDs

You can also query dimension value paths using the numerical ID of a dimension value, rather than its name.

In this case, the query returns records that are tagged with this dimension value and all of its descendants (if any). This syntax does not use the "dimensions" argument to the `collection()` function, but it does use the `endeca:dval-by-id()` helper function, as follows:

```
collection()/record[dimName = endeca:dval-by-id(dimValId)//id]
```

where:

- `dimName` is the NCName of the dimension from which the dimension values are filtered. The dimension name is not quoted and is case sensitive.
- `dimValId` is the ID of the dimension value on the records that you want returned. `dimValId` can be any dimension value in the dimension and is not quoted.
- `//id` is the path terminator that specifies that the path be extended to any descendants of `dimValId`.

You can also use the `and` or `or` operators, as shown in the second example below. You can also use one of the negation operators described in the "Negation operators" topic.

Examples

The first query example selects records that are tagged with either the dimension value whose ID is 9 or its descendants.

```
collection()/record
[
  Genre = endeca:dval-by-id(9)//id
]
```

The next query example uses an `or` operator to select records that are tagged with either dimension value 8 (or its descendants) or dimension value 11 (or its descendants).

```
collection()/record
[
  Genre = endeca:dval-by-id(8)//id
  or
  Genre = endeca:dval-by-id(11)//id
]
```

Record search queries

The `endeca:matches()` function allows a user to perform a keyword search against specific properties or dimension values assigned to records. (Record search queries are also called text search queries.)

The resulting records that have matching properties or dimension values are returned, along with any valid refinement dimension values. The search operation returns results that contain text matching all user search terms (that is, the search is conjunctive by default). To perform a less restrictive search, use the `matchMode` argument to specify a match mode other than `MatchAll` mode. Wildcard terms (using the `*` character) can be specified if the search interface or property is configured for wildcards in Developer Studio.


Note the following about record search queries:


- The text search is case insensitive, including phrase search.
- Properties must be enabled for record search (in Developer Studio). Records with properties that are not enabled for record search will not be returned in this type of query.
- For wildcard terms, properties must be enabled for wildcard search.

The syntax for a record search query is:

```
collection()/record[endeca:matches(., "searchKey", "searchTerm", "matchMode",
"languageId", autoPhrase)]
```

The meanings of the arguments are as follows:

Argument	Meaning
.	Required. The period is currently the only valid option. The period is the XPath context item, which is the node currently being considered (that is, the node to apply the function to). In effect, the context item is the record to search.
<code>searchKey</code>	Required. The name of an Oracle Commerce property or search interface which will be evaluated for the search. The name must be specified within a set of double quotes. Property names must use the NCName format and must be enabled for record search. Search interface names do not have to use the NCName format.
<code>searchTerm</code>	<p>Required. The term to search for (which may contain multiple words or phrases). Specify the search term within a pair of double quotes. Phrase searches within <code>searchTerm</code> must be enclosed within two pairs of double quotes in addition to the pair enclosing the entire <code>searchTerm</code> entry. (This is because a pair of double quotes is the XPath escape sequence for a single double quote character within a string literal.)</p> <p>For example, in "Melville ""Moby Dick"" hardcover", the phrase "Moby Dick" is enclosed in two pairs of double quotes: these yield a single escaped pair which indicates a phrase search for these words. In another example, """"Tiny Tim""", the outermost pair of double quotes delimits the full <code>searchTerm</code> value, while the two inner pairs yield a single escaped pair to indicate a phrase search.</p> <p> Note: To enable EQL parsing, use straight double-quote characters for double quotes (rather than typographer's double quotes, which the EQL parser does not accept).</p>
<code>matchMode</code>	Optional. A match mode (also called a search mode) that specifies how restrictive the match should be. The match mode must be specified within a set of double quotes.

Argument	Meaning
	<p>The valid match modes are <code>all</code> (MatchAll mode; perform a conjunctive search by matching all user search terms; this is the default), <code>partial</code> (MatchPartial mode; match some of the search terms), <code>any</code> (MatchAny mode; results need match only a single search term), <code>allpartial</code> (MatchAllPartial mode; first use MatchAll mode and, if no results are returned, then use MatchPartial mode), <code>allany</code> (MatchAllAny mode; first use MatchAll mode and, if no results are returned, then use MatchAny mode), and <code>partialmax</code> (MatchPartialMax mode; first use MatchAll mode and, if no results are returned, then return results matching all but one term, and so on).</p> <p> Note: MatchBoolean is not supported, because EQL has its own powerful set of query composition features such as the <code>and</code>, <code>or</code>, and <code>not</code> operators.</p>
<code>languageId</code>	Optional. A per-query language ID, such as "fr" for French. The ID must be specified within a set of double quotes. For a list of valid language IDs, see the topic "Using language identifiers." The default language ID is the default for the MDEX Engine.
<code>autoPhrase</code>	Optional. A TrueFunction or FalseFunction that sets the option for automatic-phrasing query re-write. The default is <code>false()</code> , which disables automatic phrasing. Specifying <code>true()</code> enables automatic phrasing, which instructs the MDEX Engine to compute a phrasing alternative for a query and then rewrite the query using that phrase. For details about automatic phrasing (including adding phrases to the project with Developer Studio), see the topic "Using automatic phrasing."

Record search query examples

This topic contains examples of record search queries.

The first query example searches for the name *jane* against the `editor_name` property of any record. Because they are not specified, these defaults are used for the other arguments: MatchAll mode, language ID is the MDEX Engine default, and automatic phrasing is disabled.

```
collection()/record
[
  endeca:matches(., "editor_name", "jane")
]
```

The next query example is identical to the first one, except that the wildcard term *ja** is used for the search term. If the `editor_name` property is wildcard-enabled, this search returns records in which the value of the property has a value that begins with *ja* (such as "Jane" or "James").

```
collection()/record
[
  endeca:matches(., "editor_name", "ja*")
]
```

The next query example searches for four individual terms against the "description" property of any records. The `partialmax` argument specifies that the MatchPartialMax match mode be used for the search. The language ID is English (as specified by the "en" argument) and automatic phrasing is disabled (because the default setting is used). Because the MatchPartialMax match mode is specified, MatchAll results are returned

if they exist. If no such results exist, then results matching all but one terms are returned; otherwise, results matching all but two terms are returned; and so forth.

```
collection()/record
[
  endeca:matches(., "description",
    "sailor seafaring ship ocean", "partialmax", "en")
]
```

The next query example illustrates a phrase search. Any phrase term must be within a pair of double quotes, as in the example `""Tiny Tim""`. This is because a pair of double quotes is the XPath escape sequence for a single double-quote character within a string literal. Thus, if the entire search term is a single phrase, there are three sets of quotes, as in the example.

```
collection()/record
[
  endeca:matches(., "description", """"Tiny Tim""")
]
```

In the final query example, the use of the `true()` function enables the automatic phrasing option. This example assumes that phrases have been added to the project with Developer Studio or Oracle Commerce Workbench. The example also illustrates the use of the `MatchAll` match mode.

```
collection()/record
[
  endeca:matches(., "description", "story of", "all", "en", true())
]
```

Range filter queries

The EQL range filter functionality allows a user, at request time, to specify either a literal value or a geocode value to limit the records returned for the query.

The remaining refinement dimension values for the records in the result set are also returned. The literal value expressions are called basic range queries and the geocode value expressions are geospatial range queries.



Note: Do not confuse EQL range filters with the range filters implemented by the `Nf` parameter. Although both types of range filters are similar in nature, EQL range filters are implemented differently, as described below.

Supported property types for range filters

EQL range filters can be applied only to Oracle Commerce properties of certain types.

The following types are supported:

- Integer (for basic range filters)
- Floating point (for basic range filters)
- DateTime (for basic range filters)
- Geocode (for geospatial range filters)

No special configuration is required for these properties. However, the property name must follow the NCName format. No Dgidx flags are necessary to enable range filters, as the range filter computational work is done at request-time. Likewise, no dgraph flags are needed to enable EQL range filters.

Basic range filter syntax

This topic describes the syntax for EQL range filters.

The syntax for a basic range filter query is:

```
collection()/record[propName rangeOp numLiteral]
```

where:

- `propName` is the name (in an NCName format) of an Oracle Commerce property of type Integer or Floating point.
- `rangeOp` is a range (relational) operator from the table below.
- `numLiteral` is a numerical literal value used for the comparison by the range operator.

The property value of `propName` must be numeric in order that a successful comparison be made against the `numLiteral` argument. The supported range operators are the following:

Operator	Meaning
<	Less than. The value of the property is less than the numeric literal.
<=	Less than or equal to. The value of the property is less than the numeric literal or equal to the numerical literal.
>	Greater than or equal to. The value of the property is greater than the numeric literal or equal to the numerical literal.
>=	Greater than. The value of the property is greater than the numeric literal.

Range filter query examples

This topic contains examples of basic range filter queries.

The first query example uses the `>` operator to return any record that has an `author_id` property whose value is greater than 100.

```
collection()/record
[
  author_id > 100
]
```

The next query example uses the `>=` operator to return Book records whose `book_id` property value is less than or equal to 99. The example also shows the use of the `and` operator.

```
collection()/record
[
  Recordtype = "book"
  and
  book_id <= 99
]
```

The last query example shows an RRN query that uses a range filter expression in its predicate. Based on a relationship filter applied to the Book and Author records, the query returns Book records (which have the

book_authorref property) of authors whose books have been edited by an editor whose ID is less than or equal to 12.

```
collection()/record
[
  book_authorref = collection()/record
  [
    author_editorref <= 12
  ]
  /author_id
]
```

Geospatial range filter syntax

Geospatial range filter queries will filter records based on the distance of a geocode property from a given reference point.

The reference point is a latitude/longitude pair of floating-point values that are arguments to the `endeca:geocode()` function. These queries are triggered by the `endeca:distance()` function, which in turn uses the `endeca:geocode()` function as one of two arguments in its predicate. The syntax for a geospatial range query is:

```
collection()/record[endeca:distance(geoPropName,
endeca:geocode(latValue,lonValue)) rangeOp distLimit]
```

where:

- `geoPropName` is the name (in NCName format) of an Oracle Commerce geocode property.
- `latValue` is the latitude of the location in either an integer or a floating point value. Positive values indicate north latitude and negative values indicate south latitude.
- `lonValue` is the longitude of the location either an integer or a floating point value. Positive values indicate east longitude and negative values indicate west longitude.
- `rangeOp` is either the `<` (less than) or `>` (greater than) operator. These range operators specify that the distance from the geocode property to the reference point is either less (`<`) or greater (`>`) than the given distance limit (the `distLimit` argument).
- `distLimit` is a numerical literal value used for the comparison by the range operator. Distance limits are always expressed in kilometers.

When the geospatial filter query is made, the records are filtered by the distance from the geocode property to the geocode reference point (the latitude/longitude pair of values).

For example, Oracle Commerce's main office is located at 42.365615 north latitude, 71.075647 west longitude. Assuming a geocode property named `Location`, a geospatial filter query would look like this:

```
collection()/record
[
  endeca:distance(Location,
    endeca:geocode(42.365615,-71.075647)) < 10
]
```

The query returns only those records whose location (as specified in the `Location` property) is less than 10 kilometers from Oracle Commerce's main office.

Dimension search queries

The `DRS` URL query parameter sets an EQL 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 filter are returned as search results.

Note the following about the `Drs` parameter:

- The syntax of `Drs` is identical to that of the `Nrs` parameter.
- `Drs` is dependent on the `D` parameter.

Because the `Drs` syntax is identical to that of `Nrs`, you can use the various EQL requests that are documented earlier in this section.

The following example illustrates a dimension search query using an EQL filter:

```
N=0&D=novel&Drs=collection()/record[author_deceased = "false"]
```

The query uses the `D` parameter to specify *novel* as the search term, while the `Drs` parameter sets a filter for records in which the `author_deceased` property is set to false (that is, records of deceased authors).

Oracle Commerce Query Language interaction with other features

Because EQL is a filtering language, it does not contain functionality to perform actions such as triggering Content Spotlighting, sorting, or relevance ranking.

However, EQL is compatible with other query parameters to provide these features for queries. A brief summary of these interactions is:

- `Nrs` is freely composable with the `N`, `Ntt`, `Nr`, and `Nf` filtering parameters. EQL filtering can be conceptualized as occurring after record filtering in terms of side-effects such as spelling auto-correction. This means that a record search within EQL, using the `endeca:matches()` function, cannot auto-correct to a spelling suggestion outside of the record filter.
- Ordering and relevance ranking parameters (`Ns`, `Nrk`, `Nrt`, `Nrr`, `Nrm`) are composable with EQL filters or other types of filters. The `Nrk`, `Nrt`, `Nrr`, and `Nrm` relevance ranking parameters take precedence over a relevance ranking declaration with the `Ntt` and `Ntx` parameters.

The following table provides an overview of these interactions. The sections after the table provide more information.

Parameter	Similar function in EQL?	Why use this parameter rather than <code>Nrs</code> ?	Parameter interaction
<code>N</code>	Yes. Dimension filtering can be done in EQL.	Use <code>N</code> to trigger Content Spotlighting and refinement generation.	The results of <code>Nrs</code> are intersected with the results of <code>N</code> .
<code>Nr</code>	Yes. EQL can filter on properties or dimensions.	Use <code>Nr</code> for security reasons or to explicitly exclude certain records from being considered in the rest of the query (e.g., for spelling suggestions).	<code>Nr</code> is a pre-filter. Only the records that pass this filter are even considered in <code>Nrs</code> .

Parameter	Similar function in EQL?	Why use this parameter rather than Nrs?	Parameter interaction
Ntt, Ntk	Yes. EQL provides the ability to do record search.	Use Ntt/Ntk to trigger Content Spotighting, as record search within Nrs does not trigger it. Use Ntt/Ntk with Nty for DYM spelling suggestions. (Nrs record search does support autocorrection, but not DYM.)	Similar to N, the results of Nrs are intersected with the results of Ntt/Ntk.
Nf	Yes. EQL provides the ability to do range filtering.	No reason to do so. EQL actually provides greater flexibility because range filters within Nrs can be OR'ed, whereas Nf range filters cannot. Similar to N, the results of Nrs are intersected with the results of Nf.	Similar to N, the results of Nrs are intersected with the results of Nf.
Ns	No. EQL does not have the ability to sort results.	N/A	As long as the property specified in Ns exists on the records being returned, the Ns parameter will sort the results.
Ne	No. EQL does not have the ability to expose dimensions.	N/A	As long as the dimensions specified in Ne exist on the records being returned, the Ne parameter will expose those dimensions.
Nrk, Nrt, Nrr, Nrm	No. EQL does not provide the ability to relevance rank the results.	N/A	This set of parameters allow the ability to apply relevance ranking to results even if record search does not exist.
Nrc	No. EQL does not provide the ability to modify refinement configuration.	N/A	The Nrc parameter lets you modify refinement configuration at query time (for dynamic ranking, statistics, and so on).

N parameter interaction

The `Nrs` parameter has a dependency on the `N` parameter.

This means that you must use `N=0` if no navigation filter is intended. Note, however, that the presence of the `N` parameter does not affect `Nrs` (for example, for such actions as spelling correction and automatic phrasing).

If the `N` parameter is used with one or more dimension value IDs, it can trigger Content Spotlighting, since dimension filtering within `Nrs` does not trigger it. The resulting record set will be an intersection of both filters. In this case, the dimension value IDs specified by the `N` parameter must belong to dimensions that exist for the records being returned by `Nrs`.

For example, if the `N` parameter is filtering on Author Location but the `Nrs` parameter is returning only Book records, then this intersection will result in zero records. In addition, during a query the `Nrs` parameter does not trigger refinement generation for multi-select and hierarchical dimensions, while `N` does. Therefore, because `Nrs` is ignored for purposes of refinement generation while `N` plays a key role, the `N` parameter should be used instead of `Nrs` for parts of the query.

Nr record filter interactions

The `Nr` parameter sets a record filter for a navigation query.

When used with an EQL request, the `Nr` parameter acts as a prefilter. That is, it restricts the set of records that are visible to the `Nrs` parameter. Because it is a prefilter, the `Nr` parameter is especially useful as a security filter to control the records that a user can see. It is also useful to explicitly exclude certain records from being considered in the rest of the query (for example, for spelling suggestions).

When using the `Nr` parameter, keep in mind that only the records that pass the `Nr` filter are even considered in `Nrs`. For example, if you have Book records and Author records, both of these record types would have to pass the `Nr` record filter logic in order for the `Nrs` parameter to determine relationships between Books and Authors.

Nf range filter interactions

The `Nf` parameter enables range filter functionality.

Unlike a record filter, the `Nf` parameter does not act as a prefilter. Instead, when used with the `Nrs` parameter, the resulting record set will be an intersection of the results of the `Nf` and `Nrs` parameters. That is, an `Nf` range filter and an EQL filter together form an AND Boolean request. For more information about range filters, see “Using Range Filters.”

Ntk and Ntt record search interaction

`Ntk` and `Ntt` are a set of parameters used for record search which act as a filter (not a prefilter).

Therefore, when used with the `Nrs` parameter, the resulting record set will be an intersection of the results of the `Nrs` parameter and the `Ntk/Ntt` parameters. There are two main advantages of using these parameters with the `Nrs` parameter:

- The `Ntk/Ntt` parameters can trigger Content Spotlighting, whereas the `Nrs` parameter cannot.
- The `Ntk/Ntt` parameters can return auxiliary information (such as DYM spelling suggestions and supplemental objects), whereas the `Nrs` parameter cannot.

In addition, you can use other parameters that depend on `Ntk`, such as the `Ntx` parameter to specify a match mode or a relevance ranking strategy.

Ns sorting interaction

You can append the `Ns` parameter to an EQL request to sort the returned record set by a property of the records.

To do so, use the following syntax:

```
Ns=sort-key-name[ | order]
```

The `Ns` parameter specifies the property or dimension by which to sort the records, and an optional list of directions in which to sort. For example, this query:

```
Nrs=collection()/records[book_authorref = collection()  
/records[author_nationality = "american"]  
/author_id]&Ns=book_year
```

returns all books written by American authors and sorts the records by the year in which the book was written (the `book_year` property). You can also add the optional order parameter to `Ns` to control the order in which the property is sorted (0 is for an ascending sort while 1 is descending). The default sort order for a property is ascending. For example, the above query returns the records in ascending order (from the earliest year to the latest), while the following `Ns` syntax uses a descending sort order:

```
Ns=book_year|1
```

Nrk relevance ranking interaction

The `Nrk`, `Nrt`, `Nrr`, and `Nrm` set of parameters can be used to order the records of an EQL request, via a specified relevance ranking strategy.

The following is an example of using these parameters:

```
Nrs=collection()/record[Recordtype = "book"]  
&Nrk=All&Nrt=novel&Nrr=maxfield&Nrm=matchall
```

The sample query returns all Book records (that is, all records that are tagged with the `Recordtype` property set to "book"). The record set is ordered with the `Maxfield` relevance ranking module (specified via `Nrr`), which uses the word *novel* (specified via `Nrt`). The search interface is specified via the `Nrk` parameter.

The `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters take precedence over the `Ntk`, `Ntt`, and `Ntx` parameters. That is, if both sets of parameters are used in a query, the relevance ranking strategy specified by the `Nrr` parameter will be used to order the records. For more information about these parameters, see the topic "Using the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters."

Ne exposed refinements interaction

The `Ne` parameter specifies which dimension (out of all valid dimensions returned in an EQL request) should return actual refinement dimension values.

The behavior of the `Ne` parameter is the same for EQL request as for other types of navigation queries.

The following example shows the `Ne` parameter being specified with an EQL text search:

```
Nrs=collection()/record[endeca:matches(.,"description",  
"story","partialmax"]]&Ne=6
```

In the query, 6 is the root dimension value ID for the Genre dimension. The query will return all dimensions for records in which the term *story* appears in the description property, as well as the refinement dimension values for the Genre dimension.

Spelling auto-correction and Did You Mean interaction

Spelling auto-correction for dimension search and record search automatically computes alternate spellings for user query terms that are misspelled.

The Did You Mean (DYM) feature provides the user with explicit alternative suggestions for a keyword search. Both features are fully explained in the "Implementing Spelling Correction and Did You Mean" section.

Both DYM and spelling auto-correction work normally when the `Ntt` parameter is used with `Nrs`. For example, in the following query:

```
Nrs=collection()/record[Recordtype = "book"]
&Ntk=description&Ntt=storye&Ntx=mode+matchall
```

the misspelled term *storye* is auto-corrected to *story* (assuming that the MDEX Engine was started with the `--spl` flag).

If DYM is enabled instead of auto-correction (using the `--dym` flag), then the `Nty=1` parameter can be used in the query:

```
Nrs=collection()/record[Recordtype = "book"]
&Ntk=description&Ntt=storye&Ntx=mode+matchall&Nty=1
```

In this case, no records are returned (assuming that the misspelled term *storye* is not in the data set), but the term *story* is returned as a DYM suggestion.

If both spelling auto-correction and DYM are enabled, the spelling auto-correction feature will take precedence. However, for a full text search with the `endeca:matches()` function, the spelling auto-correction feature will work, but the DYM feature is not supported. For example, in this query:

```
collection()/record
[
  endeca:matches(., "description", "storye")
]
```

the misspelled term *storye* is auto-corrected to *story* if auto-correction is enabled. If DYM is enabled but auto-correction is not, then no records are returned (again assuming that the misspelled term *storye* is not in the data set).

Oracle Commerce Analytics interaction

The Oracle Commerce Analytics API can be used to request analytics operations based on the results of a navigation query.

The analytics operations will work the same way as with navigation queries that do not use EQL.

Oracle Commerce Query Language per-query statistics log

The MDEX Engine can log information about the processing time of an EQL request.

The log entry is at the level of a time breakdown across the stages of query processing (including relationship filters). This information will help you to identify and tune time-consuming queries.



Note: Only EQL requests produce statistics for this log. Therefore, you should enable this log only if you are using EQL.

Implementing the per-query statistics log

The EQL per-query statistics log is turned off by default.

You can specify its creation by using the `dgraph --log_stats` flag:

```
--log_stats path
```

The path argument sets the path and filename for the log.

This argument must be a filename, not a directory. If the file cannot be opened, no logging will be performed. The log file uses an XML format, as shown in the following example that shows a log entry for this simple query:

```
fn:collection()/record[author_nationality = "english"]
```

To read the file, you can open it with a text editor, such as TextPad.

```
<?xml version="1.0" encoding="UTF-8"?>
<Queries>
<Query xmlns="endeca:stats">

  <EndecaQueryLanguage>
    <Stats>
      <RecordPath query_string="fn:collection()/record[author_nationality =
&quot;english&quot;]">
        <StatInfo number_of_records="2">
          <TimeInfo>
            <Descendant unit="ms">0.47705078125</Descendant>
            <Self unit="ms">0.194580078125</Self>
            <Total unit="ms">0.671630859375</Total>
          </TimeInfo>
        </StatInfo>
        <Predicate query_string="[author_nationality = &quot;english&quot;]">
          <StatInfo number_of_records="2">
            <TimeInfo>
              <Descendant unit="ms">0.287841796875</Descendant>
              <Self unit="ms">0.189208984375</Self>
              <Total unit="ms">0.47705078125</Total>
            </TimeInfo>
          </StatInfo>
          <PropertyComparison query_string="author_nationality = &quot;en-
glish&quot;">
            <StatInfo number_of_records="2">
              <TimeInfo>
                <Descendant unit="ms">0.001953125</Descendant>
                <Self unit="ms">0.285888671875</Self>
                <Total unit="ms">0.287841796875</Total>
              </TimeInfo>
            </StatInfo>
            <StringLiteral query_string="&quot;english&quot;">
              <StatInfo number_of_records="0">
                <TimeInfo>
                  <Descendant unit="ms">0</Descendant>
                  <Self unit="ms">0.001953125</Self>
                  <Total unit="ms">0.001953125</Total>
                </TimeInfo>
              </StatInfo>
            </StringLiteral>
          </PropertyComparison>
        </Predicate>
      </RecordPath>
    </Stats>
```

```

    </EndecaQueryLanguage>
  </Query>
</Queries>

```

Parts of the log file

The following table describes the meanings of the elements and attributes.

Element/Attribute	Description
Query	Encapsulates the statistics for a given query (that is, each query will have its own <code>Query</code> node).
RecordPath	The record path of a <code>collection()</code> function.
Predicate	Lists the time spent processing the predicate part of a query.
otherNodes	Lists the time spent processing an expression part of an query, such as <code>PropertyComparison</code> (property or range filter query), <code>StringLiteral</code> (property value query), <code>MatchesExpr</code> (text search query), and <code>DValueComparison</code> (dimension value query).
query_string	For <code>RecordPath</code> , this attribute lists the full query that was issued. For the other elements, it lists the part of the query for which statistics in that element are given.
number_of_records	Returns the number of records which satisfy the <code>query_string</code> in a given node.
StatInfo	Encapsulates the <code>TimeInfo</code> and <code>CacheInfo</code> information.
TimeInfo	Encapsulates time-related information about the node.
Descendant	The time, in milliseconds, spent in the descendants of a given node.
Self	The total amount of time, in milliseconds, spent in this node.
Total	The total amount of time, in milliseconds, spent in this node and its descendants.
CacheInfo	Encapsulates information about cache hits, misses, and insertions. Cache is checked only when a combined relationship filter and range comparison is made.

Setting the logging threshold for queries

You can set the threshold above which statistics information for a query will be logged.

You do this by using the `dgraph --log-stats-thresh` flag. Note that this flag is dependent on the `--log_stats` flag.

The syntax of the threshold flag is:

```
--log_stats_thresh value
```

The value argument is in milliseconds (1000 milliseconds = 1 second). However, the value can be specified in seconds by adding a trailing s to the number.

For example, this:

```
--log_stats_thresh 1s
```

is the same as:

```
--log_stat_thresh 1000
```

If the total execution time for an Oracle Commerce Query Language request (not the expression execution time) is above this threshold, per-query performance information will be logged. The default for the threshold is 1 minute (60000 milliseconds). That is, if you use the `--log_stats` flag but not the `--log_stats_thresh` flag, a value of 1 minute will be used as the threshold for the queries.

Creating an Oracle Commerce Query Language pipeline

This section provides information about configuring the pipeline for an application that implements EQL.

Also included are requirements for the Oracle Commerce properties and dimensions.

Creating the dimensions and properties

Before an Oracle Commerce property can be used in EQL requests, the property must be configured appropriately.

The details are as follows:

- One or more of the following must be true of the property:
 - It is explicitly enabled for use with record filters.
 - It is specified as a rollup key.
 - It is specified as a record spec.
 - It has one of the following types: double, integer, geocode, datetime, duration, or time.
- The property name must be in the NCName format, as explained in the topic “NCName format for properties and dimensions.”
- If you want to allow wildcard terms for record searches, the property must be enabled for wildcard search.

To enable a property for record filters, open the property in the Developer Studio Property editor and check the “Enable for record filters” attribute.

Use the Property editor’s Search tab to configure the property for record search and wildcard search. To use dimensions in Oracle Commerce Query Language queries:

- All dimensions are automatically enabled for use in EQL record filter expressions, and therefore do not need to be enabled for record filters.
- Dimension names (and therefore the names of root dimension values) must be in the NCName format. Names of non-root dimension values, however, do not have to be in the NCName format.

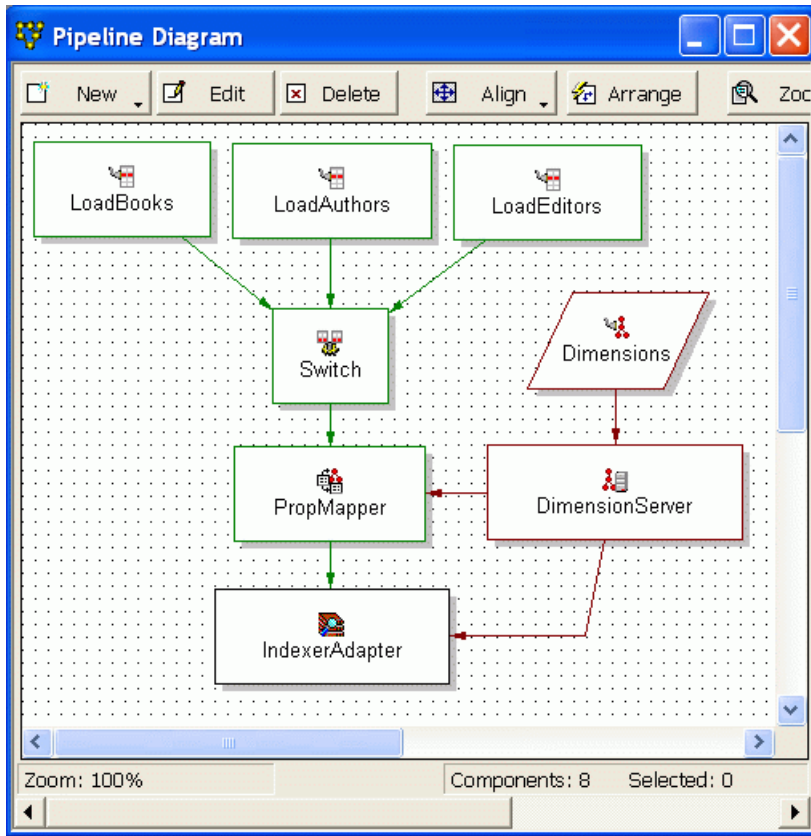
If you are using a search interface with EQL requests, the name of the search interface does not have to be an NCName.

Configuring the pipeline for Switch joins

With one exception, the pipeline used for an application that implements EQL does not have any special configuration requirements.

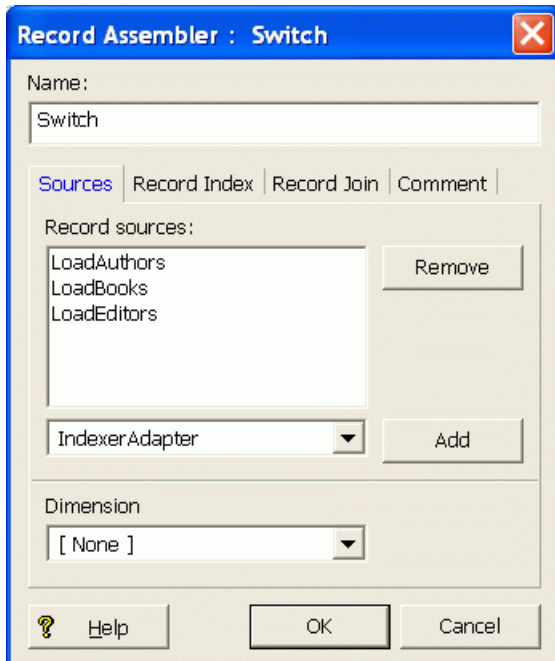
The exception is if you purchased the RRN module and will be using it to apply relationship filters at query time. In this case, you will probably be using a Switch join in the pipeline. Note that using a Switch join is not mandatory for RRN queries, but you will use one if you want to keep different record types uncombined.

For example, the pipeline used in the application that provides the sample queries (for other sections of this chapter) assumes that the data set has three types of records. The pipeline looks like this in Developer Studio's Pipeline Diagram:



The pipeline has three record adapters to load the three record types (Book records, Author records, and Editor records). These are standard record adapters and do not require any special configuration.

The record assembler (named Switch) is used to implement a Switch join on the three sets of records. The Sources tab is where you add the record sources for the record assembler, which are the three record adapters:



The record assembler will process all the records from the three record adapters. However, the records are never compared or combined. Because the three record types are not combined, you can use RRN queries to apply relationship filters. For more information about these types of queries, see the topic "Record Relationship Navigation queries."

Running the Oracle Commerce Query Language pipeline

No special configuration is needed for running an EQL pipeline.

You can run the pipeline with either the Oracle Commerce Application Controller (EAC) or control scripts. See the *Platform Services Application Controller Guide* for details on provisioning your application. For information about using control scripts, see the *Platform Services Control System Guide*.

Chapter 13

Record Filters

This section describes how to implement record filters in your Oracle Commerce application.

About record filters

Record filters allow an Oracle Commerce application to define arbitrary subsets of the total record set and dynamically restrict search and navigation results to these subsets.

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

If you specify a record filter, whether for security, custom catalogs, or any other reason, it is applied before any search processing. The result is that the search query is performed as if the data set only contained records allowed by the record filter.

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

Filter expressions can be saved and loaded from XML files, or passed directly as part of an MDEX Engine query. In either case, when a filter is selected, the set of visible records is restricted to those matching the filter expression. For example, record search queries will not return records outside the selected subset, and refinement dimension values are restricted to lead only to records contained within the subset.

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

Record filter syntax

Record filters are specified with query-based or file-based expressions.

Record filters can be specified directly within an MDEX Engine query. For example, the complete Boolean expression representing the desired record subset can be passed directly in an application URL.

In some cases, however, filter expressions require persistence (in the case where the application allows the end user to define and save custom part lists) or may grow too large to be passed conveniently as part of the query (in the case where a filter list containing thousands of part numbers). To handle cases such as these, the MDEX Engine also supports file-based filter expressions.

File-based filter expressions are simply files stored in a defined location containing XML representations of filter expressions. This section describes both the MDEX Engine query and XML syntaxes for filter expressions.

Query-level syntax

The query-level syntax supports prefix-oriented Boolean functions (AND, OR, and NOT), colon-separated paths for dimension values and property values, and numeric dimension value IDs.

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

```

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

```

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

```
( ) , : \
```

Using the FILTER operator

Aside from nested Boolean operations, a key aspect of query filter expressions is the ability to refer to file-based filter expressions using the FILTER operator. For example, if a filter is stored in a file called MyFilter, that filter can be selected as follows:

```
FILTER(MyFilter)
```

FILTER operators can be combined with normal Boolean operators to compose filter operations, as in this example:

```
AND(FILTER(MyFilter),NOT(Manufacturer:Sony))
```

The expression selects records that are satisfied by the expression contained in the file MyFilter but that are not assigned the value Sony to the Manufacturer property.

Example of a query-level filter expression

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

```
OR(AND(Manufacturer:Sony,1001),
   AND(Manufacturer:Aiwa,NOT(1002)), Manufacturer:Denon)
```

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

- Value for the Manufacturer property is *Sony* and record assigned dimension value is 1001.
- Value for Manufacturer is *Aiwa* and record is not assigned dimension value 1002.
- Value for Manufacturer property is *Denon*.

XML syntax for file-based record filter expressions

The syntax for file-based record filter expressions closely mirrors the query level syntax, with some differences.

The file-based differences from the query-level syntax are:

- In place of the AND, OR, NOT, and FILTER operators, the FILTER_AND, FILTER_OR, FILTER_NOT, and FILTER_NAME XML elements are used, respectively.
- In place of the property and dimension value syntax used for query expressions, the PROP, DVAL_ID, and DVAL_PATH elements are used. Note that the DVAL_PATH element's PATH attribute requires that paths for dimension values and property values be separated by colons, not forward slashes.
- Instead of parentheses to enclose operand lists, normal XML element nesting (implicit in the locations of element start and end tags) is used.

The full DTD for XML file-based record filter expressions is provided in the `filter.dtd` file packaged with the Oracle Commerce software release.

Examples of file-based filter expressions

As an example, the following query-level expression:

```
OR(AND(Manufacturer:Sony,1001),
   AND(Manufacturer:Aiwa,NOT(1002)), Manufacturer:Denon)
```

is represented as a file-based expression using the following XML syntax:

```
<FILTER>
  <FILTER_OR>
    <FILTER_AND>
      <PROP NAME="Manufacturer"><PVAL>Sony</PVAL></PROP>
      <DVAL_ID ID="1001"/>
    </FILTER_AND>
    <FILTER_AND>
      <PROP NAME="Manufacturer"><PVAL>Aiwa</PVAL></PROP>
      <FILTER_NOT>
        <DVAL_ID ID="1002"/>
      </FILTER_NOT>
    </FILTER_AND>
    <PROP NAME="Manufacturer"><PVAL>Denon</PVAL></PROP>
  </FILTER_OR>
</FILTER>
```

Just as file-based expressions can be composed with query expressions, file expressions can also be composed within other file expressions. For example, the following query expression:

```
AND(FILTER(MyFilter),NOT(Manufacturer:Sony))
```

can be represented as a file-based expression using the following XML:

```
<FILTER>
  <FILTER_AND>
    <FILTER_NAME NAME="MyFilter" />
    <FILTER_NOT>
      <PROP NAME="Manufacturer"><PVAL>Sony</PVAL></PROP>
    </FILTER_NOT>
  </FILTER_AND>
</FILTER>
```

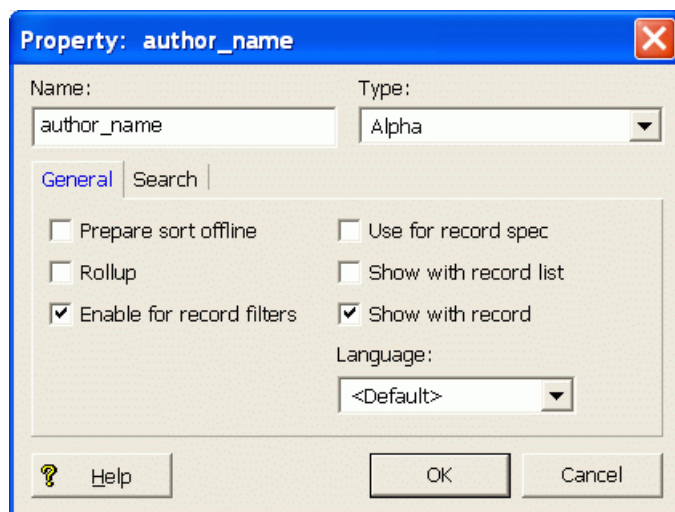
Enabling properties for use in record filters

Oracle Commerce Properties must be explicitly enabled for use in record filters.

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

To enable a property for use with record filters:

1. In Developer Studio, open the Properties view.
2. Double-click on the Oracle Commerce 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.

Data configuration for file-based filters

To use file-based filter expressions in an application, you must create a directory to contain record filter files in the same location where the MDEX Engine index data will reside.

The name of this directory must be:

```
<index_prefix>.fcl
```

For example, if the MDEX Engine index data resides in the directory:

```
/usr/local/endeca/my_app/data/partition0/dgidx_output/
```

and the index data prefix is:

```
/usr/local/endeca/my_app/data/partition0/dgidx_output/index
```

then the directory created to contain record filter files must be:

```
/usr/local/endeca/my_app/data/partition0/dgidx_output/index.fcl
```

Record filters that are needed by the application should be stored in this directory, which is searched automatically when record filters are selected in an MDEX Engine query. For example, if in the above case you create a filter file with the path:

```
/usr/local/endeca/my_app/data/partition0/dgidx_output/index.fcl/MyFilter
```

then the filter expression stored in this file will be used when the query refers to the filter `MyFilter`.

For example, the URL query:

```
N=0&Nr=FILTER(MyFilter)
```

will use this file filter.

Record filter result caching

The MDEX Engine caches the results of file-based record filter evaluations for re-use.

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



Note: The MDEX Engine only caches the results of file-based record filters, because these are generally more costly to evaluate due to XML-parsing overhead.

URL query parameters for record filters

Three MDEX Engine URL query parameters are available to control the use of record filters.

The URL query parameters are as follows:

Parameter	Description
Nr	Links to the Java <code>ENEQuery.setNavRecordFilter()</code> method and the .NET <code>ENEQuery.NavRecordFilter</code> property. The <code>Nr</code> parameter can be used to specify a record filter expression that will restrict the results of a navigation query.
Ar	Links to the Java <code>ENEQuery.setAggrERecNavRecordFilter()</code> method and the .NET <code>ENEQuery.AggrERecNavRecordFilter</code> property. The <code>Ar</code> parameter can be used to specify a record filter expression that will restrict the records contained in an aggregated-record result returned by the MDEX Engine.
Dr	Links to the Java <code>ENEQuery.setDimSearchNavRecordFilter()</code> method and the .NET <code>ENEQuery.DimSearchNavRecordFilter</code> property. The <code>Dr</code> parameter can be used to specify a record filter expression that will restrict the universe of records considered for a dimension search. Only dimension values represented on at least one record satisfying the specified filter will be returned as search results.

Using the Nr query parameter

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

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

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

```
Nr=OR( sku:123 , OR( sku:456 ) , OR( sku:789 ) )
```

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

```
Nr=sku:123
```

Examples of record filter query parameters

```
<application>?N=0&Nr=FILTER(MyFilter)
<application>?A=2496&An=0&Ar=OR(10001,20099)
<application>?D=Hawaii&Dn=0&Dr=NOT(Subject:Travel)
```

Record filter performance impact

Record filters can have an impact in some areas.

The evaluation of record filter expressions is based on the same indexing technology that supports navigation queries in the MDEX Engine. Because of this, there is no additional memory or indexing cost associated with using navigation dimension values in record filters.

Because expression evaluation is based on composition of indexed information, most expressions of moderate size (that is, tens of terms/operators) do not add significantly to request processing time.

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

Record filters can impact the following areas:

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

Interaction with spelling auto-correction and spelling DYM

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

Memory cost

The use of properties in record filters incurs a memory cost.

The evaluation of record filter dimension value expressions is based on the same indexing technology that supports navigation queries in the dgraph. Because of this, there is no additional memory or indexing cost associated with using navigation dimension values in record filters. When using property values in record filter expressions, additional memory and indexing cost is incurred because properties are not normally indexed for navigation.

This feature is controlled in Developer Studio by the **Enable for record filters** setting on the Property editor.

Expression evaluation

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

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

Large OR filters

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

The total cost of processing records can be broken down into two main parts: the parsing cost and the evaluation cost. For large expressions such as these, which will commonly be stored as file-based filters, XML parsing performance dominates total processing cost.

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

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

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

Large-scale negation

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

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

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

Chapter 14

Bulk Export of Records

This section describes the bulk export feature.

About the bulk export feature

The bulk export feature allows your application to perform a navigation query for a large number of records.

Each record in the resulting record set is returned from the MDEX Engine in a bulk-export-ready, gzipped format. The records can then be exported to external tools, such as a Microsoft Excel or a CSV (comma separated value) file.

Applications are typically limited in the number of records that can be requested by the memory requirements of the front-end application server. The bulk export feature adds a means of delaying parsing and `ERec` or `AggrERec` object instantiation, which allows front-end applications to handle requests for large numbers of records.

Configuring the bulk export feature

Oracle Commerce properties and dimensions must be configured for bulk export.

Oracle Commerce properties and/or dimensions that will be included in a result set for bulk exporting must be configured in Developer Studio with the **Show with Record List** checkbox enabled. When this checkbox is set, the property or dimension will appear in the record list display.

No `Dgidx` or `dgraph` flags are necessary to enable the bulk exporting of records. Any property or dimension that has the **Show with Record List** attribute is available to be exported.

Using URL query parameters for bulk export

A query for bulk export records is the same as any valid navigation query.

Therefore, the Navigation parameter (`N`) is required for the request. No other URL query parameters are mandatory.

Setting the number of bulk records to return

By using members from the `ENEQuery` class, you can set the number of bulk-format records to be returned by the MDEX Engine.

When creating the navigation query, the application can specify the number of Oracle Commerce records or aggregated records that should be returned in a bulk format with these Java and .NET calls:

- The Java `ENEQuery.setNavNumBulkERecs()` method and the .NET `ENEQuery.NavNumBulkERecs` property set the maximum number of Oracle Commerce records (`ERec` objects) to be returned in a bulk format from a navigation query.
- The Java `ENEQuery.setNavNumBulkAggrERecs()` method and the .NET `ENEQuery.NavNumBulkAggrERecs` property set the maximum number of aggregated Oracle Commerce records (`AggrERec` objects) to be returned in bulk format from a navigation query.

The `MAX_BULK_ERECES_AVAILABLE` constant can be used with either call to specify that all of the records that match the query should be exported; for example:

```
// Java example:
usq.setNavNumBulkERecs(MAX_BULK_ERECES_AVAILABLE);

// .NET example:
usq.NavNumBulkERecs = MAX_BULK_ERECES_AVAILABLE;
```

To find out how many records will be returned for a bulk-record navigation query, use these calls:

- The Java `ENEQuery.getNavNumBulkERecs()` method and the .NET `ENEQuery.NavNumBulkERecs` property are for Oracle Commerce records.
- The Java `ENEQuery.getNavNumBulkAggrERecs()` method and the .NET `ENEQuery.NavNumBulkAggrERecs` property are for aggregated Oracle Commerce records.

Note that all of the above calls are also available in the `UrlENEQuery` class.

The following examples set the maximum number of bulk-format records to 5,000 for a navigation query.

Java example

```
// Set MDEX Engine connection
ENEConnection nec = new HttpENEConnection(eneHost, enePort);
// Create a query
ENEQuery usq = new UrlENEQuery(request.getQueryString(), "UTF-8");
// Specify the maximum number of records to be returned
usq.setNavNumBulkERecs(5000);
// Make the query to the MDEX Engine
ENEQueryResults qr = nec.query(usq);
```

.NET example

```
// Set Navigation Engine connection
HttpENEConnection nec = new HttpENEConnection(ENEHost, ENEPort);
// Create a query
String queryString = Request.Url.Query.Substring(1);
ENEQuery usq = new UrlENEQuery(queryString, "UTF-8");
// Specify the maximum number of records to be returned
usq.NavNumBulkERecs = 5000;
// Make the request to the Navigation Engine
ENEQueryResults qr = nec.Query(usq);
```

Retrieving the bulk-format records

By using members from the `Navigation` class, you can retrieve the returned set of bulk-format records from the `Navigation` query object.

The list of Oracle Commerce records is returned from the MDEX Engine inside the standard `Navigation` object. The records are returned compressed in a gzipped format. The format is not directly exposed to the application developer; the developer only has access to the bulk data through the methods from the language being used. Note that the retrieval method depends on whether you have a Java or .NET implementation.

It is up to the front-end application developer to determine what to do with the retrieved records. For example, you can display each record's property and/or dimension values, as described in this guide. You can also write code to properly format the property and dimension values for export to an external file, such as a Microsoft Excel file or a CSV file.

Using Java Bulk Export methods

In a Java-based implementation, the list of Oracle Commerce records is returned as a standard Java `Iterator` object.

To access the bulk-record `Iterator` object, use one of these methods:

- `Navigation.getBulkERecIter()` returns an `Iterator` object containing the list of Oracle Commerce bulk-format records (`ERec` objects).
- `Navigation.getBulkAggrERecIter()` returns an `Iterator` object containing the list of aggregated Oracle Commerce bulk-format records (`AggrERec` objects).

The `Iterator` class provides access to the bulk-exported records. The `Iterator.next()` method will gunzip the next result record and materialize the per-record object. The methods in the `Iterator` class that allow access to the exported records are the following:

- `Iterator.hasNext()` returns `true` if the iterator has more records.
- `Iterator.next()` returns the next record in the iteration. The record is returned as either an `ERec` or `AggrERec` object, depending on which `Navigation` method was used to retrieve the iterator.

The following Java code fragment shows how to set the maximum number of bulk-format records to 5,000 and then obtain a record list and iterate through the list.

```
// Create a query
ENEQuery usq = new UrlENEQuery(request.getQueryString(), "UTF-8");
// Specify the maximum number of bulk export records
// to be returned
usq.setNavNumBulkERecs(5000);
// Make the query to the MDEX Engine
ENEQueryResults qr = nec.query(usq);
// Verify we have a Navigation object before doing anything.
if (qr.containsNavigation()) {
    // Get the Navigation object
    Navigation nav = ENEQueryResults.getNavigation();
    // Get the Iterator object that has the ERecs
    Iterator bulkRecs = nav.getBulkERecIter();
    // Loop through the record list
    while (bulkRecs.hasNext()) {
        // Get a record, which will be gunzipped
        ERec record = (ERec)bulkRecs.next();
        // Display its properties or format the record for export
        ...
    }
}
```

```
}
}
```

Using .NET bulk export methods

In a .NET application, the list of Oracle Commerce records is returned as an `ERecEnumerator` object.

To retrieve the `ERecEnumerator` object, use the `Navigation.BulkAggrERecEnumerator` or `Navigation.BulkERecEnumerator` property.

The following .NET code sample shows how to set the maximum number of bulk-format records to 5000, obtain the record list, and iterate through the collection. After the `ERecEnumerator` object is created, an enumerator is positioned before the first element of the collection, and the first call to `MoveNext()` moves the enumerator over the first element of the collection. After the end of the collection is passed, subsequent calls to `MoveNext()` return false. The `Current` property will gunzip the current result record in the collection and materialize the per-record object.

```
// Create a query
ENEQuery usq = new UrlENEQuery(queryString, "UTF-8");
// Set max number of returned bulk-format records
usq.NavNumBulkERecs = 5000;
// Make the query to the Navigation Engine
ENEQueryResults qr = nec.Query(usq);
// First verify we have a Navigation object.
if (qr.ContainsNavigation()) {
    // Get the Navigation object
    Navigation nav = ENEQueryResults.Navigation;
    // Get the ERecEnumerator object that has the ERecs
    ERecEnumerator bulkRecs = nav.BulkERecEnumerator;
    // Loop through the record list
    while (bulkRecs.MoveNext()) {
        // Get a record, which will be gunzipped
        ERec record = (ERec)bulkRecs.Current;
        // Display its properties or format for export
        ...
    }
}
```

Performance impact for bulk export records

The bulk export feature can reduce memory usage in your application.

Unneeded overhead is typically experienced when exporting records from an MDEX Engine without the Bulk Export feature. Currently, the front-end converts the on-wire representation of all the records into objects in the API language, which is not appropriate for bulk export given the memory footprint that results from multiplying a large number of records by the relatively high overhead of the Oracle Commerce record object format. For export, converting all of the result records to API language objects at once requires an unacceptable amount of application server memory.

Reducing the per-record memory overhead allows you to output a large number of records from existing applications. Without this feature, applications that want to export large amounts of data are required to split up the task and deal with a few records at a time to avoid running out of memory in the application server's

threads. This division of exports adds query processing overhead to the MDEX Engine which reduces system throughput and slows down the export process.

In addition, the compressed format of bulk-export records further reduces the application's memory usage.

Part 4

Dimension and Property Features

- *Property Types*
- *Working with Dimensions*
- *Dimension Value Boost and Bury*
- *Using Derived Properties*
- *Configuring Key Properties*

Chapter 15

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	<p>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.</p> <p>The accepted format is: <code>latvalue,lonvalue</code>, where:</p> <ul style="list-style-type: none">• <code>latvalue</code> is the latitude of the location in whole and fractional degrees. Positive values indicate north latitude and negative values indicate south latitude. Valid values are between -90 and 90.• <code>lonvalue</code> is the longitude of the location in whole and fractional degrees. Positive values indicate east longitude, and negative values indicate west longitude. Valid values are between -180 and 180. <p>For example, to indicate the Location geocode property located at 42.365615 north latitude, 71.075647 west longitude, specify: <code>42.365615,-71.075647</code></p>
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 their use. For details, see the section "Configuring Developer Studio for the use of Time and Duration Property Types" in the *Oracle Commerce Developer Studio Installation 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:

Name	Type	Enable sort	Enable record search	Record spec property	Show with record list	Show with record
TransactionID	Integer	No	No	Yes	Yes	Yes
TimeStamp	DateTime	No	No	No	Yes	Yes
Time	Time	No	No	No	Yes	Yes
DeliveryDelay	Duration	No	No	No	Yes	Yes

Properties: 4

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 *MDEX Engine 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 16

Working with Dimensions

This section provides information about how to handle and display Oracle Commerce 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 about 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: <code>Navigation.getDescriptorDimGroups()</code> .NET: <code>Navigation.DescriptorDimGroups</code>	Gets an object that has information about the dimension groups for the dimensions with descriptors in the current navigation state.
Java: <code>Navigation.getRefinementDimGroups()</code> .NET: <code>Navigation.RefinementDimGroups</code>	Gets an object that contains the dimensions with refinements available in the current navigation state.
Java: <code>Navigation.getIntegratedDimGroups()</code> .NET: <code>Navigation.IntegratedDimGroups</code>	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: <code>DimGroupList.size()</code> .NET: <code>DimGroupList.Count</code>	Used on the <code>DimGroupList</code> object to initiate a loop over all the dimension groups, implicit and explicit. Once this loop is initiated, a <code>DimGroup</code> object is created.
Java: <code>DimGroup.getId()</code> .NET: <code>DimGroup.Id</code>	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: <code>DimGroup.getName()</code> .NET: <code>DimGroup.Name</code>	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: <code>DimGroup.size()</code> .NET: <code>DimGroup.Count</code>	Used in initiating a loop in order to access the dimensions in the group.
Java: <code>DimGroup.getDimension()</code> .NET: <code>DimGroup.GetDimension</code>	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 enable 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 enable 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 about 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 enables 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 `Nv` 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 `Nv` (Exposed Refinements) parameter.

The `Nv` 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 `Nv` 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: <code>Navigation.getRefinementDimensions()</code> .NET: <code>Navigation.RefinementDimensions</code>	Returns a <code>DimensionList</code> object that has dimensions that potentially still have refinements available with respect to this query.
Java: <code>Navigation.getDescriptorDimensions()</code> .NET: <code>Navigation.DescriptorDimensions</code>	Returns a <code>DimensionList</code> object that has the dimensions for the descriptors for this navigation.
Java: <code>Navigation.getIntegratedDimensions()</code> .NET: <code>Navigation.IntegratedDimensions</code>	Returns a <code>DimensionList</code> object that has the dimensions integrated from the refinement dimensions and the descriptor dimensions.
Java: <code>Navigation.getCompleteDimensions()</code> .NET: <code>Navigation.CompleteDimensions</code>	Returns a <code>DimensionList</code> 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: <code>Navigation.getRefinementDimGroups()</code> .NET: <code>Navigation.RefinementDimGroups</code>	Returns a <code>DimGroupList</code> object that contains the dimensions that potentially have refinements available in the current navigation state.
Java: <code>Navigation.getDescriptorDimGroups()</code> .NET: <code>Navigation.DescriptorDimGroups</code>	Returns a <code>DimGroupList</code> object that contains the dimension groups of the dimensions for the descriptors for this navigation.
Java: <code>Navigation.getIntegratedDimGroups()</code> .NET: <code>Navigation.IntegratedDimGroups</code>	Returns a <code>DimGroupList</code> object that contains the dimension groups of the dimensions integrated from the refinement and descriptor dimensions.

API method or property	Purpose
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 that extract standard and implicit refinements from dimensions.

Extracting standard refinements from a dimension

When a refinement dimension has been retrieved, these calls can extract 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 enable 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.Refinelements;
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 `DimValidIdList` 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);
DimValidIdList 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];
DimValidIdList 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.

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 Oracle Commerce 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 enables 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 Oracle Commerce project configuration XML files used with Forge, Workbench, and Developer Studio. You also do not change any settings in Oracle Commerce 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 Presentation API methods, or URL parameters. For information, see the topics in this section.

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.

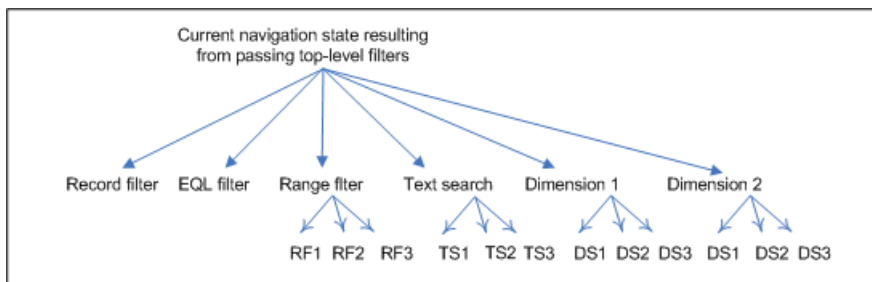
About top-level filters used for computing the base navigation state

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 enable 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 about this class and methods, see the *Oracle Commerce Presentation API for Java Reference (Javadoc)*.

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 about this class and property, see the *Oracle Commerce 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 Oracle Commerce 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.
- `<eqfilterinbase>` — a true or false value indexing whether the EQL filter is part of the base navigation state.
- `<textsearchesinbase>` — a true or false value indexing whether text searches are part of the base navigation state.
- `<rangefiltersinbase>` — a true or false value indexing 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+eqfilterinbase+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 enables 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 enables 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.

The screenshot shows the 'Dimension: Winery' dialog box with the 'Dynamic Ranking' tab selected. The 'Name' field contains 'Winery' and the 'ID' field contains '11'. The 'Member of this dimension' dropdown is set to '[None]' and the 'Refinements sort order' dropdown is set to 'Alpha'. In the 'Dynamic Ranking' section, the 'Enable dynamic ranking' checkbox is checked. The 'Maximum dimension values to' field is set to '20'. The 'Sort dimension' dropdown is set to 'Dynamically'. The 'Generate "More..." dimension value' checkbox is also checked. At the bottom, there are buttons for 'Help', 'OK', and 'Cancel'.

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: <ul style="list-style-type: none"> • 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

Field**Meaning**

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**.

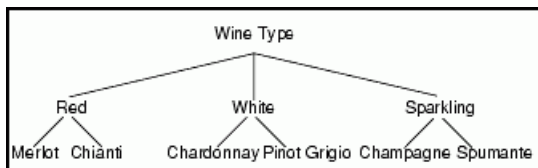
Using query-time control of dynamic refinement ranking

You can configure dynamic refinement ranking to be used on a per-query basis.

The Oracle Commerce 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);
refConf.setDynamicRefinementCount(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.

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 is accomplished with the `dgraph --esampmin` option, which enables 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	<code>Navigation.getDescriptorDimensions()</code> method
.NET	<code>Navigation.DescriptorDimensions</code> property

These calls return descriptor dimension values.

An alternative way is to use:

Option	Description
Java	<code>Navigation.getDescriptorDimGroups()</code> method
.NET	<code>Navigation.DescriptorDimGroups</code> 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
<code>Dimension.getDescriptor()</code> method (Java) and <code>Dimension.Descriptor</code> property (.NET)	Retrieve the dimension value that has been selected from this dimension.
<code>Dimension.getAncestors()</code> method (Java) and <code>Dimension.Ancestors</code> 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() %>">
</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() %>">
</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 Oracle Commerce MDEX Engine and are displayed in the user interface.

By providing the user with an indexation of the number of records (or aggregated records) that will be returned for each refinement, dimension value statistics can enhance the Oracle Commerce application's navigation controls by providing more context at each point in the Oracle Commerce 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.

The screenshot shows a dialog box titled "Dimension: Winery". It has a blue header bar with a close button (X) on the right. The dialog is divided into several sections. At the top, there are two text boxes: "Name:" containing "Winery" and "ID:" containing "11". Below these are two dropdown menus: "Member of this dimension" set to "[None]" and "Refinements sort order:" set to "Alpha". A tabbed interface is present with four tabs: "General", "Search", "Advanced" (which is selected and highlighted in blue), and "Dynamic Ranking". Under the "Advanced" tab, there are several options: "Primary" (unchecked), "Enable for rollup" (unchecked), "Compute refinement statistics" (checked), and "Collapsible dimension" (with an empty text box). To the right of these options is a "Multiselect" section with three radio buttons: "None" (selected), "Or", and "And". At the bottom of the dialog are three buttons: a "Help" button with a question mark icon, an "OK" button, and a "Cancel" button.

4. Click **OK**.

Only the configured dimensions will be considered for computation of dynamic dimension value statistics by the Oracle Commerce 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 Oracle Commerce 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"] + ")";
    }
}
```

Performance impact of refinement counts

Dynamic statistics on regular and aggregated records are expensive computations for the Oracle Commerce MDEX Engine.

You should only enable a dimension for dynamic statistics if you intend to use the statistics in your Oracle Commerce-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 Oracle Commerce-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 Oracle Commerce 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 enable 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 enable 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 contrast, 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`

☒ `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`

☒ `Chinese`

☒ `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 permits 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 enable 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.

The screenshot shows a dialog box titled "Dimension: Authors". It has a blue header bar with a close button (X). The dialog is divided into several sections. At the top, there are fields for "Name:" (containing "Authors") and "ID:" (containing "8048"). Below these are two dropdown menus: "Member of this dimension" (set to "[None]") and "Refinements sort order:" (set to "Alpha"). A tabbed interface is present with tabs for "General", "Search", "Advanced", "Dynamic Ranking", and "Cluster Discovery". The "General" tab is active, showing several checkboxes: "Prepare sort offline" (unchecked), "Show with record list" (unchecked), "Hidden" (checked), and "Show with record" (unchecked). There is also a "Language:" dropdown menu set to "English". At the bottom, there are buttons for "Help" (with a question mark icon), "OK", and "Cancel".

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 Oracle Commerce 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 Oracle Commerce Presentation API and Oracle Commerce 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 indexator 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 Oracle Commerce 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.Refineements;
// 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.

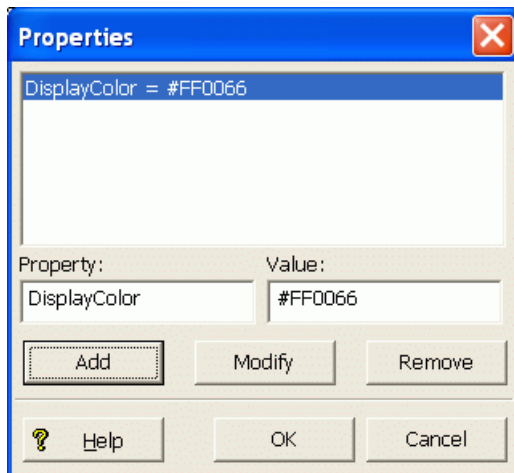
Instead, the Presentation API uses the information to support display features. For example, a property could contain the URL of an icon that should be displayed next to the dimension value.

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 Oracle Commerce 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

Oracle Commerce applications can use dimensions created outside of Developer Studio.

You can also import or otherwise access dimensions created or managed outside of Oracle Commerce Developer Studio. For details, see the *Platform Services Forge Guide*.

Chapter 17

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 enables 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.

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 Oracle Commerce Workbench. Using Workbench, 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.

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 Oracle Commerce 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: <code>AggERec.getProperties()</code> .NET: <code>AggERec.Properties</code>	Returns a <code>PropertyMap</code> object that has the derived properties of the aggregated record.
Java: <code>AggERec.getRepresentative()</code> .NET: <code>AggERec.Representative</code>	Returns an <code>ERec</code> 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 = aggrec.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 19

Configuring Key Properties

This section describes how to annotate property and dimension keys with metadata using key properties.

About key properties

Oracle Commerce 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 enables the application to present reasonable end-user choices for analytics measures. Knowledge of the set of date/time properties enables 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 enables 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 makes a cleaner application design that requires less maintenance over time as the data changes.

In addition to product-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 enables the application to render units on things like charts, while also enabling 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 enables 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 Oracle Commerce 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) 2001-2014, Oracle, 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 Oracle Commerce MDEX Engine automatically populates the following properties for each key: `Endeca.Type`, `Endeca.RecordFilterable`, `Endeca.DimensionId`, `Endeca.PrecedenceRule`, and `Endeca.MultiSelect`

Property	Description
<code>Endeca.Type</code>	The type of the key. Value is one of: <code>Dimension</code> , <code>String</code> , <code>Double</code> , <code>Int</code> , <code>Geocode</code> , <code>Date</code> , <code>Time</code> , <code>DateTime</code> , or <code>RecordReference</code> .
<code>Endeca.RecordFilterable</code>	indicates whether this key is enabled for record filters. Value one of: <code>true</code> or <code>false</code> .
<code>Endeca.DimensionId</code>	The ID of this dimension (only specified if <code>Endeca.Type=Dimension</code>).
<code>Endeca.PrecedenceRule</code>	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.
<code>Endeca.MultiSelect</code>	If <code>Endeca.Type=Dimension</code> and this dimension is enabled for multi-select, then this key property indicates the type of multi-select supported. Value one of: <code>OR</code> or <code>AND</code> .

Key property API

Key properties can be requested as part of an Oracle Commerce 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);

```


Basic Search Features

- [*Record Searches*](#)
- [*Search Interfaces*](#)
- [*Dimension Searches*](#)
- [*Record and Dimension Search Reports*](#)
- [*Using Search Modes*](#)
- [*Using Boolean Search*](#)
- [*Using Phrase Search*](#)
- [*Using Snippeting in Record Searches*](#)
- [*Using Wildcard Search*](#)
- [*Search Characters*](#)
- [*Examples of Query Matching Interaction*](#)
- [*Spelling Correction and Did You Mean*](#)
- [*Stemming and Thesaurus*](#)
- [*Automatic Phrasing*](#)
- [*Stop Words*](#)
- [*Relevance Ranking*](#)
- [*Record Boost and Bury*](#)

Chapter 20

Record Searches

This section is an introduction to record search, the Oracle Commerce equivalent of full-text search.

Keyword search overview

Users can perform two types of queries that perform searches based on keywords: *record searches* and *dimension searches*.

Keyword search enables a user to perform a 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.

Keyword search returns a complete Navigation object, which is the same object that is returned when a user filters records by selecting a dimension value. The keyword 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.

Dimension searches return dimension values containing the search term(s) that the user has specified. Dimension searches do not use search keys.

Example of keyword 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 a record property named `Description`, using the keyword `dense`, a `Navigation` object containing the following items 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`, a `Navigation` object containing these items is returned:

- 2 records (records 1 and 4)
- 2 refinement dimension values (Red and Other)



Note: 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 later in this document.

Making properties or dimensions searchable

The first step in implementing basic record search is to configure a property or dimension for record searching.

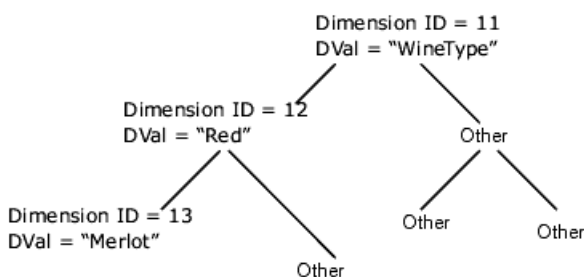
Hierarchical record searches

A record search is called *hierarchical* if it takes into consideration the ancestor dimension values of a specified dimension value.

You can specify that a record search is to be hierarchical in Developer Studio.

Record searches are non-hierarchical by default – that is, they return only those records that are assigned a dimension value whose text matches the search term.

The following example illustrates the difference between non-hierarchical and hierarchical record searches:



In the `WineType` dimension, the dimension value `Red` (with an ID of 12) is an ancestor of the dimension value `Merlot` (ID of 13). Non-hierarchical and hierarchical record searches work as follows in the `WineType` dimension:

- A non-hierarchical record search using the keyword `merlot` selects any records assigned the dimension ID 13, but selects no records if the keyword is `red merlot`.
- A hierarchical search using the keyword `red merlot` selects the same records, that is, any records assigned the dimension ID 13.

Adding search synonyms to dimension values

You can add synonyms to a dimension value so that searches on the synonyms will select the same records as searches on the original dimension value.

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.

Synonyms can be added only to child dimension values, and not to root dimension values.

Features for controlling record search

You can control the features related to record search either at indexing time or at run-time.

The following statements describe the aspects of record search behavior that you can control:

- 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 [Search Interfaces](#) on page 219.
- Dgidx flags are not needed to enable record search. If a property or dimension was properly enabled for record search, it will automatically be indexed for searching.
- Dgraph (MDEX Engine) configuration flags are not needed to enable record searching. If a property or dimension was properly enabled for record searching during 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.

URL query parameters for record search

A basic record search requires two request URL parameters, `Ntk` and `Ntt`. This topic describes `Ntk` and `Ntt` and provides examples of their use.

The search key parameters are described as follows:

- `Ntk=search_key` . The search key parameter `Ntk` specifies the property or dimension that will be examined by the search. 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 search term.

The URL query parameters for record searches have the following characteristics:

- The navigation dimension value (`N`) and the record search parameters (`Ntk`, `Ntt`) are joined by a Boolean AND.
- Only records selected by the navigation dimension value (`N`) are examined by the record search.
- Record search parameters must accompany a standard navigation request, even if that basic navigation request is empty (`N=0`).

- A request without a Navigation value (N) is invalid, even if a property key (Ntk) and keyword (Ntt) are specified.

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...

For example, the following query returns records 1 and 4, because the navigation request is empty (N=0)

```
<application>?N=0&Ntk=Description&Ntt=Ripe
```

However, the following query returns only record 1, because the navigation request (N=101) is already filtering the record set to records 1 and 2:

```
<application>?N=101&Ntk=Description&Ntt=Ripe
```

The following query is invalid because it does not include a navigation request (N):

```
<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 about performing more complex Boolean queries, see topics about using Boolean search.

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 in 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. Oracle Commerce 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 permitted by the record filter.

Step 2: Oracle Commerce Query Language filters

The Oracle Commerce Query Language (EQL) contains a rich syntax that enables 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.

Step 3: Tokenization

Tokenization is the process by which the MDEX Engine breaks up compound phrases in the search query string into their constituent words or characters, producing 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`.

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"`.

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`.

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.

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.

Step 9: Range filtering

Range filter functionality enables 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.

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.



Note: Recommended practice is to use the Experience Manager, rather than using business rules and user profiles directly. For detailed information about the Experience Manager, refer to the *Workbench User's Guide*.

Step 12: Analytics

Oracle Commerce Analytics builds on the core capabilities of the Oracle Commerce 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 *MDEX Engine 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.

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 enabled 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 Oracle Commerce properties, these are normally used for debugging purposes that do not explicitly know this information for a given data set.

In production systems, these Oracle Commerce properties are typically hard-coded at the application level, because the application requires specific search keys to be used for specific functionality.

If an Oracle Commerce 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 `Netk` 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 and the dgraph (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 21

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 enables you to control record search behavior for groups of one or more properties and dimensions.

A search interface may also contain:

- A number of attributes, such as name, cross-field information, and so on.
- An ordered collection of one or more ranking strategies.

Some of the features that can be specified for a search interface include:

- Relevance ranking
- Matching across multiple 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 enabling cross-field matches

The *enable Cross-field Matches* attribute of Search Interface editor in Developer Studio specifies when the MDEX Engine should try to match search queries across dimension or property boundaries.

The three settings for *enable Cross-field Matches* are:

Setting	Description
<i>Always</i>	<p>The MDEX Engine always looks for matches across dimension or property boundaries, in addition to matches within a dimension or property.</p> <p>If you choose to use cross-field matching, the <i>Always</i> setting is recommended and is the default.</p> <p>For example, in the <code>Sony camera</code> user query, if <i>enable Cross-field Matches</i> is set to <i>Always</i>, the MDEX Engine returns all matches with <code>Brand = Sony</code> and <code>Product_Type = camera</code>.</p>
<i>Never</i>	The MDEX Engine does not look across boundaries for matches.
<i>On Failure</i>	<p>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.</p> <p> Note: In most cases, the <i>Always</i> setting provides better results than the <i>On Failure</i> setting.</p>

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, a search interface named `MoviePeople` 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 enable 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 return only 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 *enable 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: Sachin Tendulkar
P_Tag: Jersey
```

A search against P_Tag for "sachin tendulkar 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 specify 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 permits 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.

Dimension Searches

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 enable users to perform keyword searches across dimensions for dimension values with matching names.

The result of a dimension search is a results object that contains dimension values.

The application can present these dimension values to the end-user, enabling the user to select them and create a new navigation request.

Depending on the type of dimension search that you are using, dimension values are 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 enables 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 enable the application to do the following:

- Request only the first `n` dimension values for each dimension. An additional parameter enables 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.

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 `--compoundDimSearch` 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 (an alternative to using the `Di` parameter only) is to limit the number of dimension values to return with each dimension, using the `numresults` option of the `Drc` parameter.

The `Drc` parameter depends on the Dimension Search (D) parameter.



Note: The `Drc` parameter is not supported with compound dimension search.

To limit the number of dimension values to return with each dimension:

1. In a query, specify the `Drc` parameter, the `numresults` option to the parameter, and an integer value that represents the maximum dimension value count to return.

For example, the following query:

```
D=red&Drc=id+11+numresults+1
```

returns only the following results:

Dimension	Dimension values
Wine_type	Red
Wineries	Green & Red
Drinkability	Drink with red meat

2. Optionally, use the Refinement Configuration for Dimension Search parameter (`Drc`) with the `numresults` option and the `id` option.

In this case only the first `n` dimension values for the specified dimension are returned.

For example, the following query that contains a dimension search where the `Winery` dimension has an ID of 11:

```
D=red&Drc=numresults+1|id+11
```

returns only the following results:

Dimension	Dimension values
Wineries	Green & Red

Enabling result paging

To enable an application to page through dimension search results, use the `numresults` option of the Refinement Configuration for Dimension Search parameter (`Drc`) 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 enables 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 `Drc` and `Di` parameters.

Similar to other advanced dimension search parameters, the Search Results Offset parameter (`Do`) is dependent on the Dimension Search parameter. Although it is not enforced, the Search Results Offset parameter is most frequently used in conjunction with both the `Drc` and `Di` parameters.

For example, the following dimension search query with these parameters:

```
D=red&Drc=numresults+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

Dimension	Dimension values
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; 1 < 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 Oracle Commerce 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. The `Drc` parameter has a configuration option to enable refinement counts (the `showcounts` setting). You can specify a list of dimension values by `Id` for the `showcounts` setting, or you can omit the `Id` value to enable refinement counts globally for all dimension values in an application.

No Developer Studio configuration or `dgraph` flags are required to enable this feature. For details and examples of `Drc`, see [Drc \(Refinement Configuration for Dimension Search\)](#) on page 407.

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 examples

This example gets the refinement counts for all the dimension values. Assume that the refinement counts are enabled globally for all dimension values.

```
if (results.containsDimensionSearch()) {
    DimensionSearchResult result = results.getDimensionSearch();
    DimensionSearchResultGroupList groups = result.getResults();

    for (DimensionSearchResultGroup g :
        (List<DimensionSearchResultGroup>)groups) {
        for (DimLocationList l : (List<DimLocationList>)g) {
            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") + ")";
            }
        }
    }
}
```

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.getDimensionSearchResultGroup(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 examples

This example gets the refinement counts for all the dimension values. Assume that the refinement counts are enabled globally for all dimension values.

```

if (results.ContainsDimensionSearch()) {
    DimensionSearchResult result = results.DimensionSearch;
    DimensionSearchResultGroupList groups = result.Results;
    for (int gg = 0; gg < groups.Count; ++gg) {
        DimensionSearchResultGroup group = (DimensionSearchResultGroup)groups[gg];

        for (int ii = 0; ii < group.Count; ++ii) {
            DimLocationList l = (DimLocationList)group[ii];
            DimLocation loc = (DimLocation)l[0];
            DimVal dimVal = loc.DimValue;
            PropertyMap pmap = dimVal.Properties;
            String dstats = "";
            if (pmap["DGraph.Bins"] != null) {
                dstats = " (" + pmap["DGraph.Bins"] + ")";
            }
        }
    }
}

```

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.GetDimensionSearchResultGroup(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	<p>In general, datasets with little descriptive text and extensive dimension values that represent the most frequently searched terms (for example, <code>autos</code>) are a good fit for dimension search.</p> <p>Keyword searches are usually oriented towards such keywords, as for example, <code>make</code>, <code>model</code>, <code>year</code>, and so on, which would probably be included in the list of dimensions.</p> <p>For example, searching for <code>Ford</code> would return a single dimension value from the <code>Make</code> dimension.</p>
Record search	<p>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.</p> <p>In such cases, record search enables an application to search directly against record text (such as the body of an article).</p>



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 about 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.

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.

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 `getESearchReportsComplete()` methods (Java) and `ESearchReportsComplete` properties (.NET) on the `DimensionSearchResult` and `Navigation` classes.

Both the `DimensionSearchResult` and `Navigation` classes have `getESearchReportsComplete()` methods (Java), and `ESearchReportsComplete` properties (.NET) that return a `Map` (Java), and an `IDictionary` (.NET) of search keys to a `List` of `ESearchReport` objects. In the dimension search case, the single search report is associated with the literal string `Dimension Search`.

You can also use these methods/properties if you have performed a multiple search (that is, using the `Ntk` and `Ntt` parameters with two or more search keys and terms). These accessors return a `Map` (Java) and an `IDictionary` (.NET) of `List` (Java) and `IList` (.NET) objects that contain `ESearchReports` objects.

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: <code>getKey()</code> NET: <code>Key</code>	Returns the search key used in the current search.
Java: <code>getTerms()</code> .NET: <code>Terms</code>	Returns the search terms as a single String.
Java: <code>getNumMatchingResults()</code> .NET: <code>NumMatchingResults</code>	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: <code>getSearchMode()</code> NET: <code>SearchMode</code>	Returns the requested match mode.
Java: <code>getMatchedMode()</code> .NET: <code>MatchedMode</code>	Returns the selected match mode. This is different than <code>getSearchMode()</code> (Java) and <code>SearchMode</code> (.NET) in that <code>getMatchedMode()</code> (Java) and <code>MatchedMode</code> (.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: <code>getNumMatchedTerms()</code> .NET: <code>NumMatchedTerms</code>	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: <code>getAutoSuggestions</code> NET: <code>AutoSuggestions</code>	Is used for autosuggest (alternate spelling correction) results and returns a <code>List</code> (Java), and an <code>ICollection</code> (.NET) of <code>ESearchAutoSuggestion</code> objects.
Java: <code>getDYMSuggestions</code> .NET: <code>DYMSuggestions</code>	Is used for "Did You Mean" results and returns a <code>List</code> an <code>ICollection</code> of <code>ESearchDYMSuggestion</code> 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 about 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 Lists ESearchReport objects
Map recSrchRpts = nav.getESearchReportsComplete();

// Declare the search key being sought
String desiredKey = "my_search_interface";
if (recSrchRpts.containsKey(desiredKey)) {

    // Get the list of ERecSearchReports for the desired search key
    List srchReportList = (List)recSrchRpts.get(desiredKey);

    for (Iterator i = srchReportList.iterator(); i.hasNext()) {
        ESearchReport srchReport = (ESearchReport)i.next();

        // 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 matchallMsg = "";
        if (mode == ESearchReport.MODE_ALL) {
            matchallMsg = "MatchAll mode was used";
        }

        // Print or log the message
        ...
    }
}

```

.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
    IList srchrptList = (IList)recSrchrpts[desiredKey];

    foreach (object ob in srchrptList) {
        ESearchReport srchrpt = (ESearchReport)ob;

        // Get the search term submitted for this search report
        String srchrptTerms = srchrpt.Terms;

        // Get the number of matching results
        long numMatchingResults = srchrpt.NumMatchingResults;

        // Get the match mode that was used for this search
        ESearchReport.Mode mode = srchrpt.MatchedMode;

        // Display a message if MatchAll mode was used by
        // Navigation Engine
        String matchallMessage = "";
        if (mode == ESearchReport.MODE_ALL) {
            matchallMessage = "MatchAll mode was used";
        }

        // Print or log the message
        ...
    }
}

```

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 24

Using Search Modes

By default, Oracle Commerce 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 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.
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 Oracle Commerce 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 enable 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 enables 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 enables 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 that 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 needed 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 25

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 enables users to specify complex expressions that describe the exact search criteria with which they would like to search.

Oracle Commerce 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 Oracle Commerce 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 254

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 255

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 enables 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" ONEAR/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 enabled in Boolean search queries.

The following statements describe semantics of Boolean query operators:

- The `AND` operator executes an intersection of its two operands.
- The `OR` operator executes a union of the two operands.
- The `AND NOT` operator executes a set subtract, subtracting the second operand from the first.
- The parentheses operators have two meanings, depending on their usage:

- They can either be used to group sub-expressions, as in `"(red or blue) and car"`
- Or, they can be used as `AND` operators in themselves.

For example, the query `"(red or blue) car"` automatically treats the `") "` as a `") AND "`. Thus the query would be treated as `"(red or blue) and car"`.

The same is true for usage of the left parenthesis.

- Words or phrases grouped together without any explicit operators (such as `"red car or blue bicycle"`) are also queried conjunctively.

Thus the example query would return the results for `"(red and car) or (blue and bicycle)"`. Similarly, `"red car" "blue bicycle"` will return the results for `"red car" AND "blue bicycle"`.

- As the examples demonstrate, operator names are not case sensitive, although field names are.

Operator precedence

The **NOT** operator has the highest precedence, followed by the **AND** operator, followed by the **OR** operator. You can always control the precedence by using parentheses.

For example, the expression "A OR B AND C NOT D" is interpreted as "A OR (B AND C AND (NOT D))".

Interaction of Boolean search with other features

The following table describes whether various features are supported for queries that execute a Boolean search (including the proximity operators).

Feature	Support with Boolean search	Comments
Stemming	Yes	
Thesaurus matching	No	
Misspelling correction	No	Auto-correct and "Did You Mean" are not supported.
Relevance ranking	No	
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 enabled.	The final result set is not enabled to be the result of a negation operation.
(Unexpected end of expression.	
Sony OR NOT Aiwa	The <first second> clause of the OR at position <position> is a negation. Neither clause of an OR expression may be a negation.	Neither clause of an OR expression can be the result of a negation operation.
Sony OR	Unexpected end of expression.	
Sony AND	Unexpected end of expression.	
Sony NOT	Unexpected end of expression. Expecting an opening left parenthesis, a word, or a phrase.	
(Sony	Unexpected end of expression. Expecting closing right parenthesis.	
Manufacturer:(Sony OR Item: Camera)	The key restrict operator may not be used within another key restrict expression.	
Manufacturer:	Unexpected end of expression. The key restrict operator must be followed by a word, a phrase, or a left parenthesis.	
Manufacturer:OR	The key restrict operator must be followed by a word, a phrase, or a left parenthesis.	
Foo:Sony	Unknown search index name "Foo" used for restrict operator	The 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 an `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 sample in Java shows how to obtain the information in the `ESearchReport` object:

```
// Get the Map of ESearchReport objects
Map recSrchrpts = nav.getESearchReportsComplete();
if (recSrchrpts.size() > 0) {

    // Get the user's search key
    String searchKey = request.getParameter("Ntk");
    if (searchKey != null) {
        if (recSrchrpts.containsKey(searchKey)) {
            // Get the List of ERecSearchReports for the search key
            List srchrptList = (List)recSrchrpts.get(searchKey);
            for (Iterator i = srchrptList.iterator(); i.hasNext()) {
                ESearchReport srchrpt = ESearchReport(i.next());
                // 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 sample in .NET shows how to obtain the information in the `ESearchReport` object:

```
// Get the Dictionary of ESearchReport objects
IDictionary recSrchrpts = nav.ESearchReportsComplete;

// Get the user's search key
String searchKey = Request.QueryString["Ntk"];

if (searchKey != null) {
    if (recSrchrpts.Contains(searchKey)) {
        // Get the List of ERecSearchReports for the search key
        IList srchrptList = (IList)recSrchrpts[searchKey];
        foreach (object ob in srchrptList) {
```

```
    ESearchReport srchReport = (ESearchReport)ob;

    // 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 26

Using Phrase Search

Phrase search enables users to specify a literal string to be searched. This section discusses how to use phrase search.

About phrase search

Phrase search enables 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 enables the user to put double-quote characters around the search term, thus specifying a literal string to be searched. Results of a phrase search contain all of the words specified in the user's search (not stemming, spelling, or thesaurus equivalents) in the exact order specified.

For example, if the user enters the phrase query `"run fast"`, the search finds text containing the string `run fast`, but not text containing strings such as `fast run`, `run very fast`, or `running fast`, which might be returned by a normal non-phrase query.

Additionally, phrase search queries do not ignore stop words. For example, if the word `the` is configured as a stop word, a phrase search for `"the car"` does not return results containing simply `car` (not preceded by `the`).

Also, phrase search permits 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 Oracle Commerce 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 search for phrase `cd player` is as follows:

```
N=0&Ntk=All&Ntt=%22cd+player%22
```

- A record search 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, enabling 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.

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](#)



Note: Snippet properties, unlike other Oracle Commerce properties, are not created, configured, or mapped using Developer Studio. A dynamically generated snippet property is not tagged to an Oracle Commerce record. The snippet property appears with a record only on a record list page.

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 enable you more easily to 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 snippetting.

For example, if you enable snippetting 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.

About enabling and configuring snippetting

You enable the snippetting 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, snippetting 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 snippetting

You can configure snippetting 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 snippetting.

Providing these values in a URL overrides any configuration options specified in a Developer Studio project file.

You can disable snippetting 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 snippetting

You can include snippetting only in record search operations. The following are examples of snippetting 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+description: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 `` (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 Oracle Commerce properties. For details, see the section about Displaying Oracle Commerce 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 28

Using Wildcard Search

Wildcard search enables 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, Oracle Commerce 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 about 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 about these configuration options.

Configuring wildcard search with Dimension and Property editors

The **Dimension** and **Property** editors of Developer Studio enable you to permit wildcard searches for any Oracle Commerce 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 permit wildcard search.
- Check the **Enable Record Search** option in both editors for the specified Oracle Commerce property or dimension.



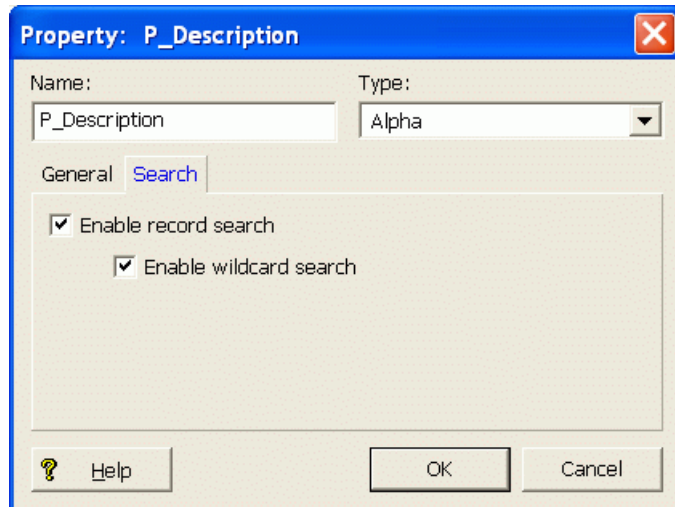
Note: If you use this method, you will only affect records enabled for search, but not dimensions enabled for search. (For dimensions enabled for search, you can permit wildcard search for ALL dimensions at once.)

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.

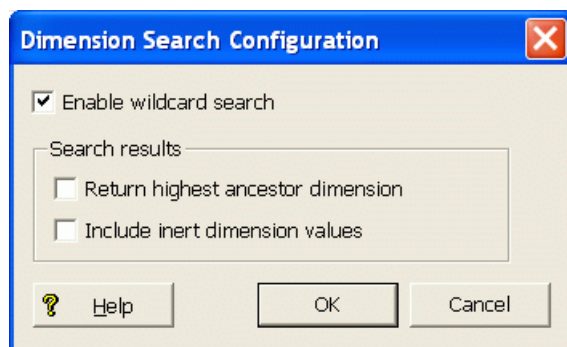
Unlike the option for enabling wildcard search in the **Search** tab of the **Dimension** editor, which affects only a single dimension, the **Dimension Search Configuration** editor globally sets the options for all dimensions in a project.



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 about tuning this parameter, see the *MDEX Engine 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 Oracle Commerce 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, Oracle 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 *MDEX Engine Performance Tuning Guide*.



Note: You enable wildcard search in Developer Studio.

Chapter 29

Search Characters

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

Using search characters

The Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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 Oracle Commerce 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.

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 results when	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	Yes	Yes

	the query is executed.	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

Spelling Correction and Did You Mean

This section describes how to implement the Spelling Correction and Did You Mean features of the Oracle Commerce MDEX Engine.

About Spelling Correction and Did You Mean

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

Spelling Correction operates by computing alternate spellings for user query terms, evaluating the likelihood that these alternate spellings are the best interpretation, and then using the best alternate spell-corrected query forms to return extra search results. For example, a user might search for records containing the text *Abrham Lincoln*. With spelling correction enabled, the Oracle Commerce MDEX Engine will return the expected results: those containing the text *Abraham Lincoln*.

The list of alternate spellings that can be used for spelling correction is known as the spelling dictionary.

Did You Mean (DYM) functionality enables an application to suggest alternatives to search terms that users enter. For example, if users search for *valle* in the sample wine data, they get six results. The terms *valley* and *vale*, however, are much more prevalent (2,414 results and 20 results respectively.) When this feature is enabled, the MDEX Engine responds with the six results for *valle*, but also suggests that *valley* or *vale* may be what the user intended. If multiple suggestions are returned, they are sorted and presented according to the closeness of the match.

The Oracle Commerce MDEX Engine supports two complementary forms of Spelling Correction:

- Auto-correction for record search and dimension search.
- Explicit spelling suggestions for record search (the "Did you mean?" dialog box).

Either or both features can be used in a single application, and all are supported by the same underlying spelling engine and Spelling Correction modules.

The behavior of Oracle Commerce spelling correction features is application-aware, because the spelling dictionary for a given data set is derived directly from the indexed source text, populated with the words found in all searchable dimension values and properties. For example, in a set of records containing computer equipment, a search for *graphi* might spell-correct to *graphics*. In a different data set for sporting equipment, the same search might spell-correct to *graphite*.

Oracle Commerce Spelling Correction features include a number of tuning parameters to control performance, behavior, and result presentation. This section describes the steps necessary to enable spelling correction for

record and/or dimension search, and provides a reference to the tuning parameters provided to allow applications to obtain various behavior and performance trade-offs from the spelling engine.

Spelling modes

Oracle Commerce spelling features compute contextual suggestions at the full query level.

That is, suggestions may include one or more corrected query terms, which can depend on context such as other words used in the query. To determine these full query suggestions, the MDEX Engine relies on low-level spelling modules to compute single-word suggestions, that is, words similar to a given user query term and contained within the application-specific dictionary.

Aspell and Espell spelling modules

The MDEX Engine supports two internal spelling modules, either or both of which can be used by an application:

- `Aspell` is the default module. It supports sound-alike corrections (using English phonetic rules). It does not support corrections to non-alphabetic/non-ASCII terms (such as *café*, *1234*, or *A&M*).
- `Espell` is a non-phonetic module. It supports non-phonetic (edit-distance-based) correction of any term.

Generally, applications that only need to correct normal English words can enable just the default Aspell module. Applications that need to correct international words, or other non-English/non-word terms (such as part numbers) should enable the Espell module.

In certain cases (such as an English-language application that also needs to correct part numbers), both Aspell and Espell can be enabled.

Supported spelling modes

Module selection is performed at index time through the selection of a spelling mode. The supported spelling modes are (the options below represent command line options you can specify to Dgidx):

- `aspell` – Use only the Aspell module. This is the default mode.
- `espell` – Use only the Espell module.
- `aspell_OR_espell` – Segment the dictionary so that Aspell is loaded with all ASCII alphabetic words and Espell is loaded with other terms. Use Aspell when attempting to correct ASCII alphabetic words; use Espell to correct other words.
- `aspell_AND_espell` – Load each module with the full application dictionary. Use both modules to correct any word, selecting the best suggestions from the union of the results.
- `disable` – Disable the Spelling Correction feature.

Disabling spelling correction on individual queries

This topic describes how to disable spelling correction and DYM suggestions on individual queries.

You may discover that it is desirable to disable spelling correction in order to reduce the cost of running some queries in performance-sensitive applications. For example:

- Queries where the MDEX Engine needs to perform matching on a very large number of terms all of which need to be ranked for spelling correction suggestions.
- Queries using terms derived directly from the raw data. For example, if your end users are searching for terms that are unique to their field, it may be desirable to disable spelling correction suggestions for those terms.

To disable spelling correction for a particular query:

Use a query configuration option, `spell`, with a parameter `nospell`.

This option has the following characteristics:

- Works for both record and dimension search.
- Disables both Aspell and Espell spelling correction modes.
- Disables spelling correction and DYM suggestions.
- Requires spelling to be enabled in Dgidx or in the dgraph. Otherwise, this option has no effect.
- Requires that you provide a `nospell` parameter to it. Providing a parameter other than `nospell` results in a warning in the error log, and spelling correction proceeds as if the option were not provided to the MDEX Engine.
- Reduces the performance cost of a particular query. You can include this option in your front-end application for particular queries if you observe that disabling spelling correction is beneficial for increasing performance of your application overall. However, there is no need to modify your existing application if you don't observe a performance penalty from using spelling correction.

Examples

In the presentation API, use the `spell+nospell` option with `Ntx` and `Dx` parameters.

For example, to disable spelling correction for a dimension search query for "blue suede shoes", change the query from this syntax:

```
D=blue+suede+shoes&Dx=mode+matchallpartial
```

To the following syntax:

```
D=blue+suede+shoes&Dx=mode+matchallpartial+spell+nospell
```

In the dgraph URL, specify the `spell+nospell` value to the `opts` parameter. For example, change this type of query from this syntax:

```
/search?terms=blue+suede+shoes&opts=mode+matchallpartial
```

To the following syntax:

```
/search?terms=blue+suede+shoes&opts=mode+matchallpartial+spell+nospell
```

In the Java Presentation API, you can disable spelling for a specific query as shown in this example:

```
ENEQuery nequery = new ENEQuery();
nequery.setDimSearchTerms("blue suede shoes");
nequery.setDimSearchOpts("spell nospell");
```

In the .NET API, you can disable spelling for a specific query as shown in this example:

```
ENEQuery nequery = new ENEQuery();
nequery.DimSearchTerms = "blue suede shoes";
nequery.DimSearchOpts = "spell nospell";
```

Spelling dictionaries created by Dgidx

No index configuration setup is strictly necessary to enable spelling correction.

By default, all words contained in searchable dimensions and properties will be considered as possible spell correction recommendations. But in practice, to achieve the best possible spelling correction behavior and

performance, it is typically necessary to configure bounds on the list of words available for spelling correction, commonly known as the dictionary.

The application-specific spelling dictionary is created by Dgidx. As Dgidx creates search indexes of property and dimension value text, it accumulates lists of words available for spelling correction by the Aspell module into the following file:

- `<db_prefix>.worddat` (for the Aspell module only)

where `<db_prefix>` is the output index prefix.



Note: The `<db_prefix>.worddat` file for the Aspell module is reloaded into the MDEX Engine each time you run the `admin?op=updateaspell` administrative command. This command enables you to make updates to the Aspell spelling dictionary without stopping and restarting the dgraph.

This file contains application-specific dictionary words separated by new-line characters. Duplicate words listed in these files are ignored.

This file is automatically compiled by the Dgidx during the indexing operation.

Configuring spelling in Developer Studio

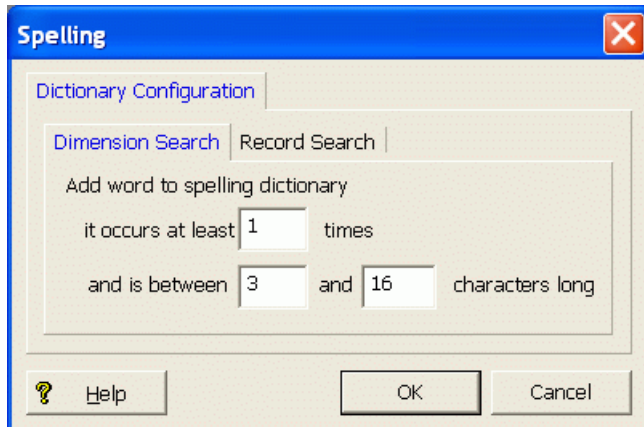
You can set constraints for the spelling dictionaries in Developer Studio.

By default, Dgidx examines dimensions and properties enabled for search and adds words that are larger than 3 characters and smaller than 16 characters to the dictionary. However, because performance of spelling correction in the MDEX Engine depends heavily on the size of the dictionary, you can set constraints on the contents of the dictionary. For example, you might choose to either increase the default from a minimum of 3 characters or reduce the default from a maximum of 16 characters. These configuration settings are useful for improving the performance of spell-corrected search operations at runtime.

These configuration options can be used to tune and improve the types of spelling corrections produced by the MDEX Engine. For example, setting the minimum number of word occurrences can direct the attention of the spelling correction algorithm away from infrequent terms and towards more popular (frequently occurring) terms, which might be deemed more likely to correspond to intended user search terms.

To configure spelling dictionary entries:

1. In the Project Explorer, expand **Search Configuration**.
2. Double-click **Spelling** to display the Spelling editor.



3. You can separately configure entries in the dictionary based for dimension search and record search. Therefore, select either the **Dimension Search** tab or the **Record Search** tab.

In this example, the **Dimension Search** tab is selected.

4. Set the constraints for adding words to the spelling dictionary:

Field	Description
it occurs at least n times	Sets the minimum number of times the word must appear in your source data before the word should be included in the spelling dictionary.
and is between n1 and n2 characters long	Sets the minimum (n1) and maximum (n2) lengths of a word for inclusion in the dictionary.

5. If desired, select the other tab and set the constraints for that type of search.
6. Click **OK**.
7. Choose **Save** from the File menu to save the project changes.

Modifying the dictionary file

You can modify or replace the Aspell dictionary file. Use the `admin?op=updateaspell` operation for the dgraph which causes updates to the Aspell dictionary file.

While the dictionary files automatically generated by Dgidx are generally adequate for most applications (especially when using a reasonable value for the minimum number of word occurrences), additional improvements in application-specific spelling behavior can be achieved through modification or replacement of the automatic dictionary file (Aspell module only).

For example, in applications with a specific set of technical terminology that requires focused spelling correction, you can replace the automatic dictionary with a manually-generated list of technical terms combined with a simple list of common words (such as `/usr/dict/words` on many UNIX systems).

About the `admin?op=updateaspell` operation

The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.

The `admin?op=updateaspell` operation performs the following actions:

- Crawls the text search index for all terms
- Compiles a text version of the `aspell` word list
- Converts this word list to the binary format required by `aspell`
- Causes the dgraph to finish processing all existing preceding queries and temporarily stop processing incoming queries
- Replaces the previous binary format word list with the updated binary format word list
- Reloads the `aspell` spelling dictionary
- Causes the dgraph to resume processing queries waiting in the queue

The dgraph applies the updated settings without needing to restart.

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

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

```
...
aspell update ran successfully.
...
```



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

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

Enabling language-specific spelling correction

If your application involves multiple languages, you may want to enable language-specific spelling correction. For information about how to enable this feature, see the "Using Internationalized Data" section.

Dgidx flags for Spelling Correction

The spelling mode can be selected using the Dgidx `--spellmode` flag.

The default spelling mode is `aspell`, which enables only the Aspell module.

The full set of supported spelling modes is:

- `--spellmode aspell`
- `--spellmode espell`
- `--spellmode aspell_OR_espell`
- `--spellmode aspell_AND_espell`
- `--spellmode disable`

Behaviors for these modes are described in the "Spelling modes" topic. If a spelling mode that includes use of the Espell module is enabled, an additional Dgidx flag, `--spellnum`, can be used to control the contents of the Espell dictionary.

The default is to disable `--spellnum`. With this flag enabled, the Espell dictionary will be allowed to contain non-word terms. A word term is one that contains only ASCII alphabetic characters and ISO-Latin1 word characters listed in Appendix C. In default mode, non-word terms are not allowed in the Espell dictionary.



Note: Auto-correct should be relatively conservative. You only want the engine to complete the correction when there is a high degree of confidence. For more aggressive suggestions, it is best to use Did You Mean.

dgraph flags for enabling Spelling Correction and DYM

Four dgraph flags enable the use of the Spelling Correction and DYM features. You can also use the `admin?op=updateaspell` operation on the dgraph to update the Aspell spelling dictionary while running partial updates (without having to stop and restart the MDEX Engine).

dgraph --spellpath flag

To enable use of spelling features in the MDEX Engine, you must first use the `--spellpath` flag to specify the path to the directory containing the spelling support files.

If you are using the Oracle Commerce Application Controller to provision and run the `dgraph`, then this flag is set automatically. By default, the `dgraph` component looks for the Aspell spelling support files in its input directory (that is, the `Dgidx` output directory). If you want to specify an alternative location, you can do so using the `spellPath` element in the WSDL, or by specifying arguments to the `dgraph` in Oracle Commerce Workbench.

If you need to, you can specify the `--spellpath` parameter yourself. The value of the `--spellpath` parameter typically matches the value specified for `--out` on the `dgwordlist` program.

Note the following about the `--spellpath` flag:

- The directory passed to the `--spellpath` flag must be an absolute path. Paths relative to the current working directory are not allowed. This directory must have write permissions enabled for the user starting the MDEX Engine process.
- The `--spellpath` option on the MDEX Engine is required for spelling features to be enabled, but this flag does not activate any spelling features on its own. Additional flags are required to enable actual spelling correction in the MDEX Engine.

Additional dgraph flags to enable spelling correction

The following MDEX Engine flags enable the supported spelling features. Any or all of these options can be specified in combination, because they control independent features.

dgraph flag	Spelling feature
<code>--spl</code>	Enables automatic spelling correction (autosuggest) for record and dimension searches.
<code>--dym</code>	Enables explicit spelling suggestions (Did You Mean) for record search operations
<code>--spell_bdgt num</code>	Sets the maximum number (<i>num</i>) of variants to be considered when computing any spelling correction (autosuggest). The default value is 32.

If `--spl` and `--dym` are both specified, explicit spelling suggestions are guaranteed not to reuse suggestions already consumed by automatic spelling correction (autosuggest). For example, the MDEX Engine will not explicitly suggest "Did you mean 'Chardonnay'?" if it has already automatically included record search results matching *Chardonnay*.

Spelling corrections generated by the MDEX Engine are determined by considering alternate versions of the user query. The computation and scoring of alternate queries takes time and can decrease performance, especially in the case of search queries with many terms. To limit the amount of spelling work performed for any single search query, use the `--spell_bdgt` flag to place a maximum on the number of variants considered for all spelling and Did You Mean corrections.

For information about other spelling-related flags, see the *dgraph Flags* topic in the *Oracle Commerce Administrator's Guide*.

URL query parameters for Spelling Correction and DYM

DYM suggestions are enabled by the `Nty` parameter.

No special URL query parameters are required for the dimension search and record search auto-correction features (`--spl` options). These features automatically engage when appropriate, given configuration settings and the user's query.



Note:

To disable spelling correction on individual queries, you can use the `Ntx` and `Dx` parameters with the `spell+nospell` option specified.

Did You Mean suggestions for record search require the use of the `Nty=1` URL query parameter. For example:

```
<application>?N=0&Ntk=Description&Ntt=sony&Nty=1
```

Setting `Nty=0` (or omitting the `Nty` parameter) prevents Did You Mean suggestions from being returned. This allows an application to control the generation of suggestions after click-through from a previous suggestion.

Spelling Correction and DYM API methods

There are no modifications that are strictly necessary in the Presentation API code to support spelling correction. However, there are API calls that return information about automatic spelling correction and DYM objects.

Spelling corrected results for both dimension search and record search operations are returned as normal search results.



Note: You can disable spelling correction suggestions (autosuggest), auto-correct suggestions and DYM suggestions on individual queries using the `"spell nospell"` option in `nequery.setDimSearchOpts` parameter of the `ENEQuery` method (Java), or in `nequery.DimSearchOpts` property (.NET). For more information, see the topic on disabling spelling correction on individual queries.

Optionally, applications can display information about automatic spelling corrections or Did You Mean suggestions for dimension or record search operations using the automatically-generated `ESearchReport` objects returned by the MDEX Engine.

For example, consider the following query, which performs two record search operations (a search for *cdd* in the AllText search interface and a search for *sny* in the Manufacturer search interface):

```
<application>?N=0&Ntk=AllText|Manufacturer&Ntt=cdd|sny&Nty=1
```

The Java `Navigation.getESearchReportsComplete()` method and the .NET `Navigation.ESearchReportsComplete` property return a map of search keys to a list of `ESearchReport` objects that provides access to the information listed in the following two tables.

ESearchReport Java method	Returned value
<code>getKey()</code>	AllText
<code>getTerms()</code>	Cdd
<code>getSearchMode()</code>	MatchAll
<code>getMatchedMode()</code>	MatchAll
<code>getNumMatchingResults()</code>	122
<code>getAutoSuggestions().get(0).getTerms()</code>	Cd
<code>getDYMSuggestions().get(0).getTerms()</code>	Ccd

ESearchReport Java method	Returned value
getDYMSuggestions().get(0).getNumMatchingResults()	6
getDYMSuggestions().get(1).getTerms()	Cdp
getDYMSuggestions().get(1).getNumMatchingResults()	7
getKey()	Manufacturer
getTerms()	Sny
getSearchMode()	MatchAll
getMatchedMode()	MatchAll
getNumMatchingResults()	121
getAutoSuggestions().get(0).getTerms()	Sony

ESearchReport .NET property	Returned value
Key	AllText
Terms	Cdd
SearchMode	MatchAll
MatchedMode	MatchAll
NumMatchingResults	122
AutoSuggestions[(0)].Terms	Cd
DYMSuggestions[(0)].Terms	Ccd
DYMSuggestions[(0)].NumMatchingResults	6
DYMSuggestions[(1)].Terms	Cdp
DYMSuggestions[(1)].NumMatchingResults	7
Key	Manufacturer
Terms	Sny
SearchMode	MatchAll
MatchedMode	MatchAll
NumMatchingResults	121
AutoSuggestions[(0)].Terms	Sony

Note that the auto-correct spelling corrections and the explicit Did You Mean suggestions are grouped with related record search operations. (In this case, *cd* is the spelling correction for *cdd* and *sony* is the spelling correction for *sny*.)

Java example of displaying autocorrect messages

```
// 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);
            // Get the List of auto-correct values
            List autoCorrectList = srchrpt.getAutoSuggestions();
            // If the list contains Auto Suggestion objects,
            // print the value of the first corrected term
            if (autoCorrectList.size() > 0) {
                // Get the Auto Suggestion object
                ESearchAutoSuggestion autoSug = (ESearchAutoSuggestion)autoCorrectList.get(0);
                // Display autocorrect message
                %>Corrected to <%= autoSug.getTerms() %>
            }
        }
    }
}

```

.NET example of displaying autocorrect messages

```

// 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.ContainsKey(searchKey)) {
        // Get the first Search Report object
        IDictionaryEnumerator ide = recSrchrpts.GetEnumerator();
        ide.MoveNext();
        ESearchReport searchReport = (ESearchReport)ide.Value;
        // Get the List of auto-correct objects
        IList autoCorrectList = searchReport.AutoSuggestions;
        // If the list contains Auto Suggestion objects,
        // print the value of the first corrected term
        if (autoCorrectList.Count > 0) {
            // Get the Auto Suggestion object
            ESearchAutoSuggestion autoSug = (ESearchAutoSuggestion)autoCorrectList[0];

            // Display autocorrect message
            %>Corrected to <%= autoSug.Terms %>
        }
    }
}

```

Java example of creating links for Did You Mean suggestions

```

// Get the Map of ESearchReport objects
Map dymRecSrchrpts = nav.getESearchReports();
if (dymRecSrchrpts.size() > 0) {
    // Get the user's search key
    String searchKey = request.getParameter("Ntk");
    if (searchKey != null) {
        if (dymRecSrchrpts.containsKey(searchKey)) {
            // Get the ERecSearchReport for the user's search key
            ESearchReport searchReport = (ESearchReport) dymRecSrchrpts.get(searchKey);

            // Get the List of Did You Mean objects
            List dymList = searchReport.getDYMSuggestions();
            // If the list contains Did You Mean objects, provide a
            // link to search on the first suggested term

```

```

        if (dymList.size() > 0) {
            // Get the Did You Mean object
            ESearchDYMSuggestion dymSug = (ESearchDYMSuggestion)dymList.get(0);
            String sug_val = dymSug.getTerms();
            if (sug_val != null){
                // Display didyoumean link
                %>Did You Mean: <%= sug_val %>
            }
        }
    }
}

```

.NET example of creating links for Did You Mean suggestions

```

dd
// Get the Dictionary of ESearchReport objects
IDictionary dymRecSrchrpts = nav.ESearchReports;
// Get the user's search key
String dymSearchKey = Request.QueryString["Ntk"];
if (dymSearchKey != null) {
    if (dymRecSrchrpts.Contains(dymSearchKey)) {
        // Get the first Search Report object
        IDictionaryEnumerator ide = dymRecSrchrpts.GetEnumerator();
        ide.MoveNext();
        ESearchReport searchReport = (ESearchReport)ide.Value;
        // Get the List of DYM objects
        IList dymList = searchReport.DYMSuggestions;
        // If the list contains DYM objects, print the value
        // of the first suggested term
        if (dymList.Count > 0) {
            // Get the DYM object
            ESearchDYMSuggestion dymSug = (ESearchDYMSuggestion)dymList[0];
            String sug_val = dymSug.Terms;
            String sug_num = dymSug.NumMatchingResults.ToString();
            // Display DYM message
            if (sug_val != null){
                %>Did You Mean: <%= sug_val %>
            }
        }
    }
}
}

```

dgraph tuning flags for Spelling Correction and Did You Mean

The MDEX Engine provides a number of advanced tuning options that allow you to achieve various performance and behavioral effects in the Spelling Correction feature.

An explanation of these tuning parameters relies on an understanding of the internal process used by the MDEX Engine to generate spelling suggestions.

At a high level, the spelling engine performs the following steps to generate alternate spelling suggestions for a given query:

1. If the user query generates more than a certain number of hits, then do not generate suggestions. This threshold number of hits is the `hthresh` parameter.

2. For each word in the user's search query, compute the N most similar words in the data set from a spelling similarity perspective (N words are computed for each user query term). This number is set internally and is not user-configurable.
3. For each word in the user's search query, from the set of N most similar spelling words determined in step 2, pick the M most likely replacement words (where $M \leq N$), based on a scoring process that combines factors such as spelling similarity and word frequency (number of hits). This narrows the set of possible spelling replacements for each user query word to M . This number is set internally and is not user-configurable.
4. Consider combinations of these replacements for the user query words, limiting consideration to only combinations that gain more than a threshold percentage number of hits relative to the user's original query, without reducing the number of query terms matched. This gain threshold percent is set internally and is not user-configurable.
5. Scoring each such alternate query using a combination of factors such as spelling similarity of words used and the number of hits generated by the query, select the K best queries and use them as suggestions. K (the maximum number of replacement queries to generate) is called the `nsug` parameter.
6. Finally, consider alternate queries computed by changing the word divisions in the user's query, with the word-break analysis feature. Using the same scoring technique and limits on suggested queries described in steps 4 and 5, include alternate word-break queries in the final suggestion set.

User-configurable parameters

The following table summarizes the user-configurable parameters described in the above process:

Parameter	Description
<code>hthresh</code>	Specifies the threshold number of hits at or above which spelling suggestions will not be generated. That is, above this threshold, the spelling feature is disabled, allowing correctly spelled queries to return only actual (non-spell-corrected) results. Results that don't match all query terms don't count toward the <code>hthresh</code> threshold. For example, if you have a 1000 results which are all partial matches (they match only a proper subset of the query terms) and <code>hthresh</code> is set to 1, then spelling correction will still engage because you have 0 full matches. Note that the case where results only match a proper subset of the query terms can only occur when the match mode is set appropriately to allow such partial matches (<code>matchany</code> , <code>matchpartial</code> , <code>matchpartialmax</code> , and so on).
<code>nsug</code>	Specifies the maximum number of alternate spelling queries to generate for a single user search query.
<code>sthresh</code>	Specifies the threshold spelling similarity score for words considered by the spelling correction engine. Scores are based on a scale where 100 points corresponds approximately to an edit distance of 1. The cost associated with correcting a query term is higher if the term corresponds to an actual word in the data. That is, correcting <i>modem</i> to <i>model</i> is considered a more significant edit than correcting <i>modek</i> to <i>model</i> , if <i>modem</i> occurs as a word in the data but <i>modek</i> does not. The threshold applies to the entire query; for multi-word queries, the edit scores associated with correcting multiple words are added together, and the sum cannot exceed the threshold. For details about the interaction of the <code>--spl_sthresh</code> and <code>--dym_sthresh</code> settings, see the section below.
<code>glom</code>	Specifies that cross-property matches are considered valid when scoring replacement queries. By default, hits that result from applying some queries terms to one text field on a record and other terms to a different text field are not counted. In some cases, these results are desirable and should be considered when computing spelling suggestions.

Parameter	Description
nobrk	Specifies that the word-break analysis portion of the spelling correction process described above is disabled.

Each of these parameters can be specified independently for each of the spelling correction features:

- For record and dimension search auto-correct, the `--spl_` prefix is used (for example, `--spl_nsug`). The flag `--spl` by itself enables auto-suggest spelling corrections for record search and dimension search.
- For explicit suggestions, the `--dym_` prefix is used (for example, `--dym_nsug`). The flag `--dym` by itself enables Did You Mean explicit query spelling suggestions for record search queries.
- For parameters that apply to all of the above, the `--spell_` prefix is used.

For additional configuration of the word-break analysis feature (beyond disabling it with `--spell_nobrk`), use the following `--wb_` flags:

- `--wb_noibrk` disables the insertion of breaks in word-break analysis.
- `--wb_norbrk` disables the removal of breaks in word-break analysis.
- `--wb_maxbrks` specifies the maximum number of word breaks to be added to or removed from a query. The default is one.
- `--wb_minbrklen` specifies the minimum length of a new term created by word-break analysis. The default is two.

Summary of the Spelling Correction and Did You Mean options

The following table summarizes the complete set of options:

Feature	Available dgraph flags
Record Search and Dimension Search	<code>--spl</code> , <code>--spl_hthresh</code> , <code>--spl_nsug</code> , <code>--spl_sthresh</code>
Did You Mean	<code>--dym</code> , <code>--dym_hthresh</code> , <code>--dym_nsug</code> , <code>--dym_sthresh</code>
Record Search and Did You Mean	<code>--spell_glom</code> Note that the <code>--spell_glom</code> option does not apply to dimension search, because cross-property matching is inherently incompatible with the dimension search feature. Dimension search matches always represent a single dimension value.
Record Search, Dimension Search, and Did You Mean	<code>--spell_nobrk</code> , <code>--wb_noibrk</code> , <code>--wb_norbrk</code> , <code>--wb_maxbrks</code> , <code>--wb_minbrklen</code>



Note: Terms that appear in the corpus more than $2 \times \max(\text{spl_hthresh}, \text{dym_hthresh})$ are never corrected, because such terms are unlikely to be misspelled.

Interaction of `--spl_sthresh` and `--dym_sthresh`

The `--spl_sthresh` and `--dym_sthresh` flags are used to set the threshold spelling correction score for words used by the auto-correct or DYM engines, respectively. This is the threshold at which the engine will consider the suggestion. Words that qualify have a score below a given threshold. The higher the edit distance for a term, the higher the score. The default for `--spl_sthresh` is 125, and the default for `--dym_sthresh` is 175.

Based on these default values, if a particular suggestion has a score of 100, it can be used for either DYM or auto-correct, and if it has a score of 200, it is not used by either. If the suggested word has a score better (that is, lower) than the default DYM threshold of 175, but not good enough (that is, higher) than the default auto-correct threshold of 125, it qualifies only for DYM.

A higher value for either of these settings generally results in more suggestions being generated for a misspelled word. In an example query against the sample wine data, changing the `--dym_stresh` value from 175 to 225 increased the number of terms considered for DYM from one to ten. However, raising scores too high could result in a lot of noise. That is to say, it is generally a good thing if nonsense strings used as search terms receive neither auto-correct nor DYM suggestions.

How dimension search treats number of results

Dimension search results may vary if spelling correction is performed.

An important note applies to the options and behavior associated with dimension search spelling correction: in situations where the number of results is evaluated by an option or in the scoring of words or queries performed by the spelling engine, dimension search uses an alternate definition of number of results. Instead of using the simple number of hits returned to the user as this value (which is perfectly reasonable in the case of record search), dimension search instead uses the number of records associated with the set of dimension value search results computed for a given query.

In other words, dimension search follows an additional level of indirection to weight the dimension value results computed by spelling suggestion queries according to the number of records that these dimension values would lead to if selected in a navigation query. This alternate definition of number of results allows consistent behavior between spelling corrections computed for dimension and record search operations when given the same query terms.

Troubleshooting Spelling Correction and Did You Mean

This topic provides some corrective solutions for spelling correction problems.

If spell-corrected results are not returned for words with expected spell-corrected options in the data, check the potential problems described in this topic.

When debugging spelling behavior, pay close attention to the errors of the dgraph on startup, at which point problems in spelling configuration are typically reported.

Did You Mean and stop words interaction

Did You Mean can in some cases correct a word to one on the stop words list.

Did You Mean and query configuration

If a record search query produces Did You Mean options, each DYM query has the same configuration as the initial record search query. For example, if the record search query had **Allow cross field matches** set to **On Failure**, then the DYM query also runs with cross field matching set to **On Failure**.

Interaction of Aspell, Espell and DYM

This section is relevant to you if you are using `aspell_AND_espell` mode with DYM enabled. It describes the interaction of both spelling modes with DYM and explains why in some instances, suggestions that should have been found by Aspell or Espell are not considered by DYM. In other words, you may observe that in some

instances user-entered words with misspellings in them do not return DYM suggestions, if the `aspell_AND_espell` mode is used.

The following statements describe the reasons behind this behavior in more detail:

- Both spelling modes, Aspell or Espell, work by generating a list of suggestion results. These suggestions are weighted based on the lowest score, according to a scoring algorithm.
- Aspell and Espell generate scores based on different scoring algorithms (described below in this section).
- When both modes are used, as is the case with `aspell_AND_espell`, DYM uses the union of the scored suggestions provided by each spelling mode, and keeps the top 10 terms from the combined list, based on the lowest scores.
- As a result, some suggestions found by Espell (that could have been relevant) do not pass the scoring criteria in the combined list, and are thus not considered by DYM.
- The following statements discuss how scores are calculated for each of the spelling engines (Aspell and Espell):
 - For information about the GNU Aspell scoring algorithm, see the documentation for this open source product.
 - The Espell scoring algorithm uses the following formula:

```
(85 - num_matching_characters_in_prefix)* edit_distance
```

The parameter `edit_distance` specifies a regular Levenshtein distance (see the Internet for more information). In `edit_distance`, character swaps, insertions and deletions count as an edit distance of 1.

The `num_matching_characters_in_prefix` is a number of all matching characters before a mismatch occurs. For example, for the term "java", this number is 2 (matching "j" and "a"); for the term "jsva", this number is 1 (matching only "j").

The directory specified for the `--spellpath` flag

- The directory specified in the `--spellpath` flag to the MDEX Engine must be an absolute path. If a relative path is used, an error message is sent to the standard error output in the format:

```
[Warning] OptiSpell couldn't open pwli file
"<--spell param>/<db_prefix>-aspell.pwli"
'Permission denied'
```

- The directory specified for the `--spellpath` flag must either be writable or already contain a valid `.pwli` file that contains an absolute path to the `spell.dat` binary dictionary file. Check the permissions on this directory. If the directory is not writable or does not contain a valid `.pwli` file, an error is issued as in the previous example.

Performance impact for Spelling Correction and Did You Mean

Spelling correction performance is impacted by the size of the dictionary in use.

Spell-corrected keyword searches with many words, in systems with very large dictionaries, can take a disproportionately long time to process relative to other MDEX Engine requests. Those searches can cause requests that immediately follow such a search to wait while the spelling recommendations are being sought and considered.

Because of this, it is important to carefully analyze the performance of the system together with application requirements prior to production application deployment.

Consider also whether performance could be improved if you disable spelling correction on individual queries. For information about disabling spelling correction on individual queries, see the topic in this guide.

Related Links

[Disabling spelling correction on individual queries](#) on page 286

This topic describes how to disable spelling correction and DYM suggestions on individual queries.

Compiling the Aspell dictionary

The Aspell dictionary must be compiled before it can be used by the MDEX Engine.

The Aspell dictionary is automatically compiled at index time, and requires no further processing. But if the selected spelling mode includes use of the Aspell module, the Aspell dictionary must be compiled. If you are manually compiling this file, perform this step after indexing but before starting the MDEX Engine.

Compilation transforms the text-based dictionary into a binary dictionary file suitable for use by the Aspell module in the MDEX Engine. This indexed form of the dictionary is contained in a file with a name of the form `<dbPath>-aspell.spelldat`.

Use one of the following ways to compile the dictionary file:

- Automatically, by running the `admin?op=updateaspell` administrative operation. For information about this operation, see the topic in this section.
- Manually, by running the `dgwordlist` utility script.
- Automatically, by letting the Oracle Commerce Application Controller create them implicitly in the Dgidx component.

Related Links

[About the admin?op=updateaspell operation](#) on page 289

The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.

[Compiling the dictionary manually](#) on page 300

The `dgwordlist` utility script is provided to compile the Aspell dictionary.

[Compiling the dictionary with EAC](#) on page 301

The Dgidx component contains a `run-aspell` setting that specifies Aspell as the spelling correction mode for the implementation.

Compiling the dictionary manually

The `dgwordlist` utility script is provided to compile the Aspell dictionary.

To manually compile the text-based `worddat` dictionary into the binary `spelldat` dictionary, you must use the utility script `dgwordlist` (on UNIX; on Windows, it is `dgwordlist.exe`).

The usage for `dgwordlist` is:

```
dgwordlist [--out <output_dir>] [--aspell <aspell_location>]
           [--datfiles <aspell_dat_files_location>] [--help]
           [--version] <dbPath>
```

Argument for dgwordlist	Description
--out	Specifies the directory where the resulting binary <code>spell.dat</code> dictionary file is placed. If not specified, this defaults to the same directory where the input index files reside (<code><dbPath></code>).
--aspell	Deprecated. If you specify this flag, it is ignored. The <code>dgwordlist</code> utility no longer needs to know the location of the Aspell dictionary indexing program. In previous releases, this flag specified the location of Aspell. This parameter could also be omitted if <code>aspell</code> (or <code>aspell.exe</code> on Windows) was in the current path.
--datfiles	Specifies the input directory location containing the spelling support files. These support files contain information such as language and character set configuration (these files end with <code>.map</code> or <code>.dat</code> extensions). If not specified, this defaults to the same directory where the input index files reside (<code><dbPath></code>).
<dbPath>	Specifies a prefix path to the input index data, including the text-based <code>word.dat</code> dictionary file. This should match the index prefix given to <code>Dgidx</code> .
--version	Prints the version information and exits.
--help	Prints the command usage and exits.

In typical operational configurations, the binary `spell.dat` dictionary file created by `dgwordlist` and the `.map` and/or `.dat` files located in the `--datfiles` directory are placed in the same directory as the indexed data prior to starting the MDEX Engine.

Example of running dgwordlist

```
$ cp /usr/local/endeca/6.1.3/lib/aspell/* ./final_output
$ /usr/local/endeca/6.1.3/bin/dgwordlist
/usr/local/endeca/6.1.3/bin/aspell ./final_output/wine
Creating "./final_output/wine-aspell.spell.dat"
```

Related Links

[About the admin?op=updateaspell operation](#) on page 289

The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.

Compiling the dictionary with EAC

The `Dgidx` component contains a `run-aspell` setting that specifies Aspell as the spelling correction mode for the implementation.

The default value of `run-aspell` is `true`; that is, it compiles the dictionary file for you by default and copies the Aspell files into its output directory, where the `dgraph` can access them.

If you do not want the spelling dictionary to be created, you must set `run-aspell` to `false` in the `Dgidx` component. You can change this setting either by directly editing your Endeca Application Controller provisioning file, or by editing the arguments for the `Dgidx` component located in Oracle Commerce Workbench on the EAC Administration Console page.

Related Links

[About the `admin?op=updateaspell` operation](#) on page 289

The `admin?op=updateaspell` administrative operation lets you rebuild the aspell dictionary for spelling correction from the data corpus without stopping and restarting the MDEX Engine.

About word-break analysis

Word-break analysis allows the Spelling Correction feature to consider alternate queries computed by changing the word divisions in the user's query.

For example, if the query is *Back Street Boys*, word-break analysis could instruct the MDEX Engine to consider the alternate *Backstreet Boys*.

When word-break analysis is applied to a query, it requires that the substrings that the term is broken up into appear in the data in succession.

For example, starting with the query *box17*, word-break analysis would find *box 17*, as well as *box-17*, assuming that the hyphen (-) has not been specified as a search character. However, it would not find *17 old boxes*, because the target terms do not appear in order.

Disabling word-break analysis

You can disable the word-break analysis feature with a dgraph flag.

Word-break analysis is enabled by default, as are its associated parameters. You can disable word-break analysis by starting the MDEX Engine with the `--spell_nobrk` flag.

Word-break analysis configuration parameters

You configure the details of word-break analysis with four dgraph flags.

Keep in mind that word-break analysis must be enabled in order for these flags to have any effect.

The four dgraph flags are as follows:

- To control the maximum number of word breaks to be added to or removed from a query, use the `--wb_maxbrks` flag. The default is one.
- To specify the minimum length for a new term created by word-break analysis, use the `--wb_minbrklen` flag. The default is two.
- To disable the ability of word-break analysis to remove breaks from the original term, use the `--wb_norbrk` flag.
- To disable the ability of word-break analysis to add breaks to the original term, use the `--wb_noibrk` flag.

Performance impact of word-break analysis

The performance impact of word-break analysis can be considerable, depending on your data.

Seemingly small deviations from default values (such as increasing the value of `--wb_maxbrks` from one to two) can have a significant impact, because they greatly increase the workload on the MDEX Engine. Oracle suggests that you tune this feature carefully and test its impact thoroughly before exposing it in a production environment.

Chapter 32

Stemming and Thesaurus

This section describes how to implement the Stemming and Thesaurus features of the Oracle Commerce MDEX Engine.

Overview of Stemming and Thesaurus

The Oracle Commerce MDEX Engine supports Stemming and Thesaurus features that allow keyword search queries to match text containing alternate forms of the query terms or phrases.

The definitions of these features are as follows:

- The Stemming feature allows the system to consider alternate forms of individual words as equivalent for the purpose of search query matching. For example, it is often desirable for singular nouns to match their plural equivalents in the searchable text, and vice versa.
- The Thesaurus feature allows the system to return matches for related concepts to words or phrases contained in user queries. For example, a thesaurus entry may allow searches for *Mark Twain* to match text containing the phrase *Samuel Clemens*.

Both the Thesaurus and Stemming features rely on defining equivalent textual forms that are used to match user queries to searchable text data. Because these features are based on similar concepts, and because they are typically configured to operate in conjunction to achieve desired query matching effects, both features and their interactions are discussed in one section.

About the Stemming feature

The Stemming feature broadens search results to include root words and variants of root words.

Stemming is intended to allow words with a common root form (such as the singular and plural forms of nouns) to be considered interchangeable in search operations. For example, search results for the word *shirt* will include the derivation *shirts*, while a search for *shirts* will also include its word root *shirt*.

Stemming equivalences are defined among single words. For example, stemming is used to produce an equivalence between the words *automobile* and *automobiles* (because the first word is the stem form of the second), but not to define an equivalence between the words *vehicle* and *automobile* (this type of concept-level mapping is done via the Thesaurus feature).

Stemming equivalences are strictly two-way (that is, all-to-all). For example, if there is a stemming entry for the word *truck*, then searches for *truck* will always return matches for both the singular form (*truck*) and its

plural form (*trucks*), and searches for *trucks* will also return matches for *truck*. In contrast, the Thesaurus feature supports one-way mappings in addition to two-way mappings.

Language support for stemming

The MDEX Engine supports stemming for multiple languages. For details about stemming and non-English data, refer to the *Oracle Commerce Internationalization Guide*.

Types of stemming matches and sort order

Stemming can produce one of three match types.

If stemming is enabled, a search on a given term (*T*) will produce one or more of these results:

- Literal matches: Any occurrence of *T* always produce a match.
- Stem form matches: Matches occur on the stem form of *T* (assuming that *T* is not a stem form). For example, if *T* is *children*, then *child* (the stem form) also matches.
- Inflected form matches: Matches occur on all inflected forms of the stem form of *T*. For example, if *T* is the verb *ran* (as in *Jane ran in the Boston Marathon*), then matches include the stem form (*run*) and inflected forms (such as *runs* and *running*). (Note that although this example is in English, stemming for inflected verb forms is not supported for English; see below for support details).

The order of the returned results depends on the sorting configuration:

- If relevance ranking is enabled and the Interpreted (interp) module is used, literal matches will always have higher priority than stem form and inflected form matches.
- If relevance ranking is not enabled but you have set a record sort order, the results will come back in that sort order.
- If relevance ranking is not enabled and there is no record sort order, the order of the results is completely arbitrary.



Note: The type of stemming described in this chapter is tokenized using a Latin-1 analyzer and is referred to as static stemming. If you are using non-English stemming, the stemming implementation may use either the Latin-1 analyzer or the Oracle Language Technology analyzer to tokenize source data. This choice results in slightly different stemming behavior. For mo.

Enabling stemming

Stemming is enabled in Developer Studio for a subset of supported languages listed in the Stemming editor.

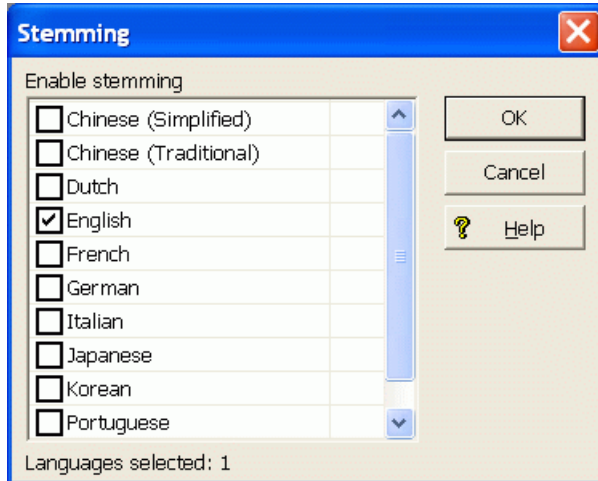
Additional ISO-639 languages are supported but not listed in the Stemming editor. For details about enabling stemming with other international languages, refer to the *Oracle Commerce Internationalization Guide*.



Important: Configuring stemming in Developer Studio overwrites any custom stemming dictionaries you may have created and specified for selected languages in your application, as well as overwriting any settings passed in to the dgraph through the `--lang` flag. You should not use the Developer Studio Stemming editor in combination with manually-configured settings.

To enable stemming:

1. Open the project in Developer Studio.
2. In the Project Explorer, expand **Search Configuration**.
3. Double-click **Stemming** to display the **Stemming** editor.



4. Check one or more of the language check boxes on the list.
5. Click **OK**.

To disable stemming, use the above procedure, but uncheck the languages for which you do not want stemming.

Supplementing the default static stemming dictionaries

You can supplement the default stemming dictionaries by specifying the `--stemming-updates` flag to Dgidx and providing an XML file of custom stemming changes. The stemming update file may include additions and deletions. Dgidx processes the file by adding and deleting entries in the stemming dictionary file.

The default stemming dictionary files are stored in `Endeca\MDEX\version\conf\stemming` (on Windows) and `usr/local/endeca/MDEX/version/conf/stemming` (on UNIX).

For most supported languages, the stemming directory contains two types of stemming dictionaries per language. One dictionary (`<RFC 3066 Language Code>_word_forms_collection.xml`) contains stemming entries that support accented characters for the particular `<RFC 3066 Language Code>`.

The other dictionary (`<RFC 3066 Language Code>-x-folded_word_forms_collection.xml`) contains stemming entries where all accented characters have been folded down (removed) for the particular `<language_code>`. If present, this is the stemming dictionary that is used if you specify `--diacritic-folding`. For details about how to map accented characters to unaccented characters, refer to the *Oracle Commerce Internationalization Guide*.

Adding entries to a stemming dictionary

To illustrate the XML you add the stemming update file, it is helpful to treat each operation (adding and deleting) as a separate use-case and show the required XML for each operation.

You specify stemming entries to add within a `<ADD_WORD_FORMS>` element and its sub-element `<WORD_FORMS_COLLECTION>`. For example, the following XML adds `apple` and its stemmed variant `apples` to the stemming dictionary:

```
<!DOCTYPE WORD_FORMS_COLLECTION_UPDATES SYSTEM "word_forms_collection_updates.dtd">
<WORD_FORMS_COLLECTION_UPDATES>
  <ADD_WORD_FORMS>
    <WORD_FORMS_COLLECTION>
      <WORD_FORMS>
        <WORD_FORM>apple</WORD_FORM>
        <WORD_FORM>apples</WORD_FORM>
      </WORD_FORMS>
    </WORD_FORMS_COLLECTION>
  </ADD_WORD_FORMS>
</WORD_FORMS_COLLECTION_UPDATES>
```

```

        </WORD_FORMS>
    </WORD_FORMS_COLLECTION>
</ADD_WORD_FORMS>
</WORD_FORMS_COLLECTION_UPDATES>

```

Deleting entries from a stemming dictionary

You specify stemming entries to delete in a `<REMOVE_WORD_FORMS_KEYS>` element. All word forms that correspond to that key are deleted. For example, the following XML deletes `aalborg` and all of its stemmed variants from the stemming dictionary:

```

<!DOCTYPE WORD_FORMS_COLLECTION_UPDATES SYSTEM "word_forms_collection_updates.dtd">
<WORD_FORMS_COLLECTION_UPDATES>
  <REMOVE_WORD_FORMS_KEYS>
    <WORD_FORM>aalborg</WORD_FORM>
  </REMOVE_WORD_FORMS_KEYS>
</WORD_FORMS_COLLECTION_UPDATES>

```

Combining deletes and adds

You can also specify a combination of deletes and then adds. Deletes are processed first and then adds are processed. For example, the following XML removes `aachen` and then adds it and several stemmed variants of it.

```

<!DOCTYPE WORD_FORMS_COLLECTION_UPDATES SYSTEM "word_forms_collection_updates.dtd">
<WORD_FORMS_COLLECTION_UPDATES>
  <REMOVE_WORD_FORMS_KEYS>
    <WORD_FORM>aachen</WORD_FORM>
  </REMOVE_WORD_FORMS_KEYS>
  <ADD_WORD_FORMS>
    <WORD_FORMS_COLLECTION>
      <WORD_FORMS>
        <WORD_FORM>aachen</WORD_FORM>
        <WORD_FORM>aachens</WORD_FORM>
        <WORD_FORM>aachenes</WORD_FORM>
      </WORD_FORMS>
    </WORD_FORMS_COLLECTION>
  </ADD_WORD_FORMS>
</WORD_FORMS_COLLECTION_UPDATES>

```

Syntax of the stemming update file name

The syntax of the stemming update file name must be as follows:

```

user_specified.<RFC 3066 Language Code>.xml

```

where

- *user_specified* is any string that is relevant to your application or stemming dictionary, for example `myAppStemmingChanges`.
- *RFC 3066 Language Code* is a two-character language code, of the stemming dictionary you want to update, for example, `en` or `en-us`. See ISO 639-1 for the full list of two-character codes and RFC 3066 for the two-character sub tag for region.

Processing the update file

To process the stemming update file, you specify the `--stemming-updates` flag to `Dgidx` and specify the XML file of stemming updates.

For example:

```
dgidx --stemming-updates myAppStemmingChanges.en.xml
```

Conflicts during updates

When Dgidx merges the changes in an update file into the stemming dictionary, there may be conflicts in cases where the variant for one root in the stemming dictionary is the same as a variant for another root in the update file. Any duplicate variants of different root words constitute a conflict.

In this case, Dgidx throws a warning about conflicting variants and rejects the variant that was specified in the update file.

Adding a custom static stemming dictionary

If your application requires a stemming language that is not available in the Stemming editor of Developer Studio, you can create and add a custom stemming dictionary. A custom stemming dictionary is available in addition to any stemming selections you may have enabled in Developer Studio. For example, you can enable English and Dutch, and then add an additional custom stemming dictionary for Swahili.

Although you can create any number of custom stemming dictionaries, only one custom stemming dictionary can be loaded into the MDEX Engine. You indicate which custom stemming dictionary to load with the `--lang` flag to Dgidx.

To add a custom stemming dictionary:

1. Create a custom dictionary file with stemming entries. For sample XML, see the XML schema of any default stemming dictionary stored in `<install path>\MDEX\<version>\conf\stemming`.

For example, this simplified file contains one term and one stemmed variant:

```
<?xml version="1.0"?>

<!ELEMENT WORD_FORMS_COLLECTION_UPDATES (COMMENT?, RE-
MOVE_WORD_FORMS_KEYS*,ADD_WORD_FORMS*)>

<WORD_FORMS_COLLECTION>

<WORD_FORMS>

<WORD_FORM>swahiliterm</WORD_FORM>

<WORD_FORM>swahiliterms</WORD_FORM>

</WORD_FORMS>

</WORD_FORMS_COLLECTION>
```

2. When you have created the custom stemming dictionary, save the XML file with one of the following name formats:

- If the dictionary contains *unaccented* characters and you use the Dgidx flag `--diacritic-folding`, save the file as `<RFC 3066 Language Code>-x-folded_word_forms_collection.xml`.
- If the dictionary contains *accented* characters and you are *not* using the Dgidx flag `--diacritic-folding`, save the file as `<RFC 3066 Language Code>_word_forms_collection.xml`.

For example, the XML above would be saved as `sw_word_forms_collection.xml` where `sw` is the ISO639-1 language code for Swahili.

3. Place the XML file in `<install path>\MDEX\<version>\conf\stemming\custom`.

- Specify the `--lang` flag to Dgidx with a `<lang id>` argument that matches the language code of the custom stemming dictionary file.
In the example above that uses a Swahili (`sw`) dictionary, you would specify:

```
dgidx --lang sw
```

Replacing a default static stemming dictionary with a custom stemming dictionary

Rather than supplement a default stemming dictionary, you may chose to entirely replace a default stemming dictionary with a custom a stemming dictionary.

To replace a default stemming dictionary with a custom stemming dictionary:

- Create a custom dictionary file with stemming entries. For example XML, see the XML schema of any default stemming dictionary stored in `<install path>\MDEX\<version>\conf\stemming`. For example, this simplified English stemming dictionary contains one term and one stemmed variant:

```
<?xml version="1.0"?>

<!ELEMENT WORD_FORMS_COLLECTION_UPDATES (COMMENT?, RE-
MOVE_WORD_FORMS_KEYS*,ADD_WORD_FORMS*)>

<WORD_FORMS_COLLECTION>

<WORD_FORMS>

<WORD_FORM>car</WORD_FORM>

<WORD_FORM>cars</WORD_FORM>

</WORD_FORMS>

</WORD_FORMS_COLLECTION>
```

- When you have created the custom stemming dictionary, save the XML file with one of the following name formats:
 - If the dictionary contains *unaccented* characters and you use the Dgidx flag `--diacritic-folding`, save the file as `<RFC 3066 Language Code>-x-folded_word_forms_collection.xml`.
 - If the dictionary contains *accented* characters and you are *not* using the Dgidx flag `--diacritic-folding`, save the file as `<RFC 3066 Language Code>_word_forms_collection.xml`.

For example, the XML above would be saved as `en_word_forms_collection.xml` where `en` is the ISO639-1 code for English.

- Place the XML file in `<install path>\MDEX\<version>\conf\stemming\custom`.
- Open your project in Developer Studio.
- In the Project Explorer, expand **Search Configuration**.
- Double-click **Stemming** to display the Stemming editor.
- Un-check the language you want to replace.
- Click **OK**.
- Specify the `--lang` flag to Dgidx with a `<lang id>` argument that matches the language code of the custom stemming dictionary file.

In the example above that uses an English (en) dictionary, you would specify:

```
dgidx --lang en
```

About the Thesaurus feature

The Thesaurus feature allows you to configure rules for matching queries to text containing equivalent words or concepts.

The thesaurus is intended for specifying concept-level mappings between words and phrases. Even a modest number of well-thought-out thesaurus entries can greatly improve your users' search experience.

The Thesaurus feature is a higher level than the Stemming feature, because thesaurus matching and query expansion respects stemming equivalences, whereas the stemming module is unaware of thesaurus equivalences.

For example, if you define a thesaurus entry mapping the words *automobile* and *car*, and there is a stemming equivalence between *car* and *cars*, then a search for *automobile* will return matches for *automobile*, *car*, and *cars*. The same results will also be returned for the queries *car* and *cars*.

The thesaurus supports specifying multi-word equivalences. For example, an equivalence might specify that the phrase *Mark Twain* is interchangeable with the phrase *Samuel Clemens*. It is also possible to mix the number of words in the phrase-forms for a single equivalence. For example, you can specify that *wine opener* is equivalent to *corkscrew*.

Multi-word equivalences are matched on a phrase basis. For example, if a thesaurus equivalence between *wine opener* and *corkscrew* is defined, then a search for *corkscrew* will match the text *stainless steel wine opener*, but will not match the text *an effective opener for wine casks*.

Thesaurus equivalences can be either one-way or two-way:

- One-way mapping specifies only one direction of equivalence. That is, one "From" term is mapped to one or more "To" terms, but none of the "To" terms are mapped to the "From" term. Only one "From" term can be specified.

For example, assume you define a one-way mapping from the phrase *red wine* to the phrases *merlot* and *cabernet sauvignon*. This one-way mapping ensures that a search for *red wine* also returns any matches containing the more specific terms *merlot* or *cabernet sauvignon*. But you avoid returning matches for the more general phrase *red wine* when the user specifically searches for either *merlot* or *cabernet sauvignon*.

- Two-way (or all-to-all) mapping means that the direction of a word mapping is equivalent between the words. For example, a two-way mapping between *stove*, *range*, and *oven* means that a search for one of these words will return all results matching any of these words (that is, the mapping marks the forms as strictly interchangeable).

When you define a two-way mapping, you do not specify a "From" term. Instead, you specify two or more "To" terms.

Unlike the Stemming module, the Thesaurus feature lets you define multiple equivalences for a single word or phrase. These multiple equivalences are considered independent and non-transitive.

For example, we might define one equivalence between *football* and *NFL*, and another between *football* and *soccer*. With these two equivalences, a search for *NFL* will return hits for *NFL* and hits for *football*, a search for *soccer* will return hits for *soccer* and *football*, and a search for *football* will return all of the hits for *football*, *NFL*, and *soccer*. However, searches for *NFL* will not return hits for *soccer* (and vice versa).

This non-transitive nature of the thesaurus is useful for defining equivalences containing ambiguous terms such as *football*. The word *football* is sometimes used interchangeably with *soccer*, but in other cases *football* refers to American football, which is played professionally in the NFL. In other words, the term *football* is ambiguous.

When you define equivalences for ambiguous terms, you do not want their specific meanings to overlap into one another. People searching for *soccer* do not want hits for *NFL*, but they may want at least some of the hits associated with the more general term *football*.

Thesaurus entries are essentially used to produce alternate forms of the user query, which in turn are used to produce additional query results. As a rule, the MDEX Engine will expand the user query into the maximum possible set of alternate queries based on the available thesaurus entries.

This behavior is particularly important in the presence of overlapping thesaurus forms. For example, suppose that you define an equivalence between *red wine* and *vino rosso*, and a second equivalence between *wine opener* and *corkscrew*. The query *red wine opener* might match the thesaurus entries in two different ways: *red wine* could be mapped to *vino rosso* based on the first entry; or *wine opener* could be mapped to *corkscrew* based on the second entry.

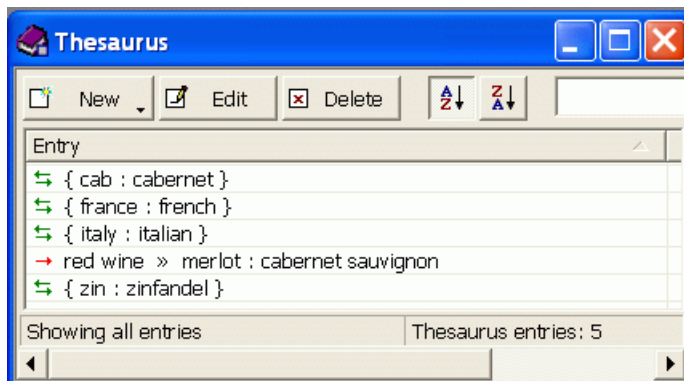
Using the maximal-expansion rule, this issue is resolved by expanding to all possible queries. In other words, the MDEX Engine returns hits for all of the queries: *red wine opener*, *vino rosso opener*, and *red corkscrew*.

Adding thesaurus entries

Thesaurus entries are added in Developer Studio.

To add a one-way or two-way thesaurus entry:

1. Open the project in Developer Studio.
2. In the Project Explorer, expand **Search Configuration**.
3. Double-click **Thesaurus** to display the Thesaurus view.



4. Click **New** and select either **One Way** or **Two Way**.
5. Configure the entry in the Thesaurus Entry dialog:
 - For a one-way entry: type in one term in the "From" field, add one or more "To" terms, and click **OK**.
 - For a two-way entry: add two or more "To" terms and click **OK**.
6. Save the project.

The Thesaurus view also allows you to modify and delete existing thesaurus entries.

Troubleshooting the thesaurus

The following thesaurus clean-up rules should be observed to avoid performance problems related to expensive and non-useful thesaurus search query expansions.

- Do not create a two-way thesaurus entry for a word with multiple meanings. For example, *khaki* can refer to a color as well as to a style of pants. If you create a two-way thesaurus entry for `khaki = pants`, then a user's search for *khaki towels* could return irrelevant results for *pants*.
- Do not create a two-way thesaurus entry between a general and several more-specific terms, such as:

```
top = shirt = sweater = vest
```

This increases the number of results the user has to go through while reducing the overall accuracy of the items returned. In this instance, better results are attained by creating individual one-way thesaurus entries between the general term *top* and each of the more-specific terms.

- A thesaurus entry should never include a term that is a substring of another term in the entry.

For example, consider the two-way equivalency:

```
Adam and Eve = Eve
```

If users type *Eve*, they get results for *Eve* or *(Adam and Eve)* (that is, the same results they would have gotten for *Eve* without the thesaurus). If users type *Adam and Eve*, they get results for *(Adam and Eve)* or *Eve*, causing the *Adam and* part of the query to be ignored.

- Stop words such as *and* or *the* should not be used in single-word thesaurus forms. For example, if *the* has been configured as a stop word, an equivalency between *thee* and *the* is not useful.

You can use stop words in multi-word thesaurus forms, because multi-word thesaurus forms are handled as phrases. In phrases, a stop word is treated as a literal word and not a stop word.

- Avoid multi-word thesaurus forms where single-word forms are appropriate. In particular, avoid multi-word forms that are not phrases that users are likely to type, or to which phrase expansion is likely to provide relevant additional results.

For example, the two-way thesaurus entry:

```
Aethelstan, King Of England (D. 939) = Athelstan, King Of England (D. 939)
```

should be replaced with the single-word form:

```
Aethelstan = Athelstan
```

- Thesaurus forms should not use non-searchable characters. For example, the one-way thesaurus entry:

```
Pikes Peak -> Pike's Peak
```

should be used only if the apostrophe (') is enabled as a search character.

Dgidx and dgraph flags for the Thesaurus

No Dgidx flags are needed to configure the Thesaurus features.

Thesaurus entries are automatically enabled for use during text indexing and during MDEX Engine search query processing. In addition, there is no MDEX Engine configuration necessary to configure thesaurus information.

The `dgraph --thesaurus_cutoff` flag can be used to tune performance associated with thesaurus expansion. By default, this flag is set to 3, meaning that if a search query contains more than 3 words that appear in "From" entries, none of the query terms are expanded.

No Presentation API development is necessary to use the Thesaurus feature.

Interactions with other search features

As core features of the MDEX Engine search subsystem, Stemming and the Thesaurus have interactions with other search features.

The following sections describe the types of interactions between the various search features.

Search characters

The search character set configured for the application dictates the set of available characters for stemming and thesaurus entries. By default, only alphanumeric ASCII characters may be used in stemming and thesaurus entries. Additional punctuation and other special characters may be enabled for use in stemming and thesaurus entries by adding these characters to the search character set.

The MDEX Engine matches user query terms to thesaurus forms using the following rule: all alphanumeric and search characters must match against the stemming and thesaurus forms exactly; other characters in the user search query are treated as word delimiters.

Spelling

Spelling correction is a closely-related feature to stemming and thesaurus functionality, because spelling auto-correction essentially provides an additional mechanism for computing alternate versions of the user query. In the MDEX Engine, spelling is handled as a higher-level feature than stemming and thesaurus. That is, spelling correction considers only the raw form of the user query when producing alternate query forms.

Alternate spell-corrected queries are then subject to all of the normal stemming and thesaurus processing. For example, if the user enters the query *television* and this query is spell-corrected to *television*, the results will also include results for the alternate forms *televisions*, *tv*, and *tv's*.

Note that in some cases, the Thesaurus feature is used as a replacement or in addition to the system's standard spelling correction features. In general, this technique is discouraged. The vast majority of actual misspelled user queries can be handled correctly by the Spelling Correction subsystem. But in some rare cases, the Spelling Correction feature cannot correct a particular misspelled query of interest; in these cases it is common to add a thesaurus entry to handle the correction. If at all possible, such entries should be avoided as they can lead to undesirable feature interactions.

Stop words

Stop words are words configured to be ignored by the MDEX Engine search query engine. A stop word list typically includes words that occur too frequently in the data to be useful (for example, the word *bottle* in a wine data set), as well as words that are too general (such as *clothing* in an apparel-only data set).

If *the* is marked as a stopword, then a query for *the computer* will match to text containing the word *computer*, but possibly missing the word *the*.

Stop words are not currently expanded by the stemming and thesaurus equivalence set. For example, suppose you mark *item* as a stopword and also include a thesaurus equivalence between the words *item* and *items*. This will not automatically mark the word *items* as a stopword; such expansions must be applied manually.

Stop words are respected when matching thesaurus entries to user queries. For example, suppose you define an equivalence between *Muhammad Ali* and *Cassius Clay* and also mark *M* as a stopword (it is not uncommon to mark all or most single letter words as stopwords). In this case, a query for *Cassius M. Clay* would match the thesaurus entry and return results for *Muhammad Ali* as expected.

Phrase search

A phrase search is a search query that contains one or more multi-word phrases enclosed in quotation marks. The words inside phrase-query terms are interpreted strictly literally and are not subject to stemming or thesaurus processing. For example, if you define a thesaurus equivalence between *Jennifer Lopez* and *JLo*, normal (unquoted) searches for *Jennifer Lopez* will also return results for *JLo*, but a quoted phrase search for "*Jennifer Lopez*" will not return the additional *JLo* results.

Relevance Ranking

It is typically desirable to return results for the actual user query ahead of results for stemming and/or thesaurus transformed versions of the query. This type of result ordering is supported by the Relevance Ranking modules. The module that is affected by thesaurus expansion and stemming is **Interp**. The module that is not affected by thesaurus and stemming is **Freq**.

Performance impact of Stemming and Thesaurus

Stemming and thesaurus equivalences generally add little or no time to data processing and indexing, and introduce little space overhead (beyond the space required to store the raw string forms of the equivalences).

In terms of online processing, both features will expand the set of results for typical user queries. While this generally slows search performance (search operations require an amount of time that grows linearly with the number of results), typically these additional results are a required part of the application behavior and cannot be avoided.

The overhead involved in matching the user query to thesaurus and stemming forms is generally low, but could slow performance in cases where a large thesaurus (tens of thousands of entries) is asked to process long search queries (dozens of terms). Typical applications exhibit neither extremely large thesauri nor very long user search queries.

Because matching for stemming entries is performed on a single-word basis, the cost for stemming-oriented query expansion does not grow with the size of the stemming database or with the length of the query. However, the stemming performance of a specific language is affected by the degree to which the language is inflected. For example, German words are much more inflected than English ones, and a query term can expand into a much larger set of compound words of which its stem is a component.

Automatic Phrasing

This section describes how to implement the Automatic Phrasing feature of the Oracle Commerce MDEX Engine.

About Automatic Phrasing

When an application user provides individual search terms in a query, the Automatic Phrasing feature groups those individual terms into a search phrase and returns query results for the phrase.

Automatic Phrasing is similar to placing quotation marks around search terms before submitting them in a query. For example *"my search terms"* is the phrased version of the query *my search terms*. However, Automatic Phrasing removes the need for application users to place quotation marks around search phrases to get phrased results.

The result of Automatic Phrasing is that a Web application can process a more restricted query and therefore return fewer and more focused search results. This feature is available only for record search.

The Automatic Phrasing feature works by:

1. Comparing individual search terms in a query to a list of application-specific search phrases. The list of search phrases are stored in a project's phrase dictionary.
2. Grouping the search terms into search phrases.
3. Returning query results that are either based on the automatically-phrased query, or returning results based on the original unphrased query along with automatically-phrased Did You Mean (DYM) alternatives.

Implementation scenarios

Step 3 above suggests the two typical implementation scenarios to choose from when using Automatic Phrasing:

- Process an automatically-phrased form of the query and suggest the original unphrased query as a DYM alternative.

In this scenario, the Automatic Phrasing feature rewrites the original query's search terms into a phrased query before processing it. If you are also using DYM, you can display the unphrased alternative so the user can opt-out of Automatic Phrasing and select their original query, if desired.

For example, an application user searches a wine catalog for the unquoted terms *low tannin*. The MDEX Engine compares the search terms against the phrase dictionary, finds a phrase entry for "low tannin", and processes the phrased query as *"low tannin"*. The MDEX Engine returns 3 records for the phrased query *"low tannin"* rather than 16 records for the user's original unphrased query *low tannin*. However, the Web

application also presents a "Did you mean low tannin?" option, so the user may opt-out of Automatic Phrasing, if desired.

- Process the original query and suggest an automatically-phrased form of the query as a DYM alternative.

In this scenario, the Automatic Phrasing feature processes the unphrased query as entered and determines if a phrased form of the query exists. If a phrased form is available, the Web application displays an automatically-phrased alternative as a Did You Mean option. The user can opt-in to Automatic Phrasing, if desired.

For example, an application user searches a wine catalog for the unquoted terms *low tannin*. The MDEX Engine returns 16 records for the user's unphrased query *low tannin*. The Web application also presents a *Did you mean "low tannin"?* option so the user may opt-in to Automatic Phrasing, if desired.

Tasks for implementation

There are two tasks to implement Automatic Phrasing:

- Add phrases to your project using Developer Studio.
- Add Presentation API code to your Web application to support either of the two implementation scenarios described above.

Using Automatic Phrasing with Spelling Correction and DYM

You should enable the MDEX Engine for both Spelling Correction and Did You Mean.

If you want spelling corrected automatic phrases, the Spelling Correction feature ensures search terms are corrected *before* the terms are automatically phrased. The DYM feature provides users the choice to opt-in or opt-out of Automatic Phrasing.

The Oracle Commerce MDEX Engine applies spelling correction to a query before automatically phrasing the terms. This processing order means, for example, if a user misspells the query as *Napa Valle*, the MDEX Engine first spell corrects it to *Napa Valley* and then automatically phrases to "*Napa Valley*". Without Spelling Correction enabled, Automatic Phrasing would typically not find a matching phrase in the phrase dictionary.

If you implement Automatic Phrasing to rewrite the query using an automatic phrase, then enabling DYM allows users a way to opt-out of Automatic Phrasing if they want to. On the other hand, if you implement Automatic Phrasing to process the original query and suggest automatically-phrased alternatives, then enabling DYM allows users to take advantage of automatically-phrased alternatives as follow-up queries.

Automatic Phrasing and query expansion

Once individual search terms in a query are grouped as a phrase, the phrase is not subject to thesaurus expansion or stemming by the MDEX Engine.

Adding phrases to a project

This section describes the two methods of adding phrases to your project.

There are two ways to include phrases in your Developer Studio project:

- Import phrases from an XML file.
- Choose dimension names and extract phrases from the dimension values.

After you add phrases and update your instance configuration, the MDEX Engine builds the phrase dictionary. You cannot view the phrases in Developer Studio. However, after adding phrases and saving your project, you can examine the phrases contained in a project's phrase dictionary by using a text editor to open the `phrases.xml` project file. Directly modifying `phrases.xml` is not supported.

Importing phrases from an XML file

You import an XML file of phrases using the Import Phrases dialog box in Developer Studio.

The import phrases XML file must conform to `phrase_import.dtd`, found in the Oracle Commerce MDEX Engine `conf/dtd` directory. Here is a simple example of a phrase file that conforms to `phrase_import.dtd`:

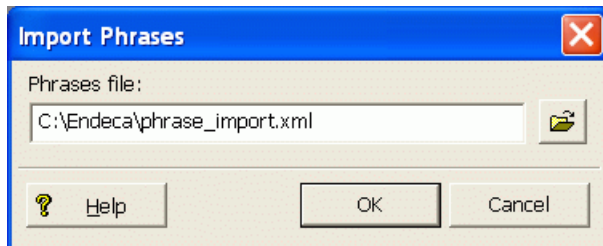
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE PHRASE_IMPORT SYSTEM "phrase_import.dtd">
<PHRASE_IMPORT>
  <PHRASE>Napa Valley</PHRASE>
  <PHRASE>low tannin</PHRASE>
</PHRASE_IMPORT>
```

To import phrases from an XML file:

1. Create the phrases XML file, using the format in the example above. You can create the file in any way you like. For example, you can type phrases into the file using an XML editor, or you can perform an XSLT transform on a phrase file in another format, and so on.

To maintain naming consistency with other Oracle Commerce project files and their corresponding DTD files, you may choose to name your file `phrase_import.xml`.

2. Open your project in Developer Studio.
3. In the Project Explorer, expand **Search Configuration**.
4. Double-click **Automatic Phrasing** to display the Automatic Phrasing editor.
5. Click the **Import Phrases...** button.
6. In the Import Phrases dialog box, either type the path to your phrases file or click the **Browse** button to locate the file.



7. Click **OK** on the Import Phrases dialog box.
The Messages pane displays the number of phrases read in from the XML file.
8. Click **OK** on the Automatic Phrasing dialog box.
9. Select **Save** from the File menu.

The project's `phrases.xml` configuration file is updated with the new phrases.

Keep in mind that if you import a newer version of an `import_phrases.xml` file, the most recent import overwrites phrases from any previous import. All phrases you want to import should be contained in a single XML file.

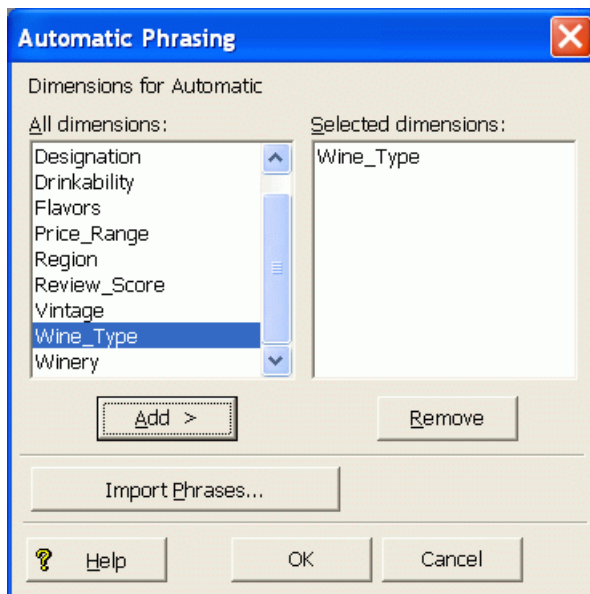
Extracting phrases from dimension names

Using Developer Studio, you can add phrases to your project based on the dimension values of any dimension you choose.

The MDEX Engine adds each multi-term dimension value in a selected dimension to the phrase dictionary. Single-term dimension values are not included. For example, if you import a WineType dimension from a wine catalog, the MDEX Engine creates a phrase entry for multi-term names such as "Pinot Noir" but not for single-term names such as "Merlot".

To extract phrases from dimension names:

1. Open your project in Developer Studio.
2. In the Project Explorer, expand **Search Configuration**.
3. Double-click **Automatic Phrasing** to display the Automatic Phrasing editor.
4. Select a dimension from the **All dimensions** panel and add it to the Selected dimensions panel by clicking **Add**. The editor should look like this example:



5. If desired, repeat step 4 to add more dimensions.
6. Click **OK** on the Automatic Phrasing dialog box.
7. Select **Save** from the File menu.

The project's `phrases.xml` configuration file is updated with the dimension names. Note that imported phrases are not overwritten by this procedure.

Adding search characters

If you have phrases that include punctuation, add those punctuation marks as search characters.

Adding the punctuation marks ensures that the MDEX Engine includes the punctuation when tokenizing the query, and therefore the MDEX Engine can match search terms with punctuation to phrases with punctuation.

For example, suppose you add phrases based on a Winery dimension, and consequently the Winery name "Anderson & Brothers" exists in your phrase dictionary. You should create a search character for the ampersand (&).

Presentation API development for Automatic Phrasing

The `ENEQuery` class has calls that handle Automatic Phrasing.

The Automatic Phrasing feature requires that the MDEX Engine compute whether an automatic phrase is available for a particular query's search terms.

The MDEX Engine computes the available phrases when setting the Java `setNavERecSearchComputeAlternativePhrasings()` method and the .NET `NavERecSearchComputeAlternativePhrasings` property to `true` in the `ENEQuery` object.

You can then optionally submit the phrased query to the MDEX Engine, instead of the user's original query, by calling the Java `setNavERecSearchRewriteQueryToAnAlternativePhrasing()` method or the .NET `NavERecSearchRewriteQueryToAnAlternativePhrasing` property with a value of `true`.

You can also call these methods by sending the necessary URL query parameters to the MDEX Engine via the `URLENEQuery` class, as shown in the next section.

When the MDEX Engine returns query results, your Web application displays whether the results were spell corrected, automatically phrased, or have DYM alternatives. Each of these Web application tasks are described in the sections below.

URL query parameters for Automatic Phrasing

Automatic Phrasing has two associated URL query parameters: `Ntpc` and `Ntpr`.

Both `Ntpc` and `Ntpr` are Boolean parameters that are enabled by setting to 1 and disabled by setting to 0.

The `Ntpc` parameter

Adding the `Ntpc=1` parameter instructs the MDEX Engine to compute phrasing alternatives for a query. Using this parameter alone, the MDEX Engine processes the original query and not any of the automatic phrasings computed by the MDEX Engine.

Here is an example URL that processes a user's query *napa valley* without phrasing and provides an alternative automatic phrasing, *Did you mean "napa valley"?*:

```
<application>?N=0&Ntk=All&Ntt=napa%20valley&Nty=1&Ntpc=1
```

If you omit `Ntpc=1` or set `Ntpc=0`, then automatic phrasing is disabled.

The `Ntpr` parameter

The `Ntpr` parameter instructs the MDEX Engine to rewrite the query using the available automatic phrase computed by `Ntpc`. The `Ntpr` parameter depends on the presence of `Ntpc=1`.

Here is an example URL that automatically phrases the user's query *napa valley* to *"napa valley"* and processes the phrased query. The Web application may also provide an unphrased alternative, so users can submit their original unphrased query (for example, *"Did you mean napa valley?"*):

```
<application>?N=0&Ntk=All&Ntt=napa%20valley&Nty=1&Ntpc=1&Ntpr=1
```

If you omit `Ntpr=1` or set `Ntpr=0`, then the query is not re-written using an automatic phrasing alternative. You can omit `Ntpr=1` and still use the `Ntpc=1` parameter to compute an available alternative for display as a DYM option.

Displaying spell-corrected and auto-phrased messages

To display messages for spell-corrected and automatically-phrased queries, your Web application code should be similar to these examples.

Java example

```
// Get the Map of lists of ESearchReport objects
Map recSrchrpts = nav.getESearchReportsComplete();
if (recSrchrpts.size() > 0) {
    // Get the user's search key
    String searchKey = request.getParameter("Ntk");
    if (searchKey != null) {
        if (recSrchrpts.containsKey(searchKey)) {
            // Get the ERecSearchReports for the search key
            List srchrptList = (List)recSrchrpts.get(searchKey);

            // for each report, display appropriate info
            for (Iterator i = srchrptList.iterator(); i.hasNext();) {
                ESearchReport srchrpt = (ESearchReport)i.next();

                // Get the List of auto-correct values
                List autoCorrectList = searchReport.getAutoSuggestions();

                // If the list contains Auto Suggestion objects,
                // print the value of the first corrected term
                if (autoCorrectList.size() > 0) {
                    // Get the Auto Suggestion object
                    ESearchAutoSuggestion autoSug =
                        (ESearchAutoSuggestion)autoCorrectList.get(0);

                    // Display appropriate autocorrect message
                    if (autoSug.didSuggestionIncludeSpellingCorrection() &&
                        !autoSug.didSuggestionIncludeAutomaticPhrasing()) {
                        %>Spelling corrected to <%= autoSug.getTerms() %> <%
                    }
                    else if (autoSug.didSuggestionIncludeSpellingCorrection() &&
                        autoSug.didSuggestionIncludeAutomaticPhrasing()) {
                        %>Spelling corrected and then phrased
                        to <%= autoSug.getTerms() %> <%
                    }
                    else if (!autoSug.didSuggestionIncludeSpellingCorrection() &&
                        autoSug.didSuggestionIncludeAutomaticPhrasing()) {
                        %>Phrased to <%= autoSug.getTerms() %> <%
                    }
                }
            }
        }
    }
}
```

.NET example

```
// Get the Dictionary of lists of ESearchReport objects
IDictionary recSrchrpts = nav.ESearchReportsComplete;

// Get the user's search key
String searchKey = Request.QueryString["Ntk"];

if (searchKey != null) {
    if (recSrchrpts.Contains(searchKey)) {
```



```

// Get the list of Search Report objects
IList srchReportList = (IList)recSrchrpts[searchKey];

// for each report, display appropriate info
foreach (object ob in srchReportList) {
    ESearchReport searchReport = (ESearchReport)ob;

    // Get the List of auto correct objects
    IList autoCorrectList = searchReport.AutoSuggestions;

    // If the list contains auto correct objects,
    // print the value of the first corrected term
    if (autoCorrectList.Count > 0) {
        // Get the Auto Suggestion object
        ESearchAutoSuggestion autoSug =
            (ESearchAutoSuggestion)autoCorrectList[0];

        // Display appropriate autocorrect message
        if (autoSug.GetDidSuggestionIncludeSpellingCorrection() &&
            !autoSug.GetDidSuggestionIncludeAutomaticPhrasing()) {
            %>Spelling corrected to <%= autoSug %> <%
        }
        else if (autoSug.GetDidSuggestionIncludeSpellingCorrection() &&
            autoSug.GetDidSuggestionIncludeAutomaticPhrasing()) {
            %>Spelling corrected and phrased to
            <%= autoSug.getTerms() %> <%
        }
        else if (!autoSug.GetDidSuggestionIncludeSpellingCorrection() &&
            autoSug.GetDidSuggestionIncludeAutomaticPhrasing()) {
            %>Phrased to <%= autoSug.getTerms() %> <%
        }
    }
}
}
}

```

Displaying DYM alternatives

To create a link for each Did You Mean alternative, your Web application code should look similar to these examples.

Note that it is important to display all the DYM alternatives (rather than just the first DYM alternative) because the user's desired query may not be the first alternative in the list of returned DYM options.

Java example

```

// Get the Map of ESearchReport objects
Map dymRecSrchrpts = nav.getESearchReports();
if (dymRecSrchrpts.size() > 0) {
    // Get the user's search key
    String searchKey = request.getParameter("Ntk");
    if (searchKey != null) {
        if (dymRecSrchrpts.containsKey(searchKey)) {
            // Get the List of ERecSearchReports for the user's search key
            List searchReportList = (List)dymRecSrchrpts.get(searchKey);

            // for each report, get the list of Did You Mean objects
            for (Iterator i = searchReportList.iterator(); i.hasNext();) {
                ESearchReport searchReport = (ESearchReport)i.next();

                // Get the List of Did You Mean objects
            }
        }
    }
}

```

```
List dymList = searchReport.getDYMSuggestions();  
// Get all Did You Mean objects to display each available  
// DYM alternative.  
for (Iterator j = dymList.Iterator(); j.hasNext();) {  
    ESearchDYMSuggestion dymSug =  
        (ESearchDYMSuggestion)j.next();  
    String sug_val = dymSug.getTerms();  
    String sug_num =  
        String.valueOf(dymSug.getNumMatchingResults());  
    String sug_sid = (String)request.getAttribute("sid");  
    if (sug_val != null) {  
  
        ...  
        // Adjust URL parameters to create new search query  
        UrlGen urlg =  
            new UrlGen(request.getQueryString(), "UTF-8");  
        urlg.removeParam("Ntt");  
        urlg.addParam("Ntt", sug_val);  
        urlg.removeParam("Ntpc");  
        urlg.addParam("Ntpc", "1");  
        urlg.removeParam("Ntpr");  
        urlg.addParam("Ntpr", "0");  
        String url = CONTROLLER+"?" +urlg;  
        // Display Did You Mean link for each DYM alternative  
        %>Did You Mean <a href="<%=url%>">  
            <%= sug_val %></a><%  
    }  
}  
}
```

.NET example

```
// Get the Dictionary of ESearchReport objects
IDictionary dymRecSrchrpts = nav.ESearchReports;

// Get the user's search key
String dymSearchKey = Request.QueryString["Ntk"];
if (dymSearchKey != null) {
    if (dymRecSrchrpts.Contains(dymSearchKey)) {
        // Get the list of Search Report objects
        IList srchrptList = (IList)recSrchrpts[searchKey];

        // for each report, display all its DYM suggestions
        foreach (object srObj in srchrptList) {
            // Get the List of Did You Mean objects
            IList dymList = ((ESearchReport)srObj).DYMSuggestions;
            foreach (object dymObj in dymList) {
                ESearchDYMSuggestion dymSug = (ESearchDYMSuggestion)dymObj;
                String sug_val = dymSug.Terms;
                String sug_num = dymSug.NumMatchingResults.ToString();
                // Adjust URL parameters to create new search query
                UrlGen urlg =
                    new UrlGen(Request.Url.Query.Substring(1), "UTF-8");
                urlg.RemoveParam("Ntt");
                urlg.AddParam("Ntt", sug_val);
                urlg.RemoveParam("Ntpc");
                urlg.AddParam("Ntpc", "1");
                urlg.RemoveParam("Ntpr");
            }
        }
    }
}
```

```

urlg.AddParam("Ntpr", "0");
urlg.AddParam("sid", Request.QueryString["sid"]);
String url = Application["CONTROLLER"].ToString()+"?" + urlg;
// Display Did You Mean message and link
// for each DYM option
%>Did You Mean <a href="<%= url %>">
<%= sug_val %></a>?<%
    }
}
}
}

```

Tips and troubleshooting for Automatic Phrasing

The following sections provide tips and troubleshooting guidance about using the Automatic Phrasing feature.

Examining how a phrased query was processed

If automatically-phrased query results are not what you expected, you can run the dgraph with the `--wordinterp` flag to show how the MDEX Engine processed the query.

Single-word phrases

You can include a single word in your `phrases_import.xml` file and treat the word as a phrase in your project. This may be useful if you do not want stemming or thesaurus expansion applied to single-word query terms. You cannot include single word phrases by extracting them from dimension values using the Phrases dialog box. They have to be imported from your `phrases_import.xml` file.

Extending user phrases

The MDEX Engine does not extend phrases a user provides to match a phrase in the phrase dictionary. For example, if a user provides the query *A "BC" D* and "BCD" is in the phrase dictionary, the MDEX Engine does not extend the user's original phrasing of "BC" to "BCD".

Term order is significant in phrases

Phrases are matched only if search terms are provided in the same exact order and with the same exact terms as the phrase in the phrase dictionary. For example, if "weekend bag" is in the phrase dictionary, the MDEX Engine does not automatically phrase the search terms *weekend getaway bag* or *bag, weekend* to match *weekend bag*.

Possible dead ends

If an application automatically phrases search terms, it is possible a query may not produce results when it seemingly should have. Specifically, one way in which a dead-end query can occur is when a search phrase is displayed as a DYM link with results and navigation state filtering excludes the results.

For example, suppose a car sales application is set up to process a user's original query and display any automatic phrase alternatives as DYM options. Further suppose a user navigates to **Cars > Less than \$15,000** and then provides the search terms *luxury package*. The search terms match the phrase "luxury package" in the phrase dictionary.

The user receives query results for **Cars > Less than \$15,000** and results that matched some occurrences of the terms *luxury* and *package*. However, if the user clicks the *Did you mean "luxury package"?* link, then no results are available because the navigation state **Cars > Less than \$15,000** excludes them.

Chapter 34

Stop Words

This section describes how to implement the Stop Words feature of the Oracle Commerce MDEX Engine.

About stop words

Stop words are words that are ignored by the Oracle Commerce MDEX Engine when the words are part of a keyword search. Typically, common words (like "the", "a", "as", etc.) are included in a stop word list and also terms that are very common in your data set.

For example, if your data set consists of lists of books, you might want to add the word "book" itself to the stop word list, because a search on that word would return an impractically large set of records. In addition, a stop word list can include the extraneous words contained in a typical question, allowing the query to focus on what the application user is really searching for.

Specifying stop words

There are two ways to specify stop words. You specify stop words manually in Developer Studio, and you can include one of the sample lists of stop words that are installed with the MDEX Engine.

Sample stop word lists

There is one sample list per language that the MDEX Engine supports. The sample lists are installed into `MDEX\<version>\olt\lang\stopword_samples` directory.

The MDEX Engine provides stop word files for the following languages: Catalan, Chinese (Simplified), Chinese (Traditional), Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Polish, Portuguese, Romanian, Russian, Spanish, Swedish, Thai, and Turkish. The language is identified in the file name by the `<language code>` value in `stop_words.<language code>.xml`.

Notes:

- Stop words are counted in any search mode that calculates results based on number of matching terms. However, the Oracle Commerce MDEX Engine reduces the minimum term match and maximum word omit requirement by the number of stop words contained in the query.
- Did You Mean can in some cases correct a word to one on the stop words list.
- The `--diacritic-folding` flag removes accent characters from stop words and prevents accented stop words from being returned in query results. For example, if `für` is a stop word, and you specify the `--diacritic-folding` flag, then that flag treats the stop word as `fur`. Any queries that search for `fur` will not return results.

Adding a sample list of stop words to an application

The MDEX Engine installation includes sample lists of stop words for each language that the MDEX Engine supports. If desired, you can incorporate one sample list of stop words into your application. The sample list can provide either the full set of stop words for your application, or it can provide a starting point that you add to using Developer Studio.

If you are using Developer Studio to create stop words, be sure to copy the sample list into your project before manually adding any stop words. The copy operation replaces the stop words created using Developer Studio.

To add a sample list of stop words to an application:

1. Locate the `olt\lang\stopword_samples` directory in the MDEX Engine installation directory.
For example, in a default Windows installation, this is
`C:\Endeca\MDEX\<version>\olt\lang\stopword_samples`.
2. In the `stopword_samples` directory, locate a sample file for the language of the records in your application.
3. Copy the sample file to the `<app_dir>\config\pipeline` directory.
4. Rename the file from `stop_words.<language code>.xml` to `<app_prefix>.stop_words.xml`
This step replaces the old stop words file.
5. Optionally, you can manually add stop words to the sample list using Developer Studio. To do so:
 - a) Start Developer Studio.
 - b) Open the Oracle Commerce project in the `<app_dir>\config\pipeline` directory.
 - c) Double click the **Stop Words** editor.
(Developer Studio loads the sample list of stop words and displays them in the **Stop Words** editor.)
 - d) Create additional stop words as necessary.
6. When you are adding the sample list of stop words, save and closet the project, then run a baseline update to process them using Dgidx.

Relevance Ranking

This section describes the tasks involved in implementing the Relevance Ranking feature of the Oracle Commerce MDEX Engine.

About the Relevance Ranking feature

Relevance Ranking controls the order in which search results are displayed to the end user of an Oracle Commerce application.

Application users are often unwilling to page through large result sets. For this reason, you need to be able to display the more important search results before the less important results. The Relevance Ranking feature enables you to do this.

Relevance ranking can control the order in which results from both record search and dimension search queries are displayed. However, while relevance ranking for record search can be configured with Developer Studio, relevance ranking for dimension search cannot. (You assign relevance ranking for dimension search through the `REL RANK_STRATEGY` attribute of `dimsearch_config.xml`, or at query time by specifying the `Dx` and `Dk` parameters of the `UrLENEQuery`.)

The importance of a search result is generally an application-specific concept. The Relevance Ranking feature provides a flexible, configurable set of result ranking modules. These modules can be used in combinations (called ranking strategies) to produce a wide range of relevance ranking effects. Because Relevance Ranking is a complex and powerful feature, Oracle Commerce provides recommended strategies that you can use as a point of departure for further development. For details, see [Recommended strategies](#) on page 347.

Relevance Ranking modules

Relevance Ranking modules are the building blocks from which you build the relevance ranking strategies that you apply to your search interfaces.

This section describes the available set of Relevance Ranking modules and their scoring behaviors.



Note: Some modules are listed in the Developer Studio interface by their abbreviated spellings, such as "Interp" for Interpreted.

Exact

The Exact module groups results into three strata based on how well they match the query string:

- The highest stratum contains results whose complete text matches the user's query exactly.
- The middle stratum contains results that contain the user's query as a subphrase.
- The lowest stratum contains other hits (such as normal conjunctive matches). Any match that would not be a match without query expansion lands in the lowest stratum. Also in this stratum are records that do not contain relevance ranking terms (such as those specified in the `Nrr` query parameter).



Note: The Exact module is computationally expensive, especially on large text fields. It is intended for use only on small text fields (such as dimension values or small property values like part IDs). This module should not be used with large or offline documents (such as `FILE` or `ENCODED_FILE` properties). Use of this module in these cases will result in very poor performance and/or application failures due to request timeouts. The Phrase module, with and without approximation turned on, does similar but less sophisticated ranking that can be used as a higher performance substitute.

Field

The Field module ranks documents based on the search interface field with the highest priority in which it matched.

Only the best field in which a match occurs is considered. The Field module is often used in relevance ranking strategies for catalog applications, because the category or product name is typically a good match. Field assigns a score to each result based on the static rank of the dimension or property member or members of the search interface that caused the document to match the query. In Developer Studio, static field ranks are assigned based on the order in which members of a search interface are listed in the Search Interfaces view. The first (left-most) member has the highest rank.

By default, matches caused by cross-field matching are assigned a score of zero. The score for cross-field matches can be set explicitly in Developer Studio by moving the `<<CROSS_FIELD>>` indexator up or down in the Selected Members list of the Search Interface editor. The `<<CROSS_FIELD>>` indexator is available only for search interfaces that have the Field module and are configured to support cross-field matches. All non-zero ranks must be non-equal and only their order matters.

For example, a search interface might contain both Title and DocumentContent properties, where hits on Title are considered more important than hits on DocumentContent (which in turn are considered more important than `<<CROSS_FIELD>>` matches). Such a ranking is implemented by assigning the highest rank to Title, the next highest rank to DocumentContent, and setting the `<<CROSS_FIELD>>` indexator at the bottom of the Selected Members list in the Search Interface editor.



Note: The Field module is only valid for record search operations. This module assigns a score of zero to all results for other types of search requests. In addition, Field treats all matches the same, whether or not they are due to query expansion.

First

Designed primarily for use with unstructured data, the First module ranks documents by how close the query terms are to the beginning of the document.

The First module groups its results into variably-sized strata. The strata are not the same size, because while the first word is probably more relevant than the tenth word, the 301st is probably not so much more relevant

than the 310th word. This module takes advantage of the fact that the closer something is to the beginning of a document, the more likely it is to be relevant.

The First module works as follows:

- When the query has a single term, First's behavior is straight-forward: it retrieves the first absolute position of the word in the document, then calculates which stratum contains that position. The score for this document is based upon that stratum; earlier strata are better than later strata.
- When the query has multiple terms, First behaves as follows: The first absolute position for each of the query terms is determined, and then the median position of these positions is calculated. This median is treated as the position of this query in the document and can be used with stratification as described in the single word case.
- With query expansion (using stemming, spelling correction, or the thesaurus), the First module treats expanded terms as if they occurred in the source query. For example, the phrase *glucose intolerance* would be corrected to *glucose intolerance* (with *intolerance* spell-corrected to *intolerance*). First then continues as it does in the non-expansion case. The first position of each term is computed and the median of these is taken.
- In a partially matched query, where only some of the query terms cause a document to match, First behaves as if the intersection of terms that occur in the document and terms that occur in the original query were the entire query. For example, if the query *cat bird dog* is partially matched to a document on the terms *cat* and *bird*, then the document is scored as if the query were *cat bird*. If no terms match, then the document is scored in the lowest strata.
- The First relevance ranking module is supported for wildcard queries.



Note: The First module does not work with Boolean searches and cross-field matching. It assigns all such matches a score of zero.

Frequency

The Frequency (Freq) module provides result scoring based on the frequency (number of occurrences) of the user's query terms in the result text.

Results with more occurrences of the user search terms are considered more relevant.

The score produced by the Freq module for a result record is the sum of the frequencies of all user search terms in all fields (properties or dimensions in the search interface in question) that match a sufficient number of terms. The number of terms depends on the match mode, such as all terms in a MatchAll query, a sufficient number of terms in a MatchPartial query, and so on. Cross-field match records are assigned a score of zero. Total scores are capped at 1024; in other words, if the sum of frequencies of the user search terms in all matching fields is greater than or equal to 1024, the record gets a score of 1024 from the Freq module.

For example, suppose we have the following record:

```
{Title="test record", Abstract="this is a test", Text="one test this is"}
```

A MatchAll search for *test this* would cause Freq to assign a score of 4, since *this* and *test* occur a total of 4 times in the fields that match all search terms (Abstract and Text, in this case). The number of phrase occurrences (just one in the Text field) doesn't matter, only the sum of the individual word occurrences. Also note that the occurrence of *test* in the Title field does not contribute to the score, since that field did not match all of the terms.

A MatchAll search for *one record* would hit this record, assuming that cross field matching was enabled. But the record would get a score of zero from Freq, because no single field matches all of the terms. Freq ignores matches due to query expansion (that is, such matches are given a rank of 0).

Glom

The Glom module ranks single-field matches ahead of cross-field matches and also ahead of non-matches (records that do not contain the search term).

This module serves as a useful tie-breaker function in combination with the Maximum Field module. It is only useful in conjunction with record search operations. If you want a strategy that ranks single-field matches first, cross-field matches second, and no matches third, then use the Glom module followed by the Nterms (Number of Terms) module.



Note: Glom treats all matches the same, whether or not they are due to query expansion.

Glom interaction with search modes

The Glom module considers a single-field match to be one in which a single field has enough terms to satisfy the conditions of the match mode. For this reason, in MatchAny search mode, cross-field matches are impossible, because a single term is sufficient to create a match. Every match is considered to be a single-field match, even if there were several search terms.

For MatchPartial search mode, if the required number of matches is two, the Glom module considers a record to be a single-field match if it has at least one field that contains two or more of the search terms. You cannot rank results based on how many terms match within a single field.

Interpreted

Interpreted (Interp) is a general-purpose module that assigns a score to each result record based on the query processing techniques used to obtain the match.

Matching techniques considered include partial matching, cross-attribute matching, spelling correction, thesaurus, and stemming matching.

Specifically, the Interpreted module ranks results as follows:

1. All non-partial matches are ranked ahead of all partial matches. For more information, see "Using Search Modes".
2. Within the above strata, all single-field matches are ranked ahead of all cross-field matches. For more information, see "About Search Interfaces".
3. Within the above strata, all non-spelling-corrected matches are ranked above all spelling-corrected matches. See the topic "Using Spelling Correction and Did You Mean" for more information.
4. Within the above strata, all thesaurus matches are ranked below all non-thesaurus matches. See the topic "Using Stemming and Thesaurus" for more information.
5. Within the above strata, all stemming matches are ranked below all non-stemming matches. See "Using Stemming and Thesaurus" for more information.



Note: Because the Interpreted module comprises the matching techniques of the Spell, Glom, Stem, and Thesaurus modules, there is no need to add them to your strategy individually as well if you are using Interpreted.

Maximum Field

The Maximum Field (Maxfield) module behaves identically to the Field module, except in how it scores cross-field matches.

Unlike Field, which assigns a static score to cross-field matches, Maximum Field selects the score of the highest-ranked field that contributed to the match.

Note the following:

- Because Maximum Field defines the score for cross-field matches dynamically, it does not make use of the <<CROSS_FIELD>> indexator set in the Search Interface editor.
- Maximum Field is only valid for record search operations. This module assigns a score of zero to all results for other types of search requests.
- Maximum Field treats all matches the same, whether or not they are due to query expansion.

Number of Fields

The Number of Fields (Numfields) module ranks results based on the number of fields in the associated search interface in which a match occurs.

Note that we are counting whole-field rather than cross-field matches. Therefore, a result that matches two fields matches each field completely, while a cross-field match typically does not match any field completely.



Note: Numfields treats all matches the same, whether or not they are due to query expansion. The Numfields module is only useful in conjunction with record search operations.

Number of Terms

The Number of Terms (or Nterms) module ranks matches according to how many query terms they match.

For example, in a three-word query, results that match all three words will be ranked above results that match only two, which will be ranked above results that match only one, which will be ranked above results that had no matches.

Note the following:

- The Nterms module is only applicable to search modes where results can vary in how many query terms they match. These include MatchAny, MatchPartial, MatchAllAny, and MatchAllPartial.
- Nterms treats all matches the same, whether or not they are due to query expansion.

Phrase

The Phrase module states that results containing the user's query as an exact phrase, or a subset of the exact phrase, should be considered more relevant than matches simply containing the user's search terms scattered throughout the text.

Records that have the phrase are ranked higher than records which do not contain the phrase.

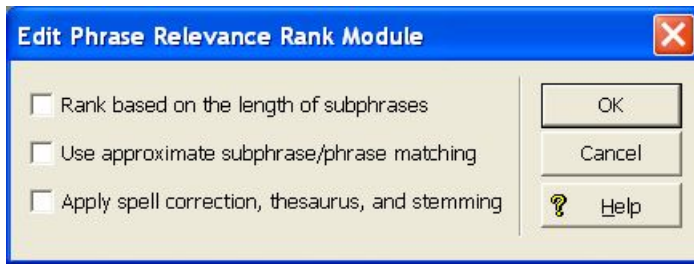
Configuring the Phrase module

The Phrase module has a variety of options that you use to customize its behavior.

The Phrase options are:

- Rank based on length of subphrases
- Use approximate subphrase/phrase matching
- Apply spell correction, thesaurus, and stemming

When you add the Phrase module in the Relevance Ranking Modules editor, you are presented with the following editor that allows you to set these options.



Ranking based on length of subphrases

When you configure the Phrase module, you have the option of enabling subphrasing.

Subphrasing ranks results based on the length of their subphrase matches. In other words, results that match three terms are considered more relevant than results that match two terms, and so on.

A subphrase is defined as a contiguous subset of the query terms the user entered, in the order that he or she entered them. For example, the query "fax cover sheets" contains the subphrases "fax", "cover", "sheets", "fax cover", "cover sheets", and "fax cover sheets", but not "fax sheets".

Content contained inside nested quotes in a phrase is treated as one term. For example, consider the following phrase:

the question is "to be or not to be"

The quoted text ("to be or not to be") is treated as one query term, so this example consists of four query terms even though it has a total of nine words.

When subphrasing is not enabled, results are ranked into two strata: those that matched the entire phrase and those that did not.

Using approximate matching

Approximate matching provides higher-performance matching, as compared to the standard Phrase module, with somewhat less exact results.

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

The approximate setting is appropriate in cases where the runtime performance of the standard Phrase module is inadequate because of large result contents and/or high site load.

Applying spelling correction, thesaurus, and stemming

Applying spelling correction, thesaurus, and stemming adjustments to the original phrase is generically known as query expansion.

With query expansion enabled, the Phrase module ranks results that match a phrase's expanded forms in the same stratum as results that match the original phrase.

Consider the following example:


- A thesaurus entry exists that expands "US" to "United States".
- The user queries for "US government".

The query "US government" is expanded to "United States government" for matching purposes, but the Phrase module gives a score of two to any results matching "United States government" because the original, unexpanded version of the query, "US government", only had two terms.

Summary of Phrase option interactions

The three configuration settings for the Phrase module can be used in a variety of combinations for different effects.

The following matrix describes the behavior of each combination.

Subphrase	Approximate	Expansion	Description
Off	Off	Off	Default. Ranks results into two strata: those that match the user's query as a whole phrase, and those that do not.
Off	Off	On	Ranks results into two strata: those that match the original, or an extended version, of the query as a whole phrase, and those that do not.
Off	On	Off	Ranks results into two strata: those that match the original query as a whole phrase, and those that do not. Look only at the first possible phrase match within each record.
Off	On	On	Ranks results into two strata: those that match the original, or an extended version, of the query as a whole phrase, and those that do not. Look only at the first possible phrase match within each record.
On	Off	Off	Ranks results into N strata where N equals the length of the query and each result's score equals the length of its matched subphrase.
On	Off	On	Ranks results into N strata where N equals the length of the query and each result's score equals the length of its matched subphrase. Extend subphrases to facilitate matching but rank based on the length of the original subphrase (before extension).  Note: This combination can have a negative performance impact on query throughput.
On	On	Off	Ranks results into N strata where N equals the length of the query and each result's score equals the length of its matched subphrase. Look only at the first possible phrase match within each record.
On	On	On	Ranks results into N strata where N equals the length of the query and each result's score equals the length of its matched subphrase. Expand the query to facilitate matching but rank based on the length of the original subphrase (before extension). Look only at the first possible phrase match within each record.



Note: You should only use one Phrase module in any given search interface and set all of your options in it.

Effect of search modes on Phrase behavior

Oracle Commerce provides a variety of search modes to facilitate matching during search (MatchAny, MatchAll, MatchPartial, and so on).

These modes only determine which results match a user's query, they have no effect on how the results are ranked after the matches have been found. Therefore, the Phrase module works as described in this section,

regardless of search mode. The one exception to this rule is MatchBoolean. Phrase, like the other relevance ranking modules, is never applied to the results of MatchBoolean queries.

Results with multiple matches

If a single result has multiple subphrase matches, either within the same field or in several different fields, the result is slotted into a stratum based on the length of the longest subphrase match.

Stop words and Phrase behavior

When using the Phrase module, stop words are always treated like non-stop word terms and stratified accordingly.

For example, the query “raining cats and dogs” will result in a rank of two for a result containing “fat cats and hungry dogs” and a rank of three for a result containing “fat cats and dogs” (this example assumes subphrase is enabled).

Cross-field matches and Phrase behavior

An entire phrase, or subphrase, must appear in a single field in order for it to be considered a match.

(In other words, matches created by concatenating fields are not considered by the Phrase module.)

Treatment of wildcards with the Phrase module

The Phrase module translates each wildcard in a query into a generic placeholder for a single term.

For example, the query “sparkling w* wine” becomes “sparkling * wine” during phrase relevance ranking, where “*” indicates a single term. This generic wildcard replacement causes slightly different behavior depending on whether subphrasing is enabled.

When subphrasing is not enabled, all results that match the generic version of the wildcard phrase exactly are still placed into the first stratum. It is important, however, to understand what constitutes a matching result from the Phrase module’s point of view.

Consider the search query “sparkling w* wine” with the MatchAny mode enabled. In MatchAny mode, search results only need to contain one of the requested terms to be valid, so a list of search results for this query could contain phrases that look like this:

```
sparkling white wine
sparkling refreshing wine
sparkling wet wine
sparkling soda
wine cooler
```

When phrase relevance ranking is applied to these search results, the Phrase module looks for matches to “sparkling * wine” not “sparkling w* wine.” Therefore, there are three results—“sparkling white wine,” “sparkling refreshing wine,” and “sparkling wet wine”—that are considered phrase matches for the purposes of ranking. These results are placed in the first stratum. The other two results are placed in the second stratum.

When subphrasing is enabled, the behavior becomes a bit more complex. Again, we have to remember that wildcards become generic placeholders and match any single term in a result. This means that any subphrase that is adjacent to a wildcard will, by definition, match at least one additional term (the wildcard). Because of this behavior, subphrases break down differently. The subphrases for “cold sparkling w* wine” break down into the following (note that w* changes to *):

```
cold
sparkling *
```

```
* wine
cold sparkling *
sparkling * wine
cold sparkling * wine
```

Notice that the subphrases “sparkling,” “wine,” and “cold sparkling” are not included in this list. Because these subphrases are adjacent to the wildcard, we know that the subphrases will match at least one additional term. Therefore, these subphrases are subsumed by the “sparkling *”, “* wine”, and “cold sparkling *” subphrases.

Like regular subphrase, stratification is based on the number of terms in the subphrase, and the wildcard placeholders are counted toward the length of the subphrase. To continue the example above, results that contain “cold” get a score of one, results that contain “sparkling *” get a score of two, and so on. Again, this is the case even if the matching result phrases are different, for example, “sparkling white” and “sparkling soda.”

Finally, it is important to note that, while the wildcard can be replaced by any term, a term must still exist. In other words, search results that contain the phrase “sparkling wine” are not acceptable matches for the phrase “sparkling * wine” because there is no term to substitute for the wildcard. Conversely, the phrase “sparkling cold white wine” is also not a match because each wildcard can be replaced by one, and only one, term. Even when wildcards are present, results must contain the correct number of terms, in the correct order, for them to be considered phrase matches by the Phrase module.

Notes about the Phrase module

Keep the following points in mind when using the Phrase module.

- If a query contains only one word, then that word constitutes the entire phrase and all of the matching results will be put into one stratum (score = 1). However, the module can rank the results into two strata: one for records that contain the phrase and a lower-ranking stratum for records that do not contain the phrase.
- Because of the way hyphenated words are positionally indexed, Oracle recommends that you enable subphrase if your results contain hyphenated words.

Proximity

Designed primarily for use with unstructured data, the Proximity module ranks how close the query terms are to each other in a document by counting the number of intervening words.

Like the First module, this module groups its results into variable sized strata, because the difference in significance of an interval of one word and one of two words is usually greater than the difference in significance of an interval of 21 words and 22. If no terms match, the document is placed in the lowest stratum.

Single words and phrases get assigned to the best stratum because there are no intervening words. When the query has multiple terms, Proximity behaves as follows:

1. All of the absolute positions for each of the query terms are computed.
2. The smallest range that includes at least one instance of each of the query terms is calculated. This range's length is given in number of words. The score for each document is the strata that contains the difference of the range's length and the number of terms in the query; smaller differences are better than larger differences.

Under query expansion (that is, stemming, spelling correction, and the thesaurus), the expanded terms are treated as if they were in the query, so the proximity metric is computed using the locations of the expanded terms in the matching document.

For example, if a user searches for *big cats* and a document contains the sentence, "Big Bird likes his cat" (stemming takes *cats* to *cat*), then the proximity metric is computed just as if the sentence were, "Big Bird likes his cats."

Proximity scores partially matched queries as if the query only contained the matching terms. For example, if a user searches for *cat dog fish* and a document is partially matched that contains only *cat* and *fish*, then the document is scored as if the query *cat fish* had been entered.



Note: Proximity does not work with Boolean searches, cross-field matching, or wildcard search. It assigns all such matches a score of zero.

Spell

The Spell module ranks spelling-corrected matches below other kinds of matches.

Spell assigns a rank of 0 to matches from spelling correction, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

Static

The Static module assigns a static or constant data-specific value to each search result, depending on the type of search operation performed and depending on optional parameters that can be passed to the module.

For record search operations, the first parameter to the module specifies a property, which will define the sort order assigned by the module. The second parameter can be specified as ascending or descending to indicate the sort order to use for the specified property.

For example, using the module `Static(Availability,descending)` would sort result records in descending order with respect to their assignments from the Availability property. Using the module `Static(Title,ascending)` would sort result records in ascending order by their Title property assignments.

In a catalog application, setting the static module by Price, descending leads to more expensive products being displayed first.

For dimension search, the first parameter can be specified as `nbins`, `depth`, or `rank`:

- Specifying `nbins` causes the static module to sort result dimension values by the number of associated records in the full data set.
- Specifying `depth` causes the static module to sort result dimension values by their depth in the dimension hierarchy.
- Specifying `rank` causes dimension values to be sorted by the ranks assigned to them for the application.

Stratify

The Stratify module is used to boost or bury records in the result set.

The Stratify module takes one or more EQL (Oracle Commerce Query Language) expressions and groups results into strata, based on how well they match the record search (with the `Ntx` parameter). 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. If an asterisk is specified instead of an EQL expression, unmatched records are placed in the corresponding stratum.

The Stratify module can also be used for record boost and bury sort operations. In this usage, you must specify `Endeca.stratify` as the name for the `Ns` parameter.

The Stratify module is the basic component of the record boost and bury feature.

Stem

The Stem module ranks matches due to stemming below other kinds of matches.

Stem assigns a rank of 0 to matches from stemming, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

Thesaurus

The Thesaurus module ranks matches due to thesaurus entries below other sorts of matches.

Thesaurus assigns a rank of 0 to matches from the thesaurus, and a rank of 1 from all other sources. That is, it ignores all other sorts of query expansion.

Weighted Frequency

Like the Frequency module, the Weighted Frequency (Wfreq) module scores results based on the frequency of user query terms in the result.

Additionally, the Weighted Frequency module weights the individual query term frequencies for each result by the information content (overall frequency in the complete data set) of each query term. Less frequent query terms (that is, terms that would result in fewer search results) are weighted more heavily than more frequently occurring terms.



Note: The Weighted Frequency module ignores matches due to query expansion (that is, such matches are given a rank of 0).

Relevance Ranking strategies

Relevance Ranking modules define the primitive search result ordering functions provided by the MDEX Engine. These primitive modules can be combined to compose more complex ordering behaviors called Relevance Ranking strategies.

You may also define and apply a strategy that consists of a single module, rather than a group of modules.

A Relevance Ranking strategy is essentially an ordered list of relevance ranking modules and (in a URL relevance ranking string) references to other relevance ranking strategies. The scores assigned by a strategy are composed from the scores assigned by its constituent modules. This composite score is constructed so that records are first ordered by the first module. After that, ties are broken by the subsequent modules in order. If any ties remain after all modules have run, the ties are resolved by the default sort. If after that any ties still remain, the order of records is determined by the system.

Relevance Ranking strategies are used in two main contexts in the MDEX Engine:

- In Developer Studio, you apply Relevance Ranking to a search interface via the Search Interface editor and the Relevance Ranking Modules editor, both of which are documented in Developer Studio online help.
- At the MDEX Engine query level, Relevance Ranking strategies can be selected to override the default specified for the selected search interface. This allows Relevance Ranking behavior to be fully customized on a per-query basis. For details, see the "URL query parameters for relevance ranking" topic.

Implementing relevance ranking

Developer Studio allows you to create and control relevance ranking for record search.

You can apply record search relevance ranking as you are creating a search interface, or afterwards. A search interface is a named group of at least one dimension and/or property. You create search interfaces so you can apply behavior like relevance ranking across a group. For more information about search interfaces, see "About Search Interfaces".

Adding a Static module

Keep the following in mind when you add a Static module to the ranking strategy.

The Static module is the only one that you can add multiple times. The interface prevents the addition of multiple instances of the other modules. In addition, adding a Static module launches the Edit Static Relevance Rank Module editor. Use this editor to add the required parameters (dimension or property name and sort order).

Ranking order for Field and Maximum Field modules

The Field and Maximum Field modules rank results based on which member property or dimension of the selected search interface caused the match.

Higher relevance-ranked values correspond to greater importance. This behavior means that the Field and Maximum Field modules will score results caused by higher-ranked properties and dimensions ahead of those caused by lower-ranked properties and dimensions.

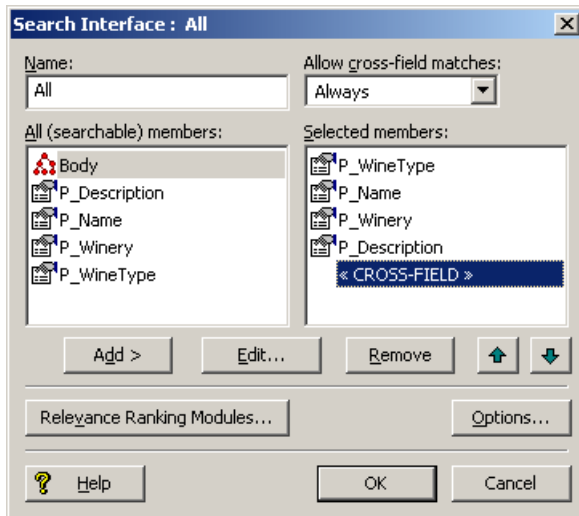
To change the relevance ranking behavior for these modules, you would move the search interface members to the appropriate position in the Search Interface editor's Selected Members list, using the Up and Down arrows.

Cross-field matching for the Field module

For search interfaces that allow cross-field matches and have a Field module, you can configure the static score assigned to cross-field matches by the Field module on an individual search interface.

You might do this if you considered cross-field matches better than description-only matches.

Such a search interface would appear similar to this example in the Search Interface editor:



In the example, note the presence of the <<CROSS-FIELD>> indexator in the Selected Members list. This indexator is present only in search interfaces with Always or On Failure cross-field matches and a ranking strategy that includes a Field module.

How relevance ranking score ties between search interfaces are resolved

In the case of multiple search interfaces and relevance ranking score ties, ties are broken based on the relevance ranking sort strategy of the search interface with the highest relevance ranking score for a given record.

If two different records belong to different search interfaces, the record from the search interface specified earlier in the query comes first.

Implementing relevance ranking strategies for dimension search

MDEX Engine configuration is not required to configure a relevance ranking strategy for record search.

To define the relevance ranking strategy for dimension search operations, modify the RELRANK_STRATEGY attribute of `dimsearch_config.xml`. This attribute specifies the name of a relevance ranking strategy for dimension search. The content of this attribute should be a relevance ranking string, as in the following examples:

```
exact,static(rank,descending)
interp,exact
```

For details about the format of the relevance ranking string, see the "URL query parameters for relevance ranking" topic.

The default ranking strategy for dimension search operations, which is applied if you do not make any changes to it, is:

```
interp,exact,static
```

The default ranking strategy for record search operations is no strategy. That is, unless you explicitly establish a relevance ranking strategy, none is used.

Retrieving the relevance ranking for records

The `dgraph --stat-brel` flag creates a dynamic property on each record named `DGraph.BinRelevanceRank`. The value of this property reflects the relevance rank assigned to a record in full text search.

The Java `ERec.getProperties()` method and the .NET `ERec.Properties` property return a list of properties (`PropertyMap` object) associated with the record. At this point, calling the Java `PropertyMap.get()` method or the .NET `PropertyMap` object with the `DGraph.BinRelevanceRank` argument returns the value of the property.

The following code samples show how to retrieve the `DGraph.BinRelevanceRank` for a given record.

Java example

```
// get the record list from the navigation object
ERecList recs = nav.getERecs();
// loop over record list
for (int i=0; i<recs.size(); i++) {
    // get individual record
    ERec rec = (ERec)recs.get(i);
    // get property map for record
    PropertyMap propsMap = rec.getProperties();
    // Check for a non-null relevance rank property
    if (propsMap.get("DGraph.BinRelevanceRank") != null) {
        String rankNum =
            (String)propsMap.get("DGraph.BinRelevanceRank");
        %>Relevance ranking for this record:
        <%= rankNum %>
        <%
    } // end of if
} // end of for loop iteration
```

.NET example

```
// get the record list from the navigation object
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;
    // Check for a non-null relevance rank property
    String rankNum = "";
    if (propsMap["DGraph.BinRelevanceRank"] != null) {
        rankNum = (String)propsMap["DGraph.BinRelevanceRank"];
        %>Relevance ranking for this record:
        <%= rankNum %>
        <%
    } // end of if
} // end of for loop iteration
```

Interpreting the values of `DGraph.BinRelevanceRank`

The MDEX Engine sorts records for relevance ranking using a more granular algorithm than the number you retrieve with `DGraph.BinRelevanceRank`.

If, for example, you need to interpret the values of the `DGraph.BinRelevanceRank` property for two different records, it is helpful to know that while these values roughly represent the sorting used for relevance-ranked

records, they are not as precise as the internal sorting numbers the MDEX Engine actually uses to sort the records.

For example, you may see the same `DGraph.BinRelevanceRank` value for two records that are sorted slightly differently. When interpreting the results of `DGraph.BinRelevanceRank` for two different records, consider these values as providing rough guidance only on whether one record has a significantly higher relevance rank than the other. However, if the value of `DGraph.BinRelevanceRank` is the same, this does not mean that the records are sorted the same, since the underlying sorting mechanism in the MDEX Engine is more precise. It is important to note that the MDEX Engine always returns consistent results and consistently interprets tie breaks in sorting, if they occur.

Controlling relevance ranking at the query level

At the MDEX Engine query level, relevance ranking strategies can be selected to override the default specified for the selected search interface.

This allows relevance ranking behavior to be fully customized on a per-query basis. MDEX Engine URL relevance ranking strategy strings must contain one or more relevance ranking module names. Module names can be any of these pre-defined modules:

- `exact`
- `field` (useful for record search only)
- `first`
- `freq`
- `glom` (useful for record search only)
- `interp`
- `maxfield` (useful for record search only)
- `nterms`
- `numfields` (useful for record search only)
- `phrase` (for details about using phrase, see the section below)
- `proximity`
- `spell`
- `stem`
- `thesaurus`
- `static` (for details about using static, see the section below)
- `wfreq`

Module names are delimited by comma (,) characters. No other stray characters (such as spaces) are allowed. Module names are listed in descending order of priority.

Exact module, First module, Nterms module, and Proximity module details

The Exact, First, Nterms, and Proximity modules can take one parameter named `considerFieldRanks`. If specified, the `considerFieldRanks` parameter indicates that the module should further sort records according to field ranking scores, after the records have been sorted according to the standard behavior of the module.

For example, if you specify `exact` without the parameter in a query, records that are an exact match are sorted into a strata that is higher than non-exact matches. Within each strata, the records are only sorted according to the default sort order or a specified sort key.

If you add the `considerFieldRanks` parameter to URL query syntax and specify `exact(considerFieldRanks)`, the records within each strata are sorted so that those with higher field ranking scores are more relevant than those with lower field ranking scores within the same strata.

Freq module and Numfields module details

The Freq module and also the Numfields module can take one parameter named `considerFieldRanks`. If specified, the `considerFieldRanks` parameter indicates that the module should further sort records according to ranking scores that are calculated across multiple fields, after the records have been sorted according to the standard behavior of the module. For these modules, cross-field matches are weighted such that matches in higher ranked fields contribute more than matches in lower ranked fields.

Phrase module details

The Phrase module can take up to four parameters:

- `approximate` - enables approximate matching.
- `considerFieldRanks` - enables further sorting according to the field rank score of the match. If specified, the `considerFieldRanks` parameter indicates that the module should further sort records according to field ranking scores, after the records have been sorted according to the standard behavior of the module.
- `query_expansion` - enables query expansion.
- `subphrase` - enables subphrase matching

The presence of a parameter indicates that the feature should be enabled, and the parameters can be in any order. For example: `phrase(subphrase, approximate, query_expansion)`

Static module details

The Static module takes two parameters. For record search, the first parameter is a property or dimension to use for assigning static scores (based on sort order) and the second is the sort order: ascending (ascend is an accepted abbreviation) or descending (or descend). The default is ascending. The parameters must be a comma-separated list enclosed in parentheses. For example: `static(Price, ascending)`

For dimension search, the first parameter can be specified as nbins, depth, or rank:

- Specifying nbins causes the static module to sort result dimension values by the number of associated records in the full data set.
- Depth causes the static module to sort result dimension values by their depth in the dimension hierarchy.
- Rank causes dimension values to be sorted by the ranks assigned to them for the application. In cases when there are ties, (for example, if you specify nbins and the number of associated records is the same), the system ranks dimension search results based on the dimension value IDs.

Valid relevance ranking strings

The following are examples of valid relevance ranking strategy strings:

- `exact`
- `exact(considerFieldRanks)`
- `field,phrase,interp`
- `static(Price, ascending)`
- `static(Availability, descending), exact, static(Price, ascending)`
- `field, MyStrategy, exact` (assuming that `MyStrategy` is the name of a valid search interface with a relevance ranking strategy)
- `phrase(approximate, subphrase)`

URL query parameters for relevance ranking

URL query parameters allow you to communicate with the Presentation API from your client browser.

There are two sets of URL query parameters that allow you to specify relevance ranking modules that will order the returned record set:

- `Dk`, `Dx`, and `Ntx` parameters.
- `Nrk`, `Nrt`, and `Nrr` parameters.

Note that all of these parameters must be specified together. These sets of URL parameters are described in the following two sections.

Using the `Dk`, `Dx`, and `Ntx` parameters

This topic describes the use of query parameters with relevance ranking.

The following query parameters affect relevance ranking:

```
Dk=<0 | 1>
Dx=rel+strategy
Ntx=rel+strategy
```

For the `Dx` and `Ntx` parameters, the `rel` option sets the relevance ranking strategy. For a list of valid module names to use in the strategy, see the "Controlling relevance ranking at the query level" topic.

Relevance ranking for record search operations is automatic. Results are returned in descending order of relevance as long as a relevance ranking strategy is enabled (either in the URL or as the default for the selected search interface) and if the user has not selected an explicit record sort operation in the record search request. If the user has requested an explicit sort ordering, relevance rank ordering for results does not apply.

For dimension search operations, relevance ranking is enabled by the `Dk` parameter. The value of this (optional) parameter can be set to zero or one:

- If the value is set to one, the dimension search results will be returned in relevance-ranked order
- If the value is set to zero, the results will be returned in their default order

The default value if the parameter is omitted is zero (that is, relevance ranking is not enabled).

For both dimension search and record search operations, the relevance ranking strategy used for the current request can be selected using the search option URL parameters (`Dx` and `Ntx`) as in the following examples:

```
<application>?D=mark+twain&Dk=1
&Dx=rel+exact,static(rank,descending)

<application>?N=0&Ntk=All&Ntt=polo+shirt
&Ntx=mode+matchany+rel+MyStrategy
```

The second example assumes that `MyStrategy` was defined in Developer Studio, and is specified via the `rel` option (which sets the relevance ranking option). The example also uses the `mode` option (which requests "match any word" query matching).

Using URL-defined strategies (as in the first example) can be especially useful during development, when you want to compare the results of multiple strategies quickly. Once you have determined what strategy works best, you can define the strategy in a search interface in Developer Studio.

Using the `Nrk`, `Nrt`, `Nrr`, and `Nrm` parameters

You can use the following set of parameters to order the records of a record search via a specified relevance ranking strategy.

The parameters are:

```
Nrk=search-interface
Nrt=relrank-terms
```

```
Nrr=relrank-strategy
Nrm=relrank-matchmode
```

All of these parameters must be specified together. None of the parameters allow the use of a pipe character (|) to specify multiple sets of arguments.

The definition of the parameters is as follows:

- **Nrk** sets the search interface to use in the navigation query for a record search. Only search interfaces can be specified; Oracle Commerce properties and dimensions cannot be used. Note that the search interface does not need to have a relevance ranking strategy defined in it.
- **Nrt** sets one or more terms that will be used by the relevance ranking module to order the records. For multiple terms, each term is delimited by a plus (+) sign. Note that these relevance ranking terms can be different from the search terms (as set by the **Ntt** parameter, for example).
- **Nrr** sets the relevance ranking strategy to be used to rank the results of the record search. For a list of valid module names to use in the *relrank-strategy* argument, see the "Controlling relevance ranking at the query level" topic.
- **Nrm** sets the relevance ranking match mode to be used to rank the results of the record search. With the exception of MatchBoolean, all of the search mode values listed in "Using Search Modes" are valid for use with the **Nrm** parameter. Attempting to use MatchBoolean with the **Nrm** parameter will cause the record search results to be returned without relevance ranking and a warning to be issued to the dgraph log.

All four parameters link to the Java `ENEQuery.setNavRelRankERecRank()` method and the .NET `ENEQuery.NavRelRankERecRank` property. Note that these parameters have a dependency on the **N** parameter, because a navigation query is being performed.

Because the **Nrt** parameter lets you specify relevance ranking terms (and not search terms), you have the freedom to perform a record search based on one set of terms (for example, *merlot* and *2003*) and then have the record set ordered by another set of terms (for example, *pear*). This behavior is different from that of the **Ntx** parameter, which uses the terms of the **Ntt** parameter to order the record set (in other words, the same set of search terms are also used to perform relevance ranking).

The following is an example of using these parameters:

```
<application>?N=0&Ntk=P_Description&Ntt=sonoma
&Nrk=All&Nrt=citrus&Nrr=maxfield&Nrm=matchall
```

In the example, a record search is first performed for the word *sonoma* against the *P_Description* property. Then **Nrk** specifies that the search interface named *All* be used. **Nrr** specifies that the Maxfield relevance ranking module use the word *citrus* (specified via **Nrt**) as the term by which the records are ordered, using the match mode specified by **Nrm**.



Note: The **Nrk**, **Nrt**, **Nrr**, and **Nrm** parameters take precedence over the **Ntk**, **Ntt**, and **Ntx** parameters. That is, if both sets of parameters are used in a query, the relevance ranking strategy specified by the **Nrr** parameter will be used to order the records.

Using relevance ranking methods

Because relevance ranking only affects the order of results (and not the content of results), there are no special objects or rendering techniques associated with relevance ranking.

Remember, though, that this ordering can have significant impact on how quickly results are rendered.

Relevance Ranking sample scenarios

This section contains two examples of relevance ranking behavior to further illustrate the capabilities of this feature.

In the first example, we first look at the effects of various relevance ranking strategies on a small sample data set that supports record search, examining the range of possible result orderings possible using only a limited set of ranking modules.

In the second example, we look at how adding a simple relevance ranking strategy can affect user results in the reference implementation.



Note: These extremely simple scenarios are provided for illustrative purposes only. For more realistic examples, see the "Recommended strategies" topic. Also note that in many relevance ranking scenarios you can set `considerFieldRanks` for tie breaking. This setting is not useful for Dimension search because all searchable dimension value synonyms are in the same field.

Example 1: Using a small data set

This scenario shows the effects of various relevance ranking strategies on a small data set.

This example illustrates the richness of relevance ranking tuning possible with the modular Oracle Commerce relevance ranking system: using two modules on a data set of three records, we found that all four possible combinations of the modules into strategies resulted in different orderings, all of which were different from the default ordering.

The example uses the following example record set:

Record	Title property	Author property
1	Great Short Stories	Mark Twain and other authors
2	Mark Twain	William Lyon Phelps
3	Tom Sawyer	Mark Twain

Creating the search interface in Developer Studio

In Oracle Commerce Developer Studio, we have defined a search interface named Books that contains both Title and Author properties. The relevance rank is determined by the order in which the dimensions or properties appear in the Selected Members list.

Assume that we have not defined an explicit default sort order for the records, in which case their default order is determined by the system.

Without relevance ranking

Suppose that the user enters a record search query against the Books search interface for *Mark Twain*. All three of the records are matches, because each record has at least one searchable property value containing at least one occurrence of both the words Mark and Twain. But in what order should the results be presented to the application user? Without relevance ranking enabled, the results are returned in their default order: 1, 2, 3.

If relevance ranking were enabled, the order depends on the relevance ranking strategy selected.

With an Exact ranking strategy

Suppose we have selected the Exact relevance ranking strategy, either by assigning this as the default strategy for the Books search interface in Developer Studio or by using URL-level search options.

In this case, the order of results would be based only on whether results were Exact, Phrase, or other matches. Because records 2 and 3 have properties whose complete values exactly match the user query *Mark Twain*, these results would be returned ahead of record 1, with the tie being broken by the default sort set by the system (remember that we have not defined a default sort).

With an Exact ranking strategy and the `considerFieldRanks` parameter

Suppose we have selected the Exact relevance ranking strategy and also specified the `considerFieldRanks` parameter in the query URL. Also, suppose that the Title property has a higher field rank value than Author for any search matches.

In this case, the order of results would be based only on whether results were Exact, Phrase, or other matches. Because records 2 and 3 have properties whose complete values exactly match the user query *Mark Twain*, these results would be returned ahead of record 1. And further, because we specified `considerFieldRanks`, record 2 would be returned ahead of record 3.

With a Field ranking strategy

Now, assume that we have selected the Field relevance ranking strategy.

The order of results would be based only on which property caused the match, with Author matches being prioritized over Title matches. Because records 1 and 3 match on Author, these are returned ahead of record 2 (again, with ties broken by the default sort imposed by the system).

With a Field,Exact ranking strategy

Now, consider using a combination of these two strategies: Field,Exact.

In this case, the primary sort is determined by the first module, Field, which again dictates that records 1 and 3 should be returned ahead of record 2. But in this case, the Field tie between records 1 and 3 is resolved by the Exact module, which prioritizes record 3 ahead of record 1. Thus, the order of results returned is: 3, 1, 2.

With an Exact,Field ranking strategy

Finally, consider combining the same two modules but in a different priority order: Exact,Field.

In this case, the primary sort is determined by the Exact module, which again prioritizes records 2 and 3 ahead of record 1. In this case, the Exact tie between records 2 and 3 is resolved by the Field module, which orders record 3 ahead of record 2 because record 3 is an Author match. Thus, the order of results returned is: 3, 2, 1.

Example 2: UI reference implementation

This scenario shows how adding a relevance ranking module can change the order of the returned records.

This example, which is somewhat more realistically scaled, uses the sample wine data in the UI reference implementation. It demonstrates how relevance ranking can affect the results displayed to your users.

In this scenario, we use the thesaurus and relevance ranking features to enable end users' access to Flavor results similar to the one they searched on, while still seeing exact matches first.

First, in Developer Studio, we establish the following two-way thesaurus entries:

```
{ cab : cabernet }
{ cinnamon : spice : nutmeg }
{ tangy : tart : sour : vinegary }
{ dusty : earthy }
```

Before applying these thesaurus equivalencies, if we search on the Dusty flavor, 83 records are returned, and if we search on the Earthy flavor, 3,814 records are returned.

After applying these thesaurus equivalencies, if we search on the Dusty property, results for both Dusty and Earthy are returned. (Because some records are flagged with both the Dusty and Earthy descriptors, the number of records is not an exact total of the two.)

Wine (by order returned)	Relevant property
A Tribute Sonoma Mountain	Earthy
Against the Wall California	Earthy
Aglianico Irpinia Rubrato	Dusty
Aglianico Sannio	Earthy

Because the application is sorting on Name in ascending order, the Dusty and Earthy results are intermingled. That is, the first two results are for Earthy and the third is for Dusty, even though we searched on Dusty, because the two Earthy records came before the Dusty one when the records were sorted in alphabetical order.

Now, suppose that while we want our users to see the synonymous entries, we want records that exactly match the search term Dusty to be returned first. We therefore would use the Interpreted ranking module to ensure that outcome.

Wine (by order returned)	Relevant property
Aglianico Irpinia Rubrato	Dusty
Bandol Cuvee Speciale La Miguoa	Dusty
Beaujolais-Villages Reserve du Chateau de Montmelas	Dusty
Beauzeaux Winemaker's Collection Napa Valley	Dusty

With the Interpreted ranking strategy, the results are different. When we search on Dusty, we see the records that matched for Dusty sorted in alphabetical order, followed by those that matched for Earthy. The wine Aglianico Irpinia Rubrato, which was returned third in the previous example, is now returned first.

Recommended strategies

This section provides some recommended strategies that depend on the implementation type.

Relevance ranking behavior is complex and powerful and requires careful, iterative development. Typically, selection of the ideal relevance ranking strategy for a given application depends on extensive experimentation during application development. The set of possible result ranking strategies is extremely rich, and because setting ranking strategies is highly dependent on the quantity and type of data you are working with, a strategy that works well in one situation could be unsatisfactory in another.

For this reason, Oracle provides recommended strategies for different types of implementations and suggests that you use them as a point of departure in creating your own strategies. The following sections describe recommended general strategies for each product in detail.



Note: These recommendations are not meant to overrule custom strategies developed for your application by Oracle Services.

Testing your strategies

When testing your own strategies, it is a good idea to try searching on diverse examples: single word terms, multi-word terms that you know are an exact match for records in your data, and multi-word terms that contain additional words as well as the ones in your data. In this way you will see the full range of relevance ranking effects.

Recommended strategy for retail catalog data

This topic describes a good starting strategy to try if you are a retailer working with a catalog data set.

The strategy assumes the following:

- The search mode is MatchAllPartial. By using this mode, you ensure that a user's search would return a two-words-out-of-five match as well as a four-words-out-of-five match, just at a lower priority.
- The strategy is based on a search interface with members such as Category, Name, and Description, in that order. The order is significant because a match on the first member ranks more highly than a cross-field match or match on the second or third member. (For details, see "About Search Interfaces").

The strategy is as follows:

- NTerms
- MaxField
- Glom
- Exact
- Static

The modules in this strategy work like this:

1. NTerms, the first module, ensures that in a multi-word search, the more words that match the better.
2. Next, MaxField puts cross-field matches as high in priority as possible, to the point where they could tie with non-cross-field matches.
3. The next module, Glom, decomposes cross-field matches, effectively breaking any ties resulting from MaxField. Together, MaxField and Glom provide the proper ordering, depending upon what matched.
4. Applying the Exact module means that an exact match in a highly-ranked member of the search interface is placed higher than a partial or cross-field match.
5. Optionally, the Static module can be used to sort remaining ties by criteria such as Price or SalesRank.

Recommended strategy for document repositories

This topic describes a good starting strategy to try if you are working with a document repository.

The strategy assumes the following:

- The search mode is MatchAllPartial. By using this mode, you ensure that a user's search would return a two-words-out-of-five match as well as a four-words-out-of-five match, just at a lower priority.

- The strategy is based on a search interface with members such as Title, Summary, and DocumentText, in that order. The order is significant because a match on the first member ranks more highly than a cross-field match or match on the second or third member.

The strategy is as follows:

- NTerms
- MaxField
- Glom
- Phrase (with or without approximate matching enabled)
- Static

The modules in this strategy work like this:

1. NTerms, the first module, ensures that in a multi-word search, the more words that match the better.
2. Next, MaxField puts cross-field matches as high in priority as possible, to the point where they could tie with non-cross-field matches.
3. The next module, Glom, decomposes cross-field matches, effectively breaking any ties resulting from MaxField. Together, MaxField and Glom provide the proper ordering, depending upon what matched.
4. Applying the Phrase module ensures that results containing the user's query as an exact phrase are given a higher priority than matching containing the user's search terms sprinkled throughout the text.
5. Optionally, the Static module can be used to sort the remaining ties by criteria such as ReleaseDate or Popularity.

Performance impact of Relevance Ranking

Relevance ranking can impose a significant computational cost in the context of affected search operations (that is, operations where relevance ranking is actually enabled).

You can minimize the performance impact of relevance ranking in your implementation by making module substitutions when appropriate, and by ordering the modules you do select sensibly within your relevance ranking strategy.

Making module substitutions

Because of the linear cost of relevance ranking in the size of the result set, the actual cost of relevance ranking depends heavily on the set of ranking modules used.

In general, modules that do not perform text evaluation introduce significantly lower computational costs than text-matching-oriented modules.

Although the relative cost of the various ranking modules is dependent on the nature of your data and the number of records, the modules can be roughly grouped into four tiers:

- Exact is very computationally expensive.
- Proximity, Phrase with Subphrase or Query Expansion options specified, and First are all high-cost modules, presented in the order of decreasing cost.
- WFreq can also be costly in some situations.
- The remaining modules (Static, Phrase with no options specified, Freq, Spell, Glom, Nterms, Interp, Numfields, Maxfields and Field) are generally relatively cheap.

In order to maximize the performance of your relevance ranking strategy, consider a less expensive way to get similar results. For example, replacing Exact with Phrase may improve performance in some cases with relatively little impact on results.



Note: Choose the set of modules used for relevance ranking most carefully when the data set is large or contains large/offline file content that is used for search operations.

Ordering modules sensibly

Relevance ranking modules are only evaluated as needed.

When higher-priority ranking modules determine the order of records, lower-priority modules do not need to be calculated. This can have a dramatic impact on performance when higher-cost modules have a lower priority than a lower-cost module.

While you have the freedom to order modules as you like, for best performance, make sure that the cheaper modules are placed before the more expensive ones in your strategy.

Record Boost and Bury

Record boost and bury is a mechanism by which specific records can be made to rank higher or lower than other records in a results list.

About the record boost and bury feature

Record boost makes specific records rank higher than others. **Record bury** makes specific records rank lower than others.

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

Feature assumptions and limitations

The following applies to the record boost and bury feature:

- EQL (Oracle Commerce 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 supported by the Java and .NET versions of the Presentation API.

Some use-case assumptions are:

- This feature is expected to be used predominately with Oracle Commerce Workbench .
- A common usage pattern will be to specify the records to be boosted/buried dynamically (per-query). Typically, this will be done through Workbench and Experience Manager, 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.

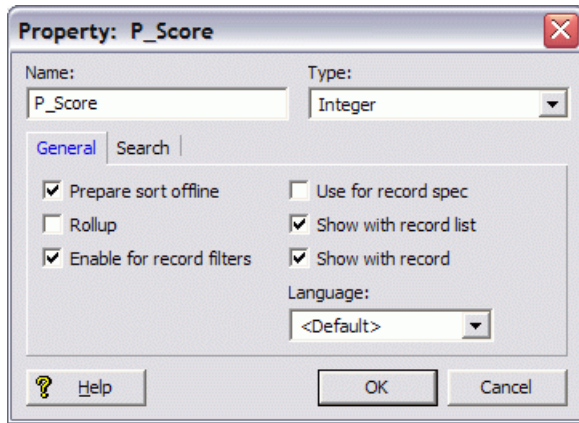
Enabling properties for filtering

Oracle Commerce 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 Oracle Commerce 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 `ENQuery.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],*,collection()/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, through 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 Oracle Commerce Query Language" chapter in the *MDEX Engine Developer's 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 Oracle Commerce records

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

The `Endeca.stratify` relevance ranking module can be specified in record search options, through 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. 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 (that is, 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 how 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);
```


Part 6

Content Spotlighting and Merchandizing

- *Implementing User Profiles*

Implementing User Profiles

Recommended practice is to use the Oracle Experience Manager, rather than directly managing business rules and user profiles, in all new application development. For information about the Experience Manager, refer to the *Workbench User's Guide*.

About user profiles

A user profile is a character-string-typed name that identifies a class of end users.

User profiles enable applications built on the Oracle Commerce Information Access Platform to tailor the content displayed to an end user based on that user's identity.

User profiles can be used to trigger dynamic business rules, where such rules are optionally constructed with an additional trigger attribute corresponding to a user profile. Oracle Commerce Guided Search can accept information about the end user, and use that information to trigger pre-configured rules and behaviors.

You set up user profiles in Developer Studio. Both Developer Studio and Oracle Commerce Workbench allow a user profile to be associated with a business rule's trigger.

This feature discusses how you create user profiles and then implement them as dynamic business rule triggers. Before reading further, make sure you are comfortable with the information in the "Promoting Records with Dynamic Business Rules" section.



Note: Each business rule is allowed to have at most one user profile trigger.

Profile-based trigger scenario

This topic shows how a dynamic business rule would utilize a user profile.

In the following scenario, an online clothing retailer wants to set up a dynamic business rule that says: "For young women who are browsing stretch t-shirts, also recommend cropped pants." We follow the shopping experience of a customer named Jane.

In order to set up this rule, a few configuration steps are necessary:

1. In Oracle Commerce Developer Studio, the retailer creates a user profile called `young_woman`, which corresponds to the set of customers who are female and are between the ages of 16 and 25.

2. In Oracle Commerce Workbench, a dynamic business rule that uses the profile as a trigger is created, as shown below. No complex Boolean logic programming is necessary here. The business user simply selects a user profile from a set of available profiles to create the business rule.

```
young_woman X DVAL(stretch t-shirt) => DVAL(cropped pants)
```

3. In the Web application that's driving the customer's experience, there needs to be logic that identifies the user and tests to see if he or she meets the requirements to be classified as a `young_woman`. Alternatively, the profile `young_woman` may already be stored along with Jane's information (such as age, address, and income) in a database or LDAP server.

The user's experience would go something like this:

1. Jane accesses the clothing retailer's Web site and is identified by a cookie on her computer. By looking up a few database tables, the application knows that it has interacted with her before. The database indicates that she is 19 years old and female.

At this point, the database may also indicate the user profiles that she belongs to: `young_woman`, `r_and_b_music_fan`, `college_student`. Alternatively, the application logic may test against her information to see which profiles she belongs to, as follows: "Jane is between 16 and 25 years old and she is female, so she belongs in the `young_woman` profile."

2. As Jane is browsing the site, the Oracle Commerce MDEX Engine is driving her catalog experience. As each query is being sent to the Oracle Commerce MDEX Engine, it is augmented with user profile information. Here is some sample Java code:

```
profileSet.add("young_woman");
eneQuery.setProfiles(profileSet);
```

3. As Jane clicks on a stretch t-shirt link, the Oracle Commerce MDEX Engine realizes that a dynamic business rule has been triggered: `young_woman X DVAL(stretch t-shirt)`. Therefore, it returns a cropped pants record in one of the dynamic business rule zones.
4. Jane sees a picture of cropped pants in a box labeled, "You also might like..."

User profile query parameters

There are no URL MDEX query parameters associated with user profiles.

In many live application scenarios, the URL query is exposed to the end user, and it is usually not appropriate for end users to see or change the user profiles with which they have been tagged.

API objects and method calls

These Java and .NET code samples demonstrate how to implement user profiles in the Web application.

In the following code samples, the application recognizes the end user as Jane Smith, looks up some database tables and determines that she is 19 years old, female, a college student and likes R&B music. These characteristics map to the following Oracle Commerce user profiles created in Oracle Commerce Developer Studio:

- `young_woman`
- `r_and_b_music_fan`
- `college_student`

User profiles can be any string. The user profiles supplied to `ENEQuery` must exactly match those configured in Oracle Commerce Developer Studio.

Java example of implementing user profiles

```
// User profiles can be any string. The user profiles must
// exactly match those configured in Developer Studio.
// Add this import statement at the top of your file:
// import java.util.*;
Set profiles = new HashSet();
// Collect all the profiles into a single Set object.
profiles.add("young_woman");
profiles.add("r_and_b_music_fan");
profiles.add("college_student");
// Augment the query with the profile information.
eneQuery.setProfiles(profiles);
```

.NET example of implementing user profiles

```
// Make sure you have the following statement at the top
// of your file:
// using System.Collections.Specialized;
StringCollection profiles = new StringCollection();
// Collect all the profiles into a single StringCollection object.
profiles.Add("young_woman");
profiles.Add("r_and_b_music_fan");
profiles.Add("college_student");
// Augment the query with the profile information.
eneQuery.Profiles = profiles;
```

Performance impact of user profiles

An application using this feature may experience additional memory costs due to user profiles being set in an `ENEQuery` object.

In addition, the application may require additional Java `ENEConnection.query()` or .NET `HttpENEConnection.Query()` response time, because the MDEX Engine must do additional work to receive profile information and check if business rules fire. However, in typical application scenarios that set one to five user profile strings of at most 20 characters in the `ENEQuery` object, the performance impact is insignificant.

Part 7

Understanding and Debugging Query Results

- *Using Why Match*
- *Using Word Interpretation*
- *Using Why Rank*
- *Using Why Precedence Rule Fired*

Chapter 39

Using Why Match

This section describes the tasks involved in implementing the Why Match feature of the Oracle Commerce MDEX Engine.

About the Why Match feature

The Why Match functionality allows an application developer to debug queries by examining which property value of a record matched a record search query and why it matched.

With Why Match enabled in an application, records returned as part of a record search query are augmented with extra dynamically generated properties that provide information about which record properties were involved in search matching.

Enabling Why Match

You enable Why Match on a per-query basis using the `Nx` (Navigation Search Options) query parameter. No Developer Studio configuration or `dgraph` flags are required to enable this feature.

However, because Why Match applies only to record search navigation requests, dynamically-generated properties only appear in records that are the result of a record search navigation query. Records in non-search navigation results do not contain Why Match properties.

Why Match API

The MDEX Engine returns match information for each record as a `DGraph.WhyMatch` property in the search results.

The following code samples show how to extract and display the `DGraph.WhyMatch` property from a record.

Java example

```
// Retrieve properties from record
PropertyMap propsMap = rec.getProperties();
// Get the WhyMatch property value
String wm = (String) propsMap.get("DGraph.WhyMatch");
// Display the WM value if one exists
if (wm != null) {
```

```

    %>This record matched on <%= wm %>
    <%
}

```

.NET example

```

// Retrieve properties from record
PropertyMap propsMap = rec.Properties;
// Get the WhyMatch property value
String wm = propsMap["DGraph.WhyMatch"].ToString();
// Display the WM value if one exists
if (wm != null) {
    %>This record matched on <%= wm %>
    <%
}

```

Why Match property format

The `DGraph.WhyMatch` property value has a three-part format that is made up of a list of fields where the terms matched, a list of the terms that matched, and several possible query expansions that may have been applied to the during processing.

The `DGraph.WhyMatch` property is returned as a JSON object with the following format :

```

[ {fields: [ <FieldName>, <FieldName>, ... ], terms: [
    {term:<TermName>, expansions:[ {type:<TypeName>},
    {type:<TypeName>}, ... ]},
    {term:<TermName>, expansions:[ {type:<TypeName>},
    {type:<TypeName>}, ... ]}} ]

```

where the supported expansion types (i.e. the `<TypeName>` values) are as follows:

- **Stemming** – returned results based on the stemming dictionaries available in the MDEX Engine.
- **Thesaurus** – returned augmented results based on thesaurus entries added in Developer Studio or Oracle Commerce Workbench.
- **Spell-corrected** – returned spell-corrected results using application-specific dictionary words.

The availability of these values depends on which search features have been enabled in the MDEX Engine.

For example, suppose there is a matchpartial query for "nueve uno firefighter" that produces a single-field match in "Spanish", a cross-field match in Spanish and English (i.e. "one" appears in English via thesaurus from uno), and firefighter is not in any field. The following `DGraph.WhyMatch` property value is returned:

```

[ {fields:[Spanish], terms:[ {term:nueve,expansions:[ ]},
    {term:uno,expansions:[ ]}} ],
  {fields:[Spanish,English], terms:[ {term:nueve,expansions:[ ]},
    {term:uno, expansions:[ {type:Thesaurus} ]}} ] ]

```

Why Match performance impact

The response times for MDEX Engine requests that include Why Match properties are more expensive than requests without this feature. The performance cost increases as the number of records returned with the `DGraph.WhyMatch` property increases.

This feature is intended for development environments to record matching. The feature is not intended for production environments and is not particularly optimized for performance.

Chapter 40

Using Word Interpretation

This section describes the tasks involved in implementing the Word Interpretation feature of the Oracle Commerce MDEX Engine.

About the Word Interpretation feature

The Word Interpretation feature reports word or phrase substitutions made during text search processing.

The Word Interpretation feature is particularly useful for highlighting variants of search keywords that appear in displayed search results. These variants may result from stemming, thesaurus expansion, or spelling correction.

Implementing Word Interpretation

The `--wordinterp` flag to the `dgraph` command must be set to enable the Word Interpretation feature.

The Word Interpretation feature does not require any work in Developer Studio. There are no Dgidx flags necessary to enable this feature, nor are there any MDEX Engine URL query parameters.

Word Interpretation API methods

The MDEX Engine returns word interpretation match information in `ESearchReport` objects.

This word interpretation information is useful for highlighting or informing users about query expansion.

The Java `ESearchReport.getWordInterps()` method and the .NET `ESearchReport.WordInterps` property return the set of word interpretations used in the current text search. Each word interpretation is a key/value pair corresponding to the original search term and its interpretation by the MDEX Engine.

In this thesaurus example, assume that you have added the following one-way thesaurus entry:

```
cab > cabernet
```

If a search for the term *cab* finds a match for *cabernet*, a single word interpretation will be returned with this key/value pair:

```
Key="cab" Value="cabernet"
```

When there are multiple substitutions for a given word or phrase, they are comma-separated; for example:

```
Key="cell phone" Value="mobile phone, wireless phone"
```

In this Automatic Phrasing example, a search for the terms *Napa Valley* are automatically phrased to "*Napa Valley*". A key/value word interpretation is returned with the original search terms as the key and the phrased terms in double quotes as the value.

```
Key=Napa Valley Value="Napa Valley"
```

The following code snippets show how to retrieve word interpretation match information.

Java example

```
// 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 searchReport = (ESearchReport)recSrchrpts.get(searchKey);
            // Get the PropertyMap of word interpretations
            PropertyMap wordMap = searchReport.getWordInterps();
            // Get property iterator
            Iterator props = wordMap.entrySet().iterator();
            // Loop over properties
            while (props.hasNext()) {
                // Get individual property
                Property prop = (Property)props.next();
                String propKey = (String)prop.getKey();
                String propVal = (String)prop.getValue();
                // Display word interpretation information
                %<tr>
                <td>Original term: <%= propKey %></td>
                <td>Interpreted as: <%= propVal %></td>
                </tr><%
            }
        }
    }
}
```

.NET example

```
// 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 first Search Report object
        IDictionaryEnumerator ide = recSrchrpts.GetEnumerator();
        ide.MoveNext();
        ESearchReport searchReport = (ESearchReport)ide.Value;
        // Get the PropertyMap of word interpretations
        PropertyMap wordMap = searchReport.WordInterps;
        // Get property iterator
        System.Collections.IList props = wordMap.EntrySet;
        // Loop over properties
        for (int j = 0; j < props.Count; j++) {
            // Get individual property
        }
    }
}
```

```
Property prop = (Property)props[j];
String propKey = prop.Key.ToString();
String propVal = prop.Value.ToString();
// Display word interpretation information
%<tr>
  <td>Original term: <%= propKey %></td>
  <td>Interpreted as: <%= propVal %></td>
</tr><%
}
}
```

Troubleshooting Word Interpretation

This topic provides some corrective solutions for word interpretation problems.

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.

Using Why Rank

This section describes the tasks involved in implementing the Why Rank feature of the Oracle Commerce MDEX Engine.

About the Why Rank feature

The Why Rank feature returns information that describes which relevance ranking modules were evaluated during a query and describes how query results were ranked. This information allows an application developer to debug relevance ranking behavior.

With Why Rank enabled in an application, the MDEX Engine returns records that are augmented with additional dynamically generated properties. The MDEX Engine also returns summary information (in a `Supplement` object) about relevance ranking for a query. The properties provide information that describe which relevance ranking modules ordered the results and indicate why a particular record was ranked in the way that it was.

Enabling Why Rank

You enable Why Rank on a per-query basis using the `Nx` (Navigation Search Options) query parameter or the `Dx` (Dimension Search Options) query parameter. No Developer Studio configuration or `dgraph` flags are required to enable this feature.

Why Rank API

The MDEX Engine returns relevance ranking information as a `DGraph.WhyRank` property on each record in the search results. The MDEX Engine also returns summary information for all record results in a `Supplement` object. (Note that the information available in a `Supplement` object is not available if you are using the MAX API.)

Per record match information

The following code samples show how to extract and display the `DGraph.WhyMatch` property from a record.

Java example

```
// Retrieve properties from record
PropertyMap propsMap = rec.getProperties();

// Get the WhyRank property value
String wr = (String) propsMap.get("DGraph.WhyRank");

// Display the WR value if one exists
if (wr != null) {
    %>This record was ranked by <%= wr %>
    <%
}
}
```

.NET example

```
// Retrieve properties from record
PropertyMap propsMap = rec.Properties;

// Get the WhyRank property value
String wr = propsMap["DGraph.WhyRank"].ToString();

// Display the WR value if one exists
if (wr != null) {
    %>This record was ranked by <%= wr %>
    <%
}
}
```

Summary match information

The Supplement object contains a "Why Summaries" property whose value is general summary information for ranking of all the records returned in a query. This information includes the number of relevance ranking modules that were evaluated, the number of strata per module, processing time per module, and so on.

Why Rank property format

The `DGraph.WhyRank` property value has a multi-part format that is made up of a list of relevance ranking modules that were evaluated and strata information for each module. Strata information includes the evaluation time, rank, description, records per strata, and so on.

The `DGraph.WhyRank` property is returned as a JSON object with the following format:

```
[
  { "<RankerName>" : { "evaluationTime" : "<number>", "stratumRank" : "<number>",
    "stratumDesc" : "<Description>", "rankedField" : "<FieldName>" }},
  ...
]
```

where the `<RankerName>` values are any of supported relevance ranking modules. The specific number of `<RankerName>` values depends on the relevance ranking modules you enabled in the MDEX Engine and how many of them were used to evaluate the current record.



Note: If a query produces only one record in a result set, the `DGraph.WhyRank` property is empty because no relevance ranking was applied.

Here is an example of a query and a `DGraph.WhyRank` property from a record in the result set. Suppose there is a query submitted to an MDEX Engine using the following query parameters:

`N=0&Ntk=NoEssay&Ntt=one+two&Ntx=rel+phrase(considerFieldRanks)&Nx=whyrank`. The query produces a result set where one of the records contains the following `DGraph.WhyRank` property:

```
<Property Key="DGraph.WhyRank" Value="[ { "phrase" : { "evaluationTime" : "0",
"stratumRank" : "20", "stratumDesc" : "phrase match", "rankedField" : "English"
}} ]">
```

Result information for relevance ranking modules


In addition to the basic reporting properties that are common to each `DGraph.WhyRank` property, there are also optional reporting properties that may be included in `DGraph.WhyRank` depending on the relevance ranking module.

The basic reporting properties in `DGraph.WhyRank` that are common to all relevance ranking modules include:

- `evaluationTime` - the time spent evaluating this relevance ranking module.
- `stratumRank` - a value indexing which stratum a record is placed in.
- `stratumDesc` - the description of the relevance ranking module (often, the name of the module, or a description of options for the module).

The following table lists the optional reporting properties that are specific to each relevance ranking module.

Relevance Rank Module Name	Additional DGraph.WhyRank Properties
Exact	<code>rankedField</code> - field name for the highest ranked exact or subphrase match described in <code>stratumDesc</code> .
Field	<code>rankedField</code> - field name for the highest ranked field match.
First	<code>rankedField</code> - field name of the highest ranked field described in <code>stratumDesc</code> .
Freq	<code>perFieldCount</code> - field-by-field count of occurrences in the format "<X1> in <field1-name>, <X2> in <field2-name>, ...".
Glom	None.
Interp	<code>rankedField</code> - field name of the highest ranked field described in <code>stratumDesc</code> .
MaxFields	<code>rankedField</code> - field name of the highest ranked field described in <code>stratumDesc</code> .
NTerms	None.
NumFields	<code>fieldsMatched</code> - if <code>considerFieldRanks</code> is enabled for the module, then <code>fieldsMatched</code> is a comma-separated list of: <field-name> + "(" + <field-rank> + ")". Otherwise, <code>fieldsMatched</code> is a comma-separated list of the field names that matched.
Phrase	<code>rankedField</code> - field name of the highest ranked field (if a phrase match).
Proximity	<code>rankedField</code> - field name of the highest ranked field (if a field match).
Spell	<code>rankedField</code> - field name of a field match that is not a spell corrected match.
Static	<ul style="list-style-type: none"> • <code>fieldCompared</code> - name of field sorted by. If there are multiple fields, names are pipe ' ' delimited.

Relevance Rank Module Name	Additional DGraph.WhyRank Properties
	<ul style="list-style-type: none"> • <code>directionCompared</code> - direction ("ascending" or "descending") of the sort. If there are multiple fields, directions are pipe ' ' delimited • <code>fieldType</code> - corresponding field type ("integer", "dimension", "string", etc). If there are multiple fields, types are ' ' delimited. <p> Note: The Static module does not return either the <code>evaluationTime</code> or the <code>stratumRank</code> properties.</p>
Stratify	None.
Stem	<code>rankedField</code> - field name of a field match that is not a stemmed match.
Thesaurus	<code>rankedField</code> - field name of a field match that is not a thesaurus match
WeightedFreq	None.

Why Rank performance impact

The response times for MDEX Engine requests that include Why Rank properties are more expensive than requests without this feature. The performance cost increases as the number of records returned with the `DGraph.WhyRank` property increases.

This feature is intended for development environments to troubleshoot relevance ranking. The feature is not intended for production environments and is not particularly optimized for performance.

Using Why Precedence Rule Fired

This section describes the tasks involved in implementing the Why Precedence Rule Fired feature of the Oracle Commerce MDEX Engine.

About the Why Precedence Rule Fired feature

The Why Precedence Rule Fired feature returns information that explains why a precedence rule fired. This information allows an application developer to debug how dimensions are displayed using precedence rules.

With the feature enabled in an application, the root dimension values that the MDEX Engine returns are augmented with additional dynamically generated properties. The properties provide information that describe how the precedence rule was triggered (explicitly or implicitly), which dimension ID and name triggered the precedence rule, and the type of precedence rule (standard, leaf, or default).

Enabling Why Precedence Rule Fired

You enable Why Precedence Rule Fired on a per-query basis using the `Nx` (Navigation Search Options) query parameter. No Developer Studio configuration or `dgraph` flags are required to enable this feature.

Why Precedence Rule Fired API

The MDEX Engine returns information about why a precedence rule fired as a `DGraph.WhyPrecedenceRuleFired` property on each root dimension value.

The following code samples show how to extract and display the `DGraph.WhyPrecedenceRuleFired` property from a root dimension value.

Java example

```
// Retrieve the results object.
Navigation result = results.getNavigation();

// Retrieve the refinements.
DimensionList l = result.getRefinementDimensions();

// Retrieve the dimension with ID 80000.
```

```

Dimension d = l.getDimension(800000);

// Retrieve the root dval for the dimension.
DimVal root = d.getRoot();
PropertyMap propsMap = root.getProperties();

// Get the WhyPrecedenceRuleFired property value
String wprf = (String) propsMap.get("DGraph.WhyPrecedenceRuleFired");

// Display the value if one exists
if (wprf != null) {
    //Do something
}

```

.NET example

```

// Retrieve the results object.
Navigation result = results.Navigation;

// Retrieve the refinements.
DimensionList l = result.RefinementDimensions;

// Retrieve the dimension with ID 80000.
Dimension d = l.GetDimension(800000);

// Retrieve the root dval for the dimension.
DimVal root = d.Root;
PropertyMap propsMap = root.Properties;

// Get the WhyPrecedenceRuleFired property value
String wprf = propsMap["DGraph.WhyPrecedenceRuleFired"].ToString();

// Display the value if one exists
if (wprf != null) {
    //Do something
}

```

Why Precedence Rule Fired property format

The `DGraph.WhyPrecedenceRuleFired` property value has a multi-part format that is made up of a list of trigger reasons and trigger values that were evaluated for each precedence rule.

The `DGraph.WhyPrecedenceRuleFired` property is returned as a JSON object with the following format:

```

[
  {
    "triggerReason" : "<Reason>",
    "triggerDimensionValues" : ["<DimensionID>", ...
    ],
    "ruleType" : "<Type>",
    "sourceDimension" : "<DimensionName>",
    "sourceDimensionValue" : "<DimensionID>" },
  ...
]

```

The following table describes the reporting values in the `DGraph.WhyPrecedenceRuleFired` property. The specific reporting values depend on the precedence rules in the MDEX Engine and how many rules the MDEX Engine evaluated for the current set of available refinement dimensions.

Reporting Value	Description
<Reason>	<p>The triggerReason can have any of the following values:</p> <ul style="list-style-type: none"> • explicit - The precedence rule triggered because a user explicitly selected a trigger dimension value in a navigation query. The triggerDimensionValues is a list of dimension IDs that triggered the rule. • explicitSelection - The precedence rule triggered because an user explicitly selected a target dimension value, and there are more refinements available. The triggerDimensionValues is a list of dimension IDs that triggered the rule. • implicit - The precedence rule triggered because a user implicitly selected a trigger dimension value. For example, it is implicit because a user could select a dimension value that resulted from a text search rather selecting a refinement from a navigation query. The triggerDimensionValues is a list of dimension IDs that triggered the rule. • implicitSelection - The precedence rule triggered because a user implicitly selected a target dimension value, and there are more refinements available. • default - The precedence rule triggered because it is a default rule that is set up to always trigger. (Forge creates default rules during automatic property mapping.)
<DimensionID>	The triggerDimensionValues is followed by a list of integers representing the dimension IDs.
<Type>	<p>The ruleType can have any of the following values:</p> <ul style="list-style-type: none"> • standard - Standard precedence rules display the target dimension if the source dimension value or its descendants are in the navigation state. • leaf - Leaf precedence rules display the target dimension only after leaf descendants of the source dimension value have been selected.
<DimensionName>	A string representing the name of the dimension.

The `DGraph.WhyPrecedenceRuleFired` property may contain any number of `triggerReason` reporting values. However, there is one exception in the case where the value of `triggerReason` is `default`. In that case, there would be a single `triggerReason` value.

Here is an example query that contains at least the following two URL query parameters:

`N=310002&Nx=whyprecedencerulefired`. The value of 310002 is the dimension value ID that triggers a precedence rule for dimension 300000. The query produces a result with a root dimension value that contains the following `DGraph.WhyPrecedenceRuleFired` property:

```
<Dimension Id=300000 Name=Number of Digits>
  <Root>
    <DimVal Name="Number of Digits" Id=300000>
      <PropertyList Size=1>
        <Property Key="DGraph.WhyPrecedenceRuleFired" Value="[ { "trigger-
```

```
Reason" : "explicitSelection", "triggerDimensionValues" : [310002] } ]">
```

Performance impact of Why Precedence Rule Fired

The Why Precedence Rule Fired feature is intended for a production environment. The response times for MDEX Engine requests that include `DGraph.WhyPrecedenceRuleFired` properties are slightly more expensive than requests without this feature. In general, the feature adds performance throughput costs that are typically observed to be less than 5%.

Appendix A

Oracle Commerce URL Parameter Reference

This appendix provides a reference to the URL-based syntax for navigation, record, aggregated record, and dimension search queries.

About the Oracle Commerce URL query syntax

The Oracle Commerce query syntax defines how the client browser communicates with the Presentation API.

This appendix describes two methods:

- URL parameters
- `ENEQuery` setter methods (Java) and properties (.NET)

URL parameter description format

The tables in this appendix describe the Oracle Commerce 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 <code>ENEQuery</code> Java setter method for the parameter.
.NET setter property	The corresponding <code>ENEQuery</code> .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	<code>ENEQuery.setNavDescriptors()</code>
.NET setter property	<code>ENEQuery.NavDescriptors</code>
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	<code>ENEQuery.setNavAggrERecsOffset()</code>
.NET setter property	<code>ENEQuery.NavAggrERecsOffset</code>
Type	int
Description	Specifies a number indexing 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>+<eqlfilterinbase>+<true/false>+<rangefiltersinbase>+<true/false>+...
Description	<p>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.</p> <p>Configuration settings include:</p> <ul style="list-style-type: none"> • <basedimid> — an ID of a dimension that is to be included in the base navigation state. • <eqlfilterinbase> — a true or false value indexating whether the EQL filter is part of the base navigation state. • <textsearchesinbase> — a true or false value indexating whether text searches are part of the base navigation state. • <rangefiltersinbase> — a true or false value indexating whether range filters are part of the base navigation state. <p>When the Ndr parameter equals zero, no disabled refinement values are returned for any dimensions (which improves performance).</p>
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	<code>ENEQuery.setNavExposedRefinements()</code>
.NET setter property	<code>ENEQuery.NavExposedRefinements</code>
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	<code>ENEQuery.setNavRangeFilters()</code>
.NET setter property	<code>ENEQuery.NavRangeFilters</code>
Type	<string> [[[LT LTEQ GT GTEQ] <numeric value> BTWN <numeric value> <numeric value>]

	<key>[[GCLT GCGT GCBTWN][+<geocode reference point>]+<value>[+<value>]
Description	<p>Sets the range filters for the navigation query on properties, or for the navigation query on dimensions. If your application is built on the Assembler, you must separate multiple range filters with a double vertical pipe () delimiter. If you are using the Presentation API, use a single pipe () delimiter, instead.</p> <p>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).</p>
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	<p><string> value of the form:</p> <p>YYYY-MM-DDTHH:MM</p> <p>The letter T is a separator between the day value and the hour value. Time zone information is omitted.</p>
Description	Sets a preview time that overrides the clock of the MDEX Engine. Enables 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	<code>ENEQuery.setNavMerchRuleFilter()</code>
.NET setter property	<code>ENEQuery.NavMerchRuleFilter</code>
Type	This filter can include strings, integers, separator characters, Boolean operators, wildcard operators, and Oracle Commerce 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)
```

When `Nmrf` is present in the query, all rules that successfully triggered for that nav state, even if `INACTIVE`, are returned. If you do not use an `Nmrf` filter, the `ACTIVE/INACTIVE` property on the rules is honored, and `INACTIVE` rules do not get returned.

The workaround is to append a filter for the `ACTIVE` state in the `Nmrf` filter to prevent inactive rules from being applied; for example:

```
Nmrf=AND(endeca.internal.workflow.state:ACTIVE,16)
```

The filter above returns only the rule with a state of `ACTIVE` and a rule ID of 16.

No (Record Offset)

The `No` parameter sets the navigation record list offset.

Parameter	No
Name	Record Offset
Java setter method	<code>ENEQuery.setNavERecsOffset()</code>
.NET setter property	<code>ENEQuery.NavERecsOffset</code>
Type	int
Description	The offset defines the starting index for a navigation object's record list. If the <code>No</code> 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 enables 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	<code>ENEQuery.setNavERecsPerAggrERec()</code>
.NET setter property	<code>ENEQuery.NavERecsPerAggrERec</code>
Type	0, 1, or 2
Description	<p>Specifies the number of records to be returned with an aggregated record:</p> <ul style="list-style-type: none"> • 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. <p>To improve performance, use 0 or 1.</p>
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	<code>ENEQuery.setNavRecordFilter()</code>
.NET setter property	<code>ENEQuery.NavRecordFilter</code>

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	<code>ENEQuery.setNavRefinementConfigs()</code>
.NET setter property	<code>ENEQuery.NavRefinementConfigs</code>
Type	<string>+<string>+<string>...
Description	<p>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 <code>id</code> setting.</p> <p>The configuration settings are:</p> <ul style="list-style-type: none"> • <code>id</code> indicates the dimension value ID • <code>exposed</code> either true if the dimension value's refinements are exposed or false if not • <code>showcounts</code> indicates whether to show counts for a dimension value's refinements. Valid values are <code>true</code> to indicate counts are shown and <code>false</code> to indicate counts are not shown. • <code>synonyms</code> indicates whether to show synonyms for a navigation query. Valid values are <code>true</code> to show synonyms and <code>false</code> to not show synonyms. • <code>dynrank</code> whether the dimension value has Dynamic Ranking enabled: enabled, disabled, or default • <code>dyncount</code> maximum number of dimension values to return: either default or an integer <code>>= 0</code> • <code>dynorder</code> sort order: static, dynamic, or default <p>Omitting a setting or specifying <code>default</code> results in using the setting in Developer Studio.</p>
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
```

This example returns synonyms for the dimension value with an id of 700000 and does not return synonyms for the dimension value with an id 800000.

```
/controller.php?N=0&NrC=id+700000+synonyms+true|id+800000+synonyms+false
```

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	<code>ENEQuery.setNavStratifiedDimVals()</code>
.NET setter property	<code>ENEQuery.NavStratifiedDimVals</code>
Type	<code>int,int;int,int;...</code>
Description	<p>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:</p> <pre>stratumInt,dimvalID</pre> <p>where <i>dimvalID</i> is the ID of the dimension value and <i>stratumInt</i> is a signed integer that signifies that stratum into which the dimension value will be placed. For <i>stratumInt</i>, 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.</p>
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	<code>ENEQuery.setNavRelRankERecRank()</code>
.NET setter property	<code>ENEQuery.NavRelRankERecRank</code>
Type	<search interface>

Description	<p>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.</p> <p>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 enabled.</p> <p>Note that the <code>Nrk</code>, <code>Nrt</code>, <code>Nrr</code>, and <code>Nrm</code> parameters take precedence over <code>Ntk</code>, <code>Ntt</code>, and <code>Ntx</code>.</p>
Object	Navigation
Dependency	N, Nrt, Nrr

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sonoma&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	<code>ENEQuery.setNavRelRankERecRank()</code>
.NET setter property	<code>ENEQuery.NavRelRankERecRank</code>
Type	<string>
Description	<p>With the exception of <code>MatchBoolean</code>, all of the search modes are valid for use: <code>MatchAll</code>, <code>MatchPartial</code>, <code>MatchAny</code>, <code>MatchAllAny</code>, <code>MatchAllPartial</code>, and <code>MatchPartialMax</code>. Attempting to use <code>MatchBoolean</code> with this parameter causes the record search results to be returned without relevance ranking.</p> <p>This parameter does not support multiple match modes; therefore, the use of a pipe () is not enabled.</p> <p>Note that the <code>Nrk</code>, <code>Nrt</code>, <code>Nrr</code>, and <code>Nrm</code> parameters take precedence over <code>Ntk</code>, <code>Ntt</code>, and <code>Ntx</code>.</p>
Object	Navigation
Dependency	N, Nrk, Nrt, Nrr

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sonoma&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	<code>ENEQuery.setNavRelRankERecRank()</code>
.NET setter property	<code>ENEQuery.NavRelRankERecRank</code>
Type	<string>
Description	<p>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.</p> <p>This parameter does not support multiple relevance ranking strategies; therefore, the use of a pipe () is not enabled.</p> <p>Note that the <code>Nrk</code>, <code>Nrt</code>, <code>Nrr</code>, and <code>Nrm</code> parameters take precedence over <code>Ntk</code>, <code>Ntt</code>, and <code>Ntx</code>.</p>
Object	Navigation
Dependency	N, Nrk, Nrt

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sonoma&Nrk=All&Nrt=pear&Nrr=field&Nrm=matchall
```

Nrs (Oracle Commerce Query Language Filter)

The `Nrs` parameter sets an EQL record filter on a navigation query.

Parameter	Nrs
Name	Oracle Commerce Query Language Filter
Java setter method	<code>ENEQuery.setNavRecordStructureExpr()</code>
.NET setter property	<code>ENEQuery.NavRecordStructureExpr</code>
Type	<string>
Description	<p>Sets the Oracle Commerce Query Language expression for the navigation query. The expression will act as a filter to restrict the results of the query.</p> <p>The <code>Nrs</code> parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.</p>
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	<code>ENEQuery.setNavRelRankERecRank()</code>
.NET setter property	<code>ENEQuery.NavRelRankERecRank</code>
Type	<string>+<string>+<string>...
Description	<p>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.</p> <p>This parameter does not support multiple sets of terms; therefore, the use of a pipe () is not enabled.</p> <p>The <code>Nrt</code> parameter must be used with the <code>Nrk</code> parameter (which sets the search interface) and the <code>Nrr</code> parameter (which indicates the relevance ranking strategy to use for ordering the record set).</p> <p>Note that the <code>Nrk</code>, <code>Nrt</code>, <code>Nrr</code>, and <code>Nrm</code> parameters take precedence over <code>Ntk</code>, <code>Ntt</code>, and <code>Ntx</code>.</p>
Object	Navigation
Dependency	N, Nrk, Nrr

Example

```
/controller.php?N=0&Ntk=P_Desc&Ntt=sonoma&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	<code>ENEQuery.setNavActiveSortKeys()</code>
.NET setter property	<code>ENEQuery.NavActiveSortKeys</code>

Type	<code>Ns=sort-key-names[(geocode)][[sort order]][[...]]</code>
Description	<p>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.</p> <p>In other words, in order to sort records returned for a navigation query, you must append a sort key parameter (<code>Ns</code>) to the query, using the following syntax:</p> <pre>Ns=sort-key-names[(geocode)][[sort order]][[...]]</pre> <p>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.</p> <p>Whether the values for the sort key are sorted alphabetically, numerically, or geospatially is specified in Developer Studio.</p> <p>To sort records by their geocode property, add the optional <code>geocode</code> 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.</p> <p>Sorting can only be performed when accompanying a navigation query. Therefore, the sort key (<code>Ns</code>) parameter must accompany a basic navigation value parameter (<code>N</code>).</p>
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	<code>ENEQuery.setNavSortOrder()</code>
.NET setter property	<code>ENEQuery.NavSortOrder</code>
Type	0 or 1
Description	<p>Specifies the sort order for a navigation object's record list:</p> <ul style="list-style-type: none"> A value of 0 indicates an ascending sort, which is the default if the <code>Nso</code> parameter is not present.

	<ul style="list-style-type: none"> A value of 1 indicates a descending sort. <p>Note that previously, a sort key was specified with the <code>Ns=key</code> parameter and a sort order was specified with <code>Nso=1</code>. The <code>Nso</code> parameter has been deprecated. Now, the preferred way of specifying the sort order is also through the <code>Ns</code> parameter, using <code>Ns=key 1</code>.</p>
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	<code>ENEQuery.setNavERecSearches()</code>
.NET setter property	<code>ENEQuery.NavERecSearches</code>
Type	<search key>
Description	<p>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.</p> <p>The <code>Ntk</code> parameter must be used with the <code>Ntt</code> parameter, which indicates the search terms for each key. In addition, <code>Ntt</code> should have the same number of term sets as <code>Ntk</code> has keys.</p> <p>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.</p>
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	<code>ENEQuery.setNavERecSearchComputeAlternativePhrasings()</code>
.NET setter property	<code>ENEQuery.NavERecSearchComputeAlternativePhrasings</code>
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 `Ntpr` 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	<code>ENEQuery.setNavERecSearchRewriteQueryToAnAlternativePhrasing()</code>
.NET setter property	<code>ENEQuery.NavERecSearchRewriteQueryToAnAlternativePhrasing</code>
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	<code>Ntt</code>
Name	Record Search Terms
Java setter method	<code>ENEQuery.setNavERecSearches()</code>
.NET setter property	<code>ENEQuery.NavERecSearches</code>
Type	<code><string>+<string>+<string>...</code>
Description	<p>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 ().</p> <p>The <code>Ntt</code> parameter must be used with the <code>Ntk</code> parameter, which indicates which keys of the records to search. In addition, <code>Ntt</code> should have the same number of term sets as <code>Ntk</code> has keys.</p> <p>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.</p>
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	<code>Ntx</code>
Name	Record Search Mode
Java setter method	<code>ENEQuery.setNavERecSearches()</code>
.NET setter property	<code>ENEQuery.NavERecSearches</code>
Type	<code><string>+<string>+<string>...</code>
Description	<p>Sets the options for record search in the navigation query. The options include:</p> <ul style="list-style-type: none"> • <code>mode</code> for specifying a search mode. • <code>rel</code> for specifying a relevance ranking module. • <code>spell+nospell</code> for disabling spelling correction and DYM suggestions on individual queries. • <code>snip</code> and <code>nosnip</code> 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+MyStrategy
```

```
/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	<code>ENEQuery.setNavERecSearchDidYouMean()</code>
.NET setter property	<code>ENEQuery.NavERecSearchDidYouMean</code>
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	<code>ENEQuery.setNavRollupKey()</code>
.NET setter property	<code>ENEQuery.NavRollupKey</code>
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 Oracle Commerce records (<code>AggERec</code> objects) will be returned by the navigation query instead of Oracle Commerce records (<code>ERec</code> 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	<p>Sets the navigation search options used to enable Why Match, Why Rank, and Why Precedence Rule Fired.</p> <p>Valid string values include:</p> <ul style="list-style-type: none"> • <code>whymatch</code> — a string indexing that Why Match is enabled for the query. • <code>whyrank</code> — a string indexing that Why Rank is enabled for the query. • <code>whyprecedencerulefired</code> — a string indexing 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	<code>ENEQuery.setERecs()</code>
.NET setter property	<code>ENEQuery.ERecs</code>
Type	<record ID>
Description	Query to obtain a single specific Oracle Commerce record.
Object	Record (<code>ERec</code>)
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	<code>ENEQuery.setAggrERecSpec()</code>
.NET setter property	<code>ENEQuery.AggrERecSpec</code>
Type	<agg record ID>
Description	Query to obtain a single aggregated record from the MDEX Engine.
Object	Aggregated Record (<code>AggrERec</code>)
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	<code>ENEQuery.setAggERecNavRangeFilters()</code>
.NET setter property	<code>ENEQuery.AggrERecNavRangeFilters</code>

Type	<code><string> [[LT LTEQ GT GTEQ] <numeric value> BTWN <numeric value> <numeric value>]</code> <code><key> [[GCLT GCGT GCBTWN][+<geocode reference point>]+<value>[+<value>]</code>
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	<code>ENEQuery.setAggrERecNavDescriptors()</code>
.NET setter property	<code>ENEQuery.AggrERecNavDescriptors</code>
Type	<code><dimension value id>+<dimension value id>+<dimension value id>...</code>
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	<code>ENEQuery.setAggrERecNavRecordFilter()</code>
.NET setter property	<code>ENEQuery.AggrERecNavRecordFilter</code>

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	<code>Ars</code>
Name	Aggregated EQL Filter
Java setter method	<code>ENEQuery.setAggrERecStructureExpr()</code>
.NET setter property	<code>ENEQuery.AggrERecStructureExpr</code>
Type	<string>
Description	<p>Sets the Oracle Commerce Query Language expression for aggregated record query. The expression will act as a filter to restrict the results of the query.</p> <p>The <code>Ars</code> parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.</p>
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 von-
negut")]
```

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	<code>As</code>
Name	Aggregated Record Sort Key
Java setter method	<code>ENEQuery.setAggrERecActiveSortKeys()</code>

.NET setter property	<code>ENEQuery.AggrERecActiveSortKeys</code>
Type	<code>As=sort-key-names[(geocode)][[sort order]][...]</code>
Description	<p>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.</p> <p>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:</p> <pre>As=sort-key-names[(geocode)][[sort order]][...]</pre> <p>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.</p> <p>Whether the values for the sort key are sorted alphabetically, numerically, or geospatially is specified in Developer Studio.</p> <p>To sort records by their geocode property, add the optional <code>geocode</code> 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.</p>
Object	Aggregated Record (<code>AggrERec</code>)
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	<code>Au</code>
Name	Aggregated Record Rollup Key
Java setter method	<code>ENEQuery.setAggrERecRollupKey()</code>
.NET setter property	<code>ENEQuery.AggrERecRollupKey</code>
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 (<code>AggrERec</code>)
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	<code>ENEQuery.setDimSearchDimensions()</code>
.NET setter property	<code>ENEQuery.DimSearchDimensions</code>
Type	<dimension id> or <dimension id>+<dimension id>...
Description	<p>The Di parameter can be used with two types of dimension search:</p> <ul style="list-style-type: none"> • Default dimension search • Compound dimension search <p>Note that by default, all dimensions are enabled for default dimension search. If you use <code>Dgidx --compoundDimSearch</code> flag, all dimensions are enabled for compound dimension search.</p> <p>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).</p> <p>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).</p>
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	<code>ENEQuery.setDimSearchRankResults()</code>
.NET setter property	<code>ENEQuery.DimSearchRankResults</code>
Type	0 or 1
Description	<p>Sets the dimension search behavior used to rank results:</p> <ul style="list-style-type: none"> • 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	<code>Dn</code>
Name	Dimension Search Scope
Java setter method	<code>ENEQuery.setDimSearchNavDescriptors()</code>
.NET setter property	<code>ENEQuery.DimSearchNavDescriptors</code>
Type	<dimension value id>+<dimension value id>+<dimension value id>...
Description	<p>Specifies the navigation values that describe a navigation state that restrict the number of values that can be searched from.</p> <p>The <code>Dn</code> parameter takes a single dimension value for a given single-select dimension, and multiple dimension values for a given multiselect dimension.</p> <p>When the search query is combined with this parameter, the MDEX Engine returns dimension values that create valid navigation objects.</p>
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	<code>ENEQuery.setDimSearchResultsOffset()</code>
.NET setter property	<code>ENEQuery.DimSearchResultsOffset</code>
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 has been deprecated in MDEX Engine 6.3.0. Use the `numresults` configuration setting of the *Drc* parameter instead.

The *Dp* parameter sets the number of dimension value matches to return per dimension.

Parameter	Dp
Name	Dimension Value Count
Java setter method	<code>ENEQuery.setDimSearchNumDimValues()</code>
.NET setter property	<code>ENEQuery.DimSearchNumDimValues</code>
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 <i>Dp</i> 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	<code>ENEQuery.setDimSearchNavRecordFilter()</code>
.NET setter property	<code>ENEQuery.DimSearchNavRecordFilter</code>
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.



Note: The `Drc` parameter is not supported with compound dimension search.

Parameter	Drc
Name	Refinement Configuration for Dimension Search
Java setter method	<code>ENEQuery.setDimSearchRefinementConfigs()</code>
.NET setter property	<code>ENEQuery.DimSearchRefinementConfigs</code>
Type	<string>+<string>+<string>...
Description	<p>Sets one or more dynamic refinement configurations for a dimension search query. Each refinement configuration option is delimited by the pipe character.</p> <p>The configuration settings are:</p> <ul style="list-style-type: none"> <code>id</code> specifies a dimension value ID for the <code>numresults</code>, <code>showcounts</code>, and <code>synonyms</code> options. The <code>id</code> setting is not valid with <code>maxdepth</code> or <code>includeinert</code>. <code>numresults</code> specifies the maximum number of dimension values to return for a specified <code>id</code>. Valid values are integers greater than or equal to zero. If an <code>id</code> setting is omitted, then the <code>numresults</code> setting applies to all dimensions as a global setting. <code>showcounts</code> indicates whether to show the number of refinement counts for a specified <code>id</code>. Valid values are <code>true</code> to return counts and <code>false</code> to not return counts. If you omit an <code>id</code> setting, then the <code>showcounts</code> setting applies to all dimension values (a global setting).

	<ul style="list-style-type: none"> • <code>synonyms</code> indicates whether to show synonyms for a dimension value's refinements. Valid values are <code>true</code> to show synonyms and <code>false</code> to not show synonyms. If you omit <code>id</code> setting, then the <code>synonyms</code> setting applies to all dimension values (a global setting). • <code>maxdepth</code> indicates the maximum depth of dimension values to return. Valid values are 0 and 1. Specifying 0 returns either the root dimension value or values specified in with the <code>Di</code> parameter. Specifying 1 returns the root dimension value and the next level of the dimension hierarchy. If you omit <code>maxdepth</code>, all dimension values are returned by default (This is same as unlimited depth.) The <code>maxdepth</code> setting is a global setting for <code>Drc</code>; it cannot be restricted with the <code>id</code> setting. • <code>includeinert</code> indicates whether to return dimension values that are inert. Valid values are <code>true</code> which returns inert dimension values and <code>false</code> which does not. The default is <code>false</code>. The <code>includeinert</code> setting is a global setting for <code>Drc</code>; it cannot be restricted with the <code>id</code> setting.
Object	<code>DimensionSearchResult</code>
Dependency	<code>D</code>

Examples

This example shows refinement counts for dimension values 134711 and 132830.

```
/controller.php?D=1*&Drc=id+134711+showcounts+true|id+132830+showcounts+false
```

This example shows refinement counts for all dimension values except dimension value 600000.

```
/controller.php?D=1*&Drc=showcounts+true|id+600000+showcounts+false
```

This example returns up to 10 dimension values per dimension for all dimension values except 600000 which returns up to 15 dimension values.

```
/controller.php?D=1*&Drc=numresults+10|id+600000+numresults+15
```

This example returns the synonyms for a dimension value with an id of 700000.

```
/controller.php?D=*&Di=700000&Drc=id+700000+synonyms+true
```

This example returns dimension values to a maximum depth of 1 for three dimensions indicated with the `Di` parameter.

```
/controller.php?D=*&Di=500000+400000+300000&Drc=maxdepth+1
```

Drs (Dimension Search EQL Filter)

The `Drs` parameter sets the dimension search EQL filter.

Parameter	<code>Drs</code>
Name	Dimension Search EQL Filter
Java setter method	<code>ENEQuery.setDimSearchNavRecordStructureExpr()</code>
.NET setter property	<code>ENEQuery.DimSearchNavRecordStructureExpr</code>
Type	<string>

Description	<p>Sets the Oracle Commerce 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.</p> <p>Note that the <code>Drs</code> parameter must be URL-encoded. For clarity's sake, however, the example below is not URL-encoded.</p>
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	<code>ENEQuery.setDimSearchOpts()</code>
.NET setter property	<code>ENEQuery.DimSearchOpts</code>
Type	<string>+<string>+<string>...
Description	<p>Sets the dimension search options used in search mode and relevance ranking. The options include:</p> <ul style="list-style-type: none"> • <code>mode</code> for specifying a search mode. • <code>rel</code> for specifying a relevance ranking module. • <code>spell+nospell</code> for disabling spelling correction and DYM suggestions on individual queries. • <code>whyrank</code> 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	<code>ENEQuery.setDimSearchRollupKey()</code>
.NET setter property	<code>ENEQuery.DimSearchRollupKey</code>
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 <code>Drc</code> . Note that the rollup attribute of the property or dimension must be set in Developer Studio.
Object	<code>DimensionSearchResult</code>
Dependency	D and Drc

Examples

```
/controller.php?D=Merlot&Drc=id+1000+showcounts+true&Du=P_Winery
```

Appendix B

MDEX Engine Logging Variables

This section describes the MDEX Engine logging variables.

About MDEX Engine logging variables

The MDEX Engine logging variables can be used with the log-enable and log-disable URL config operations to toggle logging verbosity for specified features.

This makes it possible to get detailed information about MDEX Engine processing, to help diagnose unexpected application behavior or performance problems, without stopping and restarting the dgraph or requiring a configuration update.

Logging variable operation syntax

MDEX Engine logging variables are toggled using the `/config?op=log-enable&name=<variable-name>` and `/config?op=log-disable&name=<variable-name>` operations.

You can include multiple logging variables in a single request. Unrecognized logging variables generate warnings.

For example, this operation:

```
/config?op=log-enable&name=merchverbose
```

turns on verbose logging for the dynamic business rule feature, while this operation:

```
config?op=log-enable&name=textsearchrelrankverbose&name=textsearchspellverbose
```

turns on verbose logging for both the text search relevance ranking and spelling features.

However, this operation:

```
config?op=log-enable&name=allmylogs
```

returns an “unsupported logging setting” message.

In addition, the following operations are supported:

- `/config?op=log-status` returns a list of all logging variables with their values (true or false).
- `/config?op=log-enable` and `/config?op=log-disable` with no arguments return the same thing as `log-status`.
- The special name `all` can be used with `/config?op=log-enable` or `/config?op=log-disable` to set all logging variables.

Supported logging variables

The following table describes the supported logging variables.

Logging variable names are not case sensitive

Variable	Description
verbose	Enables verbose mode.
requestverbose	Prints information about each request to stdout.
updateverbose	Show verbose messages while processing updates.
recordfilterperfverbose	Enables verbose information about record filter performance.
merchverbose	Enables verbose debugging messages during merchandising rule processing.
textsearchreirankverbose	Enables verbose information about relevance ranking during search query processing.
textsearchspellverbose	Enables verbose output for spelling correction features.
dgraphperfverbose	Enables verbose performance debugging messages during core dgraph navigation computations.
dgraphrefinementgroupverbose	Enables refinement verbose/debugging messages.

Appendix C

Diacritical Character to ASCII Character Mapping

The `--diacritic-folding` flag on `Dgidx` maps accented characters to their simple ASCII equivalent as listed in the table below (characters not listed are not affected by the `--diacritic-folding` option).

Mapping table

Note that capital characters are mapped to lower case equivalents because Oracle Commerce Guided Search indexing is always case-folded.

ISO Latin1 decimal code	ISO Latin 1 character	ASCII map character	Description
192	À	a	Capital A, grave accent
193	Á	a	Capital A, acute accent
194	Â	a	Capital A, circumflex accent
195	Ã	a	Capital A, tilde
196	Ä	a	Capital A, dieresis or umlaut mark
197	Å	a	Capital A, ring
198	Æ	a	Capital AE diphthong
199	Ç	c	Capital C, cedilla
200	È	e	Capital E, grave accent
201	É	e	Capital E, acute accent
202	Ê	e	Capital E, circumflex accent
203	Ë	e	Capital E, dieresis or umlaut mark
204	Ì	i	Capital I, grave accent
205	Í	i	Capital I, acute accent
206	Î	i	Capital I, circumflex accent
207	Ï	i	Capital I, dieresis or umlaut mark
208	Ð	e	Capital Eth, Icelandic

ISO Latin1 decimal code	ISO Latin 1 character	ASCII map character	Description
209	Ñ	n	Capital N, tilde
210	Ò	o	Capital O, grave accent
211	Ó	o	Capital O, acute accent
212	Ô	o	Capital O, circumflex accent
213	Õ	o	Capital O, tilde
214	Ö	o	Capital O, dieresis or umlaut mark
216	Ø	o	Capital O, slash
217	Ù	u	Capital U, grave accent
218	Ú	u	Capital U, acute accent
219	Û	u	Capital U, circumflex accent
220	Ü	u	Capital U, dieresis or umlaut mark
221	Ý	y	Capital Y, acute accent
222	Þ	p	Capital thorn, Icelandic
223	ß	s	Small sharp s, German
224	à	a	Small a, grave accent
225	á	a	Small a, acute accent
226	â	a	Small a, circumflex accent
227	ã	a	Small a, tilde
228	ä	a	Small a, dieresis or umlaut mark
229	å	a	Small a, ring
230	æ	a	Small ae diphthong
231	ç	c	Small c, cedilla
232	è	e	Small e, grave accent
233	é	e	Small e, acute accent
234	ê	e	Small e, circumflex accent
235	ë	e	Small e, dieresis or umlaut mark
236	ì	i	Small i, grave accent
237	í	i	Small i, acute accent
238	î	i	Small i, circumflex accent
239	ï	i	Small i, dieresis or umlaut mark
240	ð	e	Small eth, Icelandic
241	ñ	n	Small n, tilde
242	ò	o	Small o, grave accent

ISO Latin1 decimal code	ISO Latin 1 character	ASCII map character	Description
243	ó	o	Small o, acute accent
244	ô	o	Small o, circumflex accent
245	õ	o	Small o, tilde
246	ö	o	Small o, dieresis or umlaut mark
248	ø	o	Small o, slash
249	ù	u	Small u, grave accent
250	ú	u	Small u, acute accent
251	û	u	Small u, circumflex accent
252	ü	u	Small u, dieresis or umlaut mark
253	ý	y	Small y, acute accent
254	þ	p	Small thorn, Icelandic
255	ÿ	y	Small y, dieresis or umlaut mark

ISO Latin1 Extended A decimal code	ISO Latin 1 Extended A character	ASCII map character	Description
256	Ā	a	Capital A, macron accent
257	ā	a	Small a, macron accent
258	Ă	a	Capital A, breve accent
259	ă	a	Small a, breve accent
260	Ą	a	Capital A, ogonek accent
261	ą	a	Small a, ogonek accent
262	Ć	c	Capital C, acute accent
263	ć	c	Small c, acute accent
264	Ĉ	c	Capital C, circumflex accent
265	ĉ	c	Small c, circumflex accent
266	Ċ	c	Capital C, dot accent
267	ċ	c	Small c, dot accent
268	Č	c	Capital C, caron accent
269	č	c	Small c, caron accent
270	Ď	d	Capital D, caron accent
271	ď	d	Small d, caron accent
272	Ð	d	Capital D, with stroke accent
273	ð	d	Small d, with stroke accent

ISO Latin1 Extended A decimal code	ISO Latin 1 Extended A character	ASCII map character	Description
274	Ē	e	Capital E, macron accent
275	ē	e	Small e, macron accent
276	Ě	e	Capital E, breve accent
277	ě	e	Small e, breve accent
278	Ė	e	Capital E, dot accent
279	ė	e	Small e, dot accent
280	Ę	e	Capital E, ogonek accent
281	ę	e	Small e, ogonek accent
282	Ě	e	Capital E, caron accent
283	ě	e	Small e, caron accent
284	Ĝ	g	Capital G, circumflex accent
285	ĝ	g	Small g, circumflex accent
286	Ğ	g	Capital G, breve accent
287	ğ	g	Small g, breve accent
288	Ġ	g	Capital G, dot accent
289	ġ	g	Small g, dot accent
290	Ģ	g	Capital G, cedilla accent
291	ģ	g	Small g, cedilla accent
292	Ĥ	h	Capital H, circumflex accent
293	ĥ	h	Small h, circumflex accent
294	Ħ	h	Capital H, with stroke accent
295	ħ	h	Small h, with stroke accent
296	Ĩ	i	Capital I, tilde accent
297	ĩ	i	Small i, tilde accent
298	Ī	i	Capital I, macron accent
299	ī	i	Small i, macron accent
300	Ĭ	i	Capital I, breve accent
301	ĭ	i	Small i, breve accent
302	Į	i	Capital I, ogonek accent
303	į	i	Small i, ogonek accent
304	İ	i	Capital I, dot accent
305	ı	i	Small dotless i

ISO Latin1 Extended A decimal code	ISO Latin 1 Extended A character	ASCII map character	Description
306	IJ	i	Capital ligature IJ
307	ij	i	Small ligature IJ
308	Ĵ	j	Capital J, circumflex accent
309	ĵ	j	Small j, circumflex accent
310	Ƙ	k	Capital K, cedilla accent
311	ƙ	k	Small k, cedilla accent
312	κ	k	Small Kra
313	Ĺ	l	Capital L, acute accent
314	ĺ	l	Small l, acute accent
315	Ł	l	Capital L, cedilla accent
316	ł	l	Small l, cedilla accent
317	Ľ	l	Capital L, caron accent
318	ľ	l	Small L, caron accent
319	Ľ	l	Capital L, middle dot accent
320	ļ	l	Small l, middle dot accent
321	Ł	l	Capital L, with stroke accent
322	ł	l	Small l, with stroke accent
323	Ń	n	Capital N, acute accent
324	ń	n	Small n, acute accent
325	Ɲ	n	Capital N, cedilla accent
326	ɲ	n	Small n, cedilla accent
327	Ñ	n	Capital N, caron accent
328	ň	n	Small n, caron accent
329	’n	n	Small N, preceded by apostrophe
330	Ŋ	n	Capital Eng
331	ɲ	n	Small Eng
332	Ō	o	Capital O, macron accent
333	ō	o	Small o, macron accent
334	Ö	o	Capital O, breve accent
335	ö	o	Small o, breve accent
336	Ő	o	Capital O, with double acute accent
337	ő	o	Small O, with double acute accent

ISO Latin1 Extended A decimal code	ISO Latin 1 Extended A character	ASCII map character	Description
338	Œ	o	Capital Ligature OE
339	œ	o	Small Ligature OE
340	Ŕ	r	Capital R, acute accent
341	ŕ	r	Small R, acute accent
342	Ŗ	r	Capital R, cedilla accent
343	ŗ	r	Small r, cedilla accent
344	Ř	r	Capital R, caron accent
345	ř	r	Small r, caron accent
346	Ś	s	Capital S, acute accent
347	ś	s	Small s, acute accent
348	Ŝ	s	Capital S, circumflex accent
349	ŝ	s	Small s, circumflex accent
350	Ş	s	Capital S, cedilla accent
351	ş	s	Small s, cedilla accent
352	Š	s	Capital S, caron accent
353	š	s	Small s, caron accent
354	Ţ	t	Capital T, cedilla accent
355	ţ	t	Small t, cedilla accent
356	Ť	t	Capital T, caron accent
357	ť	t	Small t, caron accent
358	Ƨ	t	Capital T, with stroke accent
359	Ƨ	t	Small t, with stroke accent
360	Ũ	u	Capital U, tilde accent
361	ũ	u	Small u, tilde accent
362	Ū	u	Capital U, macron accent
363	ū	u	Small u, macron accent
364	Ů	u	Capital U, breve accent
365	ů	u	Small u, breve accent
366	Ů	u	Capital U with ring above
367	ů	u	Small u with ring above
368	Ű	u	Capital U, double acute accent
369	ű	u	Small u, double acute accent

ISO Latin1 Extended A decimal code	ISO Latin 1 Extended A character	ASCII map character	Description
370	Ų	u	Capital U, ogonek accent
371	ų	u	Small u, ogonek accent
372	Ŵ	w	Capital W, circumflex accent
373	ŵ	w	Small w, circumflex accent
374	Ŷ	y	Capital Y, circumflex accent
375	ŷ	y	Small y, circumflex accent
376	ÿ	y	Capital Y, diaeresis accent
377	Ž	z	Capital Z, acute accent
378	ž	z	Small z, acute accent
379	Ž	z	Capital Z, dot accent
380	ž	z	Small Z, dot accent
381	Ž	z	Capital Z, caron accent
382	ž	z	Small z, caron accent
383	ſ	s	Small long s

Related Links

Index

- .NET reference implementation
 - setting up 59
 - testing with 60

A

- A (Aggregated Record) parameter 92, 399
- adding sample stop words 326
- Af (Aggregated Record Range Filter) parameter 399
- agg_rec module 45, 49
- aggregated records
 - creating record queries 92
 - getting from ENEQueryResults objects 93
 - methods for rollup keys 90
 - overview 89
 - ranking of refinements 97
 - refinement counts 95
 - retrieving attributes from AggrERec object 94
 - retrieving from Navigation object 94
 - setting maximum number 92
 - sorting 92
 - specifying rollup key for queries 91
- alphanumeric characters, indexing 278
- An (Aggregated Record Descriptors) parameter 92, 400
- ancestors, getting dimension 160
- Ar (Aggregated Record Filter) parameter 135, 400
- Ars (Aggregated EQL Filter) parameter 401
- As (Aggregated Record Sort Key) parameter 401
- As (Aggregated Record Sort) parameter 92
- Aspell dictionary
 - about 286
 - compiling with dgwordlist 300
 - compiling with EAC 301
 - modifying 289
 - updateaspell admin operation 289
- aspell_AND_espell and Did You Mean interaction 298
- Au (Aggregated Record Rollup Key) parameter 92, 402
- automatic key properties 205
- automatic phrasing 215
- Automatic Phrasing
 - about 315
 - API methods 319
 - extracting phrases from dimensions 318
 - importing phrases 317
 - troubleshooting 323
 - URL query parameters 319
 - use with Spelling Correction and DYM 316
 - using punctuation 318

B

- basic filtering capabilities of EQL 103

- basic queries
 - aggregated Oracle Commerce record 35
 - dimension search 35
 - navigation 35
 - Oracle Commerce record 35
- Boolean search
 - about 253
 - error messages 259
 - examples of using the key restrict operator (:) 255
 - interaction with other features 258
 - operator precedence 258
 - proximity search 255
 - semantics 257
 - syntax 254
 - URL query parameters 260
- Boolean syntax for record filters 131
- boost and bury, See dimension value boost
- browser requests transformed into MDEX Engine queries 24
- building an Oracle Commerce-enabled Web application 40
- bulk export of records
 - configuration 139
 - introduced 139
 - objects and method calls 140
 - performance impact 142
 - URL query parameters 139
- business rules and keyword redirects 217

C

- caching for record filters 135
- categories of characters in indexed text 277
- characters
 - indexing alphanumeric 278
 - indexing search 278
- characters, indexing non-alphanumeric 278
- collapsible dimension values 227
- complete dimensions 156
- compound dimension search
 - about 224
 - enabling 225
 - enabling and creating a query 229
 - enabling with Dgidx flag 228
 - example of ordering results 226
 - flags in Dgidx 225
 - limiting results 230
- compoundDimSearch flag 225
- configuring
 - dimension search 224
 - snippeting 269
- controller module 44, 47
- creating a query for default dimension search 228
- cross-field matching 220

D

- D (Dimension Search) parameter 403
- DateTime properties 149
- dead ends, *See* disabled refinements
- dead-end query results, avoiding 184
- default dimension search
 - about 223
 - creating a query 228
 - enabling 224
 - enabling for dimensions 228
 - example of ordering results 225
- derived properties
 - about 199
 - configuring 199
 - performance impact 199
 - Presentation API methods 200
- DERIVED_PROP element 199
- descriptor dimensions 155
- descriptors
 - creating new queries from 177
 - displaying 173
 - performance impact 174
 - removing from navigation state 176
 - retrieving dimension values 174
 - URL parameters 173
- Developer Studio
 - enabling hierarchical record search 210
- Df (Dimension Search Range Filter) parameter 403
- Dgidx
 - compoundDimSearch flag 225
 - nostrictattrs flag 66
 - sort flag 75
 - flags for search characters 279
- dgraph.Aggrbins property for aggregated record counts 95, 179, 235
- dgraph.Bins property for regular record counts 179, 235
- dgraph.Strata property 196
- DGraph.WhyPrecedenceRuleFired property 377
- DGraph.WhyRank property 365, 373
- dgwordlist utility for Aspell dictionary 300
- Di (Search Dimension) parameter 404
- dictionaries created by Dgidx 287
- did you mean 216
- Did You Mean feature, *See* Spelling Correction and DYM
- dimension groups
 - API methods 151
 - displaying 151
 - performance impact 154
 - ranking 153
 - versus dimension hierarchy 153
- dimension refinements
 - displaying 154
 - extracting 157
 - Ne parameter for 154
 - retrieving values for 156
- dimension search
 - about 223
 - compound, about 223
 - default, about 223
 - dimension search (*continued*)
 - enabling dimensions for it 224
 - enabling paging 231
 - filtering results 226
 - limiting results 230
 - limiting results of queries 230
 - ordering of results 225
 - performance impact 239
 - ranking results 232
 - reports 241
 - searching within a navigation state 232
 - troubleshooting 238
 - URL query parameters 228
 - when to use 238
 - dimension search results from spelling corrections 298
 - dimension value boost
 - API methods 195
 - dgraph.Strata property 196
 - interaction with disabled refinements 196
 - Nrcs parameter 194
 - overview 193
 - dimension value properties
 - about 190
 - accessing 190
 - configuring 190
 - performance impact 192
 - dimension values
 - boost and bury feature 193
 - collapsible 227
 - numeric sort on non-numeric values 74
 - dimensions
 - accessing hierarchy 160
 - configuring for record sort 74
 - extracting implicit refinements from 158
 - extracting standard refinements from 157
 - hidden 186
 - multiselect 182
 - performance impact when displaying 70
 - working with external 192
 - disabled refinements 162
 - .NET API 163
 - configuring with the Presentation API 163
 - identifying from query output 166
 - interaction with dimension value boost feature 196
 - interaction with navigation features 166
 - Java API 163
 - performance impact 166
 - URL parameter 165
 - disabling
 - spelling correction, per query 286
 - displayKey parameter 45
 - Dk (Dimension Search Rank) parameter 404
 - Dn (Dimension Search Scope) parameter 405
 - Do (Dimension Search Offset) parameter 406
 - Dp (Dimension Value Count) parameter 406
 - Dr (Dimension Record Filter) parameter 135
 - Dr (Dimension Search Filter) parameter 407
 - Drc (Refinement Configuration for Dimension Search) parameter 407
 - Drs (Dimension Search EQL Filter) parameter 408

- Du (Rollup Key for Dimension Search) parameter 410
- Duration properties 149
- Dx (Dimension Search Options) parameter 409
- dynamic refinement ranking
 - about 167
 - API calls 171
 - configuring in Developer Studio 168
 - displaying 172
 - Nrc parameter 170
 - query-time control 169

E

- enabling compound dimension search 228
- Endeca Application Controller
 - compiling Aspell dictionary 301
- Endeca.stratify sort module 354
- eneHost parameter 45
- enePort parameter 45
- ENEQuery class
 - building a basic query with 36
 - introduced 31
- ENEQueryResults class
 - described 34
 - introduced 31
- ERecEnumerator class 142
- ERecList object, displaying records in 64
- Espell module 286
- example
 - record search with search characters enabled 282
 - record search with wildcard and search characters 283
 - record search with wildcard but not search characters 283
 - record search without search characters enabled 281
- expression evaluation of record filters 137
- external dimensions 192

F

- filtering results from dimension searches 226

G

- geocode sorting
 - URL parameters for filters 85
 - use with Ns parameter 76
- geospatial sorting
 - API methods 79
 - dynamically-created properties 80
 - Ns parameter 79
 - overview 78
 - performance impact 81
 - Perl manipulator 78
- grayed out refinements 162
- Guided Search APIs 23

H

- hidden dimensions
 - about 186
 - configuring 186
 - example 187
 - handling in an application 187
 - performance impact 187
- hideMerch parameter 46
- hideProps parameter 45
- hideSups parameter 46
- hierarchical record search 210
- HttpENEConnection class 31

I

- implementing
 - search characters 279
 - Boolean search 260
 - phrase search 263
 - search modes 250
 - wildcard search 271, 272
 - wildcard search for a search interface 274
 - wildcard search, globally 273
- implicit dimension refinements
 - about 155
 - extracting 158
- indexing
 - search characters 278
 - non-alphanumeric characters 278
- inert dimension values
 - about 187
 - configuring 188
 - handling in an application 188
- Information Transformation Layer 18

J

- JSP reference implementation
 - setting up, on UNIX 56
 - setting up, on Windows 55

K

- key properties
 - about 203
 - API 205
 - automatic 205
 - defining 204
- key-based record sets
 - about 99
 - URL query parameters 100

L

- large OR filter performance impact 137
- logging variables
 - MDEX Engine 411

logging variables (*continued*)
 operation syntax 411
 supported variables for 412

M

mapping record properties 65
 MatchPartial mode and stop words 248
 MDEX Engine 214
 logging variables for 411
 flags for search characters 280
 package overview 17
 query result objects 26
 spelling correction flags 291
 MDEX Engine queries
 building with the ENEQuery class 36
 building with the UrlENEQuery class 32, 33, 35
 creating 32
 creating with ENEQuery from state information 33
 exceptions 40
 executing 34
 four basic queries 35
 results 34
 using the core objects 34
 working with results 37, 38
 MDEX Engine query
 aggregated Oracle Commerce record objects 27
 creating from a client browser request 24
 dimension search objects 27
 navigation objects 26
 record objects 26
 memory costs of record filters 137
 multi-select OR
 refinement counts 185
 multiselect dimensions
 avoiding dead-end query results 184
 configuring 182
 displaying 182
 handling in applications 183
 performance impact 185

N

N (Navigation) parameter 382
 N parameter interaction with EQL 122
 Nao (Aggregated Record Offset) parameter 382
 nav module 44, 47
 navigation filtering 217
 NCName format and EQL 106
 Ndr (Disabled Refinements) parameter 383
 Ne (Exposed Refinements) parameter 154, 384
 Ne exposed refinements interaction with EQL 123
 NEAR Boolean operator 256
 Nf (Range Filter) parameter 84, 85, 384
 Nf range filter interactions with EQL 122
 Nmpt (Merchandising Preview Time) parameter 385
 Nmrf (Merchandising Rule Filter) parameter 386
 No (Record Offset) parameter 71, 386
 non-alphanumeric characters, indexing 278

non-MDEX Engine parameters in UI reference implementations 45
 non-navigable dimension values, using 187
 Np (Records per Aggregated Record) parameter 92, 387
 Nr (Record Filter) parameter 135, 387
 Nr record filter interactions with EQL 122
 Nrc (Dynamic Refinement Ranking) parameter 170, 388
 Nrds (Dimension Value Stratification) parameter 194, 389
 Nrk (Relevance Ranking Key) parameter 389
 Nrk relevance ranking interaction with EQL 123
 Nrm (Relevance Ranking Match Mode) parameter 390
 Nrr (Relevance Ranking Strategy) parameter 391
 Nrs (Oracle Commerce Query Language Filter) parameter 391
 Nrt (Relevance Ranking Terms) parameter 392
 Ns (Sort Key) parameter 75, 354, 392
 Ns sorting interaction with EQL 123
 Nso (Sort Order) parameter 393
 Ntk (Record Search Key) parameter 394
 Ntk and Ntt record search interaction with EQL 122
 Ntpc (Compute Phrasings) parameter 395
 Ntpr (Rewrite Query with an Alternative Phrasing) parameter 395
 Ntt (Record Search Terms) parameter 396
 Ntx (Record Search Mode) parameter 353, 396
 Nty (Did You Mean) parameter 397
 Nu (Rollup Key) parameter 91, 397
 numeric sort and non-numeric dimension values 74
 Nx (Navigation Search Options) parameter 398

O

one-way thesaurus entries 309
 ONEAR Boolean operator 256
 operation syntax for MDEX Engine logging variables 411
 Oracle Commerce Analytics and EQL 124
 Oracle Commerce Presentation API
 ENEQuery class 32
 ENEQueryResults class 34
 HttpENEConnection class 31
 UrlENEQuery class 32
 Oracle Commerce Query Language
 about 103
 and dimension value IDs 114
 and dimension value paths 112, 113
 and range filter queries 117
 and record search queries 115
 and RRN queries 108
 basic filtering capabilities 103
 basic range filter syntax 118
 creating the pipeline 127
 dimension search queries 120
 dimension value queries 112
 geospatial range filter syntax 119
 implementing the per-query statistics log 125
 interaction with other features 120
 making requests 107
 N parameter interaction 122
 NCName format with 106

- Oracle Commerce Query Language (*continued*)
 - Ne exposed refinements interaction 123
 - Nf range filter interactions 122
 - Nr record filter interactions 122
 - Nrk relevance ranking interaction 123
 - Ns sorting interaction 123
 - Ntk and Ntt record search interaction 122
 - Oracle Commerce Analytics interaction 124
 - per-query statistics log 124
 - pipeline dimensions and properties 127
 - pipeline Switch joins 128
 - property value queries 107
 - range filter query examples 118
 - record search query examples 116
 - RRN module 104
 - running the pipeline 129
 - setting the logging threshold 126
 - supported property types for range filters 117
 - syntax 105
 - URL query parameters for 106
 - spelling correction and DYM interaction 124
- Oracle Commerce records
 - boost and bury feature 351
 - displaying dimension values for 68
 - paging through a record set 70
 - sorting 73
- ordering
 - compound dimension search results 226
 - default dimension search results 225
 - results of dimension search 225
- overview
 - MDEX Engine package 17

P

- paging
 - in dimension search results 231
 - through a record set 70
- per-query statistics log for EQL 124
- performance impact
 - derived properties 199
 - descriptors 174
 - dimension groups 154
 - dimension search 239
 - dimension value properties 192
 - disabled refinements 166
 - displaying dimensions 70
 - displaying refinements 162
 - dynamic refinement ranking 172
 - geospatial sorting 81
 - hidden dimensions 187
 - multiselect dimensions 185
 - phrase search 265
 - range filters 88
 - record search 218
 - refinement statistics 181
 - snippeting 270
 - sorting records 77
 - wildcard search 275

- phrase search
 - examples of queries 264
 - implementing 263
 - performance impact 265
 - URL query parameters 264
- pipeline for EQL, creating 127
- positional indexing, about 264
- Presentation API
 - architecture 23
 - Web application modules 24
- primitive term and phrase lookup 216
- processing order for record search queries 214
- properties
 - accessing 66
 - configuring for record sort 73
 - dimension value 190
 - displaying 65
 - indexing 66
 - mapping 65
 - returned as supplemental objects by the MDEX Engine 67
 - types supported in the MDEX Engine 147
- property value queries for EQL 107

Q

- queries
 - examples of limiting results with compound dimension search 230
 - examples with compound dimension search 229
- query expansion, configuring 332
- query matching interaction examples
- query matching semantics 277
- Query method (.NET) 31
- query method (Java) 31

R

- R (Record) parameter 399
- range filtering 216
- range filters
 - configuring properties and dimensions for 83
 - dynamically-created properties 86
 - Nf parameter examples 86
 - overview 83
 - performance impact 88
 - rendering results 87
 - troubleshooting 88
 - URL parameter 84
 - using multiple 86
- ranking results for dimension search 232
- rec module 44, 48
- record boost and bury
 - enabling properties 351
 - Ntx parameter 353
 - overview 167, 351
 - sorting 354
 - stratify relevance ranking 352
- record filtering during record searches 215

- record filters
 - about 131
 - caching in MDEX Engine 135
 - data configuration 134
 - enabling properties for use 134, 351
 - expression evaluation 137
 - large scale negation 137
 - memory cost 137
 - performance impact 136
 - query syntax 132
 - syntax 131
 - URL query parameters 135
 - Record Relationship Navigation filters 110
 - Record Relationship Navigation module 104
 - Record Relationship Navigation queries 108
 - examples 110
 - syntax for 109
 - record search
 - about 209
 - against multiple terms 213
 - auto correction 215
 - enabling for dimensions or properties 210
 - examples 210, 212
 - features for controlling it 211
 - MDEX Engine processing logic 214
 - methods for rendering results 213
 - performance impact 218
 - reports 241
 - stemming 216
 - thesaurus expansion 215
 - tokenization 215
 - troubleshooting 217
 - URL query parameters 211
 - when to use 238
 - reference application
 - verifying the installation 58
 - reference implementation
 - .NET 59
 - JSP 55
 - reference implementations
 - primary modules 44
 - UI 43
 - refinement counts
 - for multi-select OR refinements 185
 - refinement dimensions
 - creating a new query from a value 159
 - displaying counts 178
 - performance impact of 162
 - query-time control of dynamic ranking 169
 - retrieving values for 156
 - refinement ranking
 - record boost and bury 167
 - refinement statistics
 - displaying 178
 - enabling 178
 - performance impact 181
 - retrieving 179, 235
 - retrieving for records that match descriptors 180
 - refinements
 - disabled 162
 - refinements (*continued*)
 - grayed out 162
 - Relevance Ranking
 - about 327
 - Exact module 328
 - Field module 328
 - First module 329
 - Frequency module 329
 - Glom module 330
 - Interpreted module 330
 - list of modules 327
 - Maximum Field module 331
 - Number of Fields module 331
 - Number of Terms module 331
 - performance impact 349
 - Phrase module 331
 - Proximity module 335
 - recommended strategies 347
 - resolving tied scores 339
 - sample scenarios 345
 - Spell module 336
 - Static module 336
 - Stem module 337
 - Stratify module 336
 - Thesaurus module 337
 - URL query parameters 343
 - Weighted Frequency module 337
 - rendering results for record search 213
 - reports for record and dimension search 241
 - request parameters
 - extracting, Endeca-specific 24
 - methods for passing to the application modules 25
 - requests, making EQL 107
 - rollup keys, determining available 90
- ## S
- search characters
 - categories of characters 277
 - implementing 279
 - indexing specified search characters 278
 - MDEX Engine flags for 280
 - Presentation API development for 280
 - query matching semantics 277
 - Dgidx flags for 279
 - indexing alphanumeric 278
 - using 277
 - search interface
 - about 219
 - configuring wildcard search for it 274
 - search interfaces
 - cross-field matching 220
 - implementing 219
 - methods in Java 222
 - properties in .NET 222
 - troubleshooting 222
 - URL query parameters 221
 - search modes
 - about

- search modes (*continued*)
 - examples 250
 - implementing 250
 - list of, valid 247
 - MatchAll 248
 - MatchAllAny 249
 - MatchAllPartial 249
 - MatchAny 248
 - MatchBoolean mode 249
 - MatchPartial mode 248
 - MatchPartialMax mode 249
 - methods 251
 - URL query parameters 250
- search query processing 278
- search query processing order 214
- search reports
 - implementing 241
 - list of methods and properties 242
 - methods used 241
 - retrieving 241
 - troubleshooting 244
- Select feature for record sets 99
- snippeting
 - about 267
 - configuring 269
 - disabling 269
 - examples of queries 269
 - performance impact 270
 - reformatting for display 270
 - tips 270
 - URL query parameters 269
- sorting records
 - API methods 76
 - changing sort order 75
 - geospatial sort 78
 - Ns parameter for queries 75
 - numeric sort on non-numeric values 74
 - overview 73
 - performance impact 77
 - record boost and bury 354
 - troubleshooting problems 77
 - with no sort key 74
- spelling correction 215
 - disabling per query 286
- Spelling Correction and DYM
 - about 285
 - API methods 292
 - Aspell and Espell modules 286
 - compiling Aspell dictionary manually 300
 - compiling Aspell dictionary with EAC 301
 - configuring in Developer Studio 288
 - Dgidx flags 290
 - dgraph flags 291
 - modifying Aspell dictionary 289
 - performance impact 299
 - supported spelling modes 286
 - troubleshooting 298
 - URL query parameters 292
 - use with Automatic Phrasing 316
 - using word-break analysis 302

- Spelling Correction and DYM (*continued*)
 - with EQL 124
- stemming 216
- stemming and thesaurus
 - about 303
 - about the thesaurus 309
 - adding thesaurus entries 310
 - enabling stemming 304
 - interaction with other features 312
 - performance impact 313
 - sort order of stemmed results 304
 - troubleshooting the thesaurus 311
- stop words
 - about 325
 - and Did You Mean 298
- stop words and MatchPartial mode 248
- stratify relevance ranking module 352
- synonyms used for search 211
- syntax
 - for EQL 105
 - record filters 131

T

- taxonomies, external 192
- temporal properties, about 148
- testing
 - with JSP reference application 58
- thesaurus, See stemming and thesaurus
- thesaurus expansion 215
- time and date properties
 - defining 148
 - working with 149
- Time properties 148
- tokenization in record search 215
- troubleshooting record search 217
- two-way thesaurus entries 309

U

- UI reference implementation
 - intended usage 43
 - Javascript in 46
 - module descriptions 51
 - module maps (.NET) 49
 - module maps (Java) 47
 - non-MDEX Engine parameters in 45
 - tips on using 54
- URL parameters
 - A 92
 - A (Aggregated Record) 399
 - Af (Aggregated Record Range Filter) 399
 - An 92
 - An (Aggregated Record Descriptors) 400
 - Ar (Aggregated Record Filter) 400
 - Ars (Aggregated EQL Filter) 401
 - As 92
 - As (Aggregated Record Sort Key) 401
 - Au 92

URL parameters (*continued*)

- Au (Aggregated Record Rollup Key) 402
- D (Dimension Search) 403
- Df (Dimension Search Range Filter) 403
- Di (Search Dimension) 404
- Dk (Dimension Search Rank) 404
- Dn (Dimension Search Scope) 405
- Do (Dimension Search Offset) 406
- Dp (Dimension Value Count) 406
- Dr (Dimension Search Filter) 407
- Drs (Dimension Search EQL Filter) 408
- Du (Rollup Key for Dimension Search) 410
- Dx (Dimension Search Options) 409
- geocode range filters 85
- N (Navigation) 382
- Nao (Aggregated Record Offset) 382
- Ndr (Disabled Refinements) 383
- Ne (Exposed Refinements) 154, 384
- Nf (Range Filter) 384
- Nmpt (Merchandising Preview Time) 385
- Nmrf (Merchandising Rule Filter) 386
- No (Record Offset) 386
- non-MDEX Engine-specific 45
- Np 92
- Np (Records per Aggregated Record) 387
- Nr (Record Filter) 387
- Nrc (Dynamic Refinement Ranking) 170, 388
- Nrc (Refinement Configuration for Dimension Search) 407
- Nrcs (Dimension Value Stratification) 389
- Nrk (Relevance Ranking Key) 389
- Nrm (Relevance Ranking Match Mode) 390
- Nrr (Relevance Ranking Strategy) 391
- Nrs (Oracle Commerce Query Language Filter) 391
- Nrt (Relevance Ranking Terms) 392
- Ns (Sort Key) 392
- Nso (Sort Order) 393
- Ntk (Record Search Key) 394
- Ntpc (Compute Phrasings) 395
- Ntpr (Rewrite Query with an Alternative Phrasing) 395
- Ntt (Record Search Terms) 396
- Ntx (Record Search Mode) 396
- Nty (Did You Mean) 397
- Nu 91
- Nu (Rollup Key) 397
- Nx (Navigation Search Options) 398
- R (Record) 399
- range filters 84
- sorting record 75

URL query parameters

- for EQL 106
- Automatic Phrasing 319
- Boolean search 260
- bulk export of records 139
- for dimension search 228
- key-based record sets 100
- phrase search 264
- record filters 135
- record search 211

URL query parameters (*continued*)

- relevance ranking 343
- search interfaces 221
- search modes 250
- snippeting 269

UrlENEQuery class 32, 35

user profiles

- about 359
- API objects and calls 361
- Developer Studio configuration 359
- performance impact 361
- scenario 359

V

variables supported in MDEX Engine logging 412

verifying installation

- with JSP reference application 58

W

Web applications

- building for Oracle Commerce implementation 40
- modules 24
- primary functions 24

Why Match

- about 365
- URL query parameters 365

Why Precedence Rule Fired

- about 377
- format of dgraph property 377
- URL query parameters 377

Why Rank

- about 373
- format of dgraph property 365, 373
- URL query parameters 373

wildcard search

- about 271
- configuring 272
- configuring for a search interface 274
- configuring globally 273
- false positive matches and performance 274
- front-end application tips 275
- implementing 271
- interaction with other features 272
- performance impact 275
- retrieving error messages 274

Word Interpretation

- about 369
- API methods 369
- implementing 369
- troubleshooting 371

word-break analysis

- about 302
- configuration flags 302
- disabling 302
- performance impact 302