Oracle Commerce Guided Search

Internationalization Guide

Version 11.1 • July 2014

ORACLE®

COMMERCE

Contents

Copyright and disclaimer

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, configure, and deploy a Guided Search application that deals with multiple languages.

Who should use this guide?

This guide is intended for the use of anyone who has responsibility for creating, configuring, or deploying a Guided Search implementation that deals with multiple languages.

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 Internationalization

Chapter 1

Goals of Oracle Commerce Guided Search Internationalization

This guide explains how to use Oracle Commerce Guided Search to present your web pages in the languages spoken where you are concentrating your sales efforts.

What's In This Guide

Internationalizing a web page requires more than providing text in the language spoken by your target audience.

It also requires that your application be configured to manage the text properly -- for example, by recognizing the inflected variants of a word as equivalent to each other; by breaking up compound words so that searches may be done on their individual parts; by ignoring words (such as "the", "and", "or") that have no meaning for searches; by providing a thesaurus of words that will be treated as synonyms by searches, and so on. This guide describes how to modify your application in ways such as these to manage the language or languages spoken by your targeted customers.

This guide also provides information that will help you decide whether to use a single MDEX Engine and Oracle Commerce Guided Search application for all the languages that you need to support, or to use a separate MDEX Engine and application for each language; see *How Many MDEX Engines Do I Need?* on page 13.

For information about the parts of your Oracle Commerce Guide Search application that can or must be modified for internationalization, see *Table of Components that Require Internationalization* on page 17.

Some Assumptions Made by this Guide

The guide makes the following assumptions:

- If working with Chinese, you are familiar with the encoding and character sets (Traditional versus Simplified, Big5, GBK, and so on).
- If working with Chinese, Japanese, or Thai, you know that these languages do not use white space to delimit words.
- If working with Japanese, you are familiar with the shift_jis variants and how the same character can represent either the Yen symbol or the backslash character.

Chapter 2

How Many MDEX Engines Do I Need?

This chapter describes the most important factors to consider when you decide whether to use a single MDEX Engine to handle all of your target languages, or a separate MDEX Engine for each language.

Determining the number of MDEX Engines Needed by Your Application

No simple rule can tell you whether you need a single MDEX Engine for all the languages that your Oracle Commerce Guided Search implementation must deal with, or a separate MDEX Engine for each language. Instead, you must base your decision on the combined advantages and disadvantages of each approach, as they apply to your implementation.

Note: This section does not discuss the number of servers that Guided Search implementations may require. A single server can host multiple MDEX Engines, the number being limited largely by the performance desired.

An MDEX Engine is a Dgraph process that uses as input a single index – that is, a set of files produced by a Dgidx process. This set of files is the customer's source data, indexed by the Dgidx process for use by Guided Search operations.

An index can contain customer information in a single language or in more than one language.

The following table lists some of the factors that make a single index (and MDEX Engine) for all languages, or a separate index (and MDEX Engine) for each language, the better choice for processing your data.

	Reasons to use a single index	Reasons to use a separate index for each language
Data Ingest	Data for multiple languages comes from the same data source and has the same structure. Consistent pipeline logic and data manipulation can be applied to all your languages.	Data for multiple languages comes from separate data sources with different data structures. Each of your languages requires its own pipeline logic and data manipulation.
Baseline and Partial Updates	Data for all languages can be updated at the same time, making the development of control scripts simpler and easier.	

	Reasons to use a single index	Reasons to use a separate index for each language
Deployment	Deployment is simpler and less likely to require co-ordination of updates.	
Dimensions and Languages	Your customer data includes a single or small number of languages, and/or, your application defines a small number of dimensions. In these cases, providing a version of each dimension in each language within a single index is unlikely to slow a customer's access to the data.	Your customer data includes several languages and your application defines a large number of dimensions. The large number of dimension versions required (one for each dimension in each language) may slow the customer's access to the data if provided within a single index. Access time is less likely to be affected if a separate index is used for each language.
Query Results	Hit rates on queries may be higher when there is only one cache on the server where the MDEX Engine is installed. **Note**: Each index requires its own cache.	
Records		For most purposes, multiple languages require separate indexes only when the amount of indexed data in each language is large – for example, 100 GB.
Keyword Redirects		An index can support only a single set of keyword redirects. When an index contains keywords in different languages, keyword redirects may be executed inappropriately. This can happen when a keyword in one language is mistaken for a keyword in another language, and the keywords have different redirect behaviors. When each language is handled by a different index, there is no possibility of confusion among keywords written in different languages.
Merchandising Triggers		An index can support only a single set of keywords. If an index contains keywords in different languages, business rules may be executed inappropriately, because keywords in different languages can be mistaken for each other. When each language is handled by a different index, there is no possibility of confusion among keywords written in different languages. **Note**: Keywords that trigger merchandising rules must be specified in each supported language. For example, if English, French, and Spanish are supported by a single index, the English keyword "pants" must also be specified in French ("pantalon") and in Spanish ("pantalones").
OLT Languages	Your source data includes text in several languages that do not require or benefit from OLT analysis, such	Your source data includes more than one language that is better processed using OLT language analysis,

	Reasons to use a single index	Reasons to use a separate index for each language
	as English, French, Spanish, and Italian.	such as Chinese, Japanese, Korean, or German. In this case, use a different index for each OLT language.
Stop Words		An index can use only one set of stop words. When an application uses a single index for more than one language, it is possible for a word in one of these languages to be mistaken for a stop word in one of the other languages. For example, the French word "thé" (tea) might be mistaken for the English stop word "the" (the definite article). This type of confusion is avoided when a separate index is used for each language.
Thesaurus		An index can use only one thesaurus. When an application uses a single index for more than one language, it is possible for a word in one language to be mistake for a thesaurus entry in another. For example, the German word "Gift" (poison) might be mistaken for the English word "gift" (a present). This type of confusion is avoided when a separate index is used for each language.

Note: The number of MDEX Engines that you need is also influenced by factors such as the number of Oracle Commerce Guided Search features that your implementation is using, the number of records in the index, and the throughput (ops/sec) that you want to achieve.

Chapter 3

Oracle Commerce Guided Search Components that Require Internationalization

To create one or more versions of your Guided Search implementation to support the different languages targeted by your sales efforts, you must modify several components of that implementation.

Table of Components that Require Internationalization

The following table lists the components of an Oracle Commerce Guided Search implementation that you must modify or create in order to support the language or languages that your application is targeting.

Component	Changes required for internationalization	For more information, see:
Pre-processing scripts	Pre-processing scripts must specify the character sets to be used by your Oracle Commerce Guided Search implementations.	*Character Encoding* on page 21
Source data	Source data must be tagged with ISO-639 codes that identify the languages in which the data is to be presented to the users of your Oracle Commerce Guided Search implementation.	*Assigning Language IDs globally, per record, and per property* on page 31
Language analysis	Either Latin-1 language analysis or Oracle Language Technology (OLT) Analysis must be selected to determine how dgidx and dgraph parse and process text.	*How the MDEX Engine Chooses a Language Analysis* on page 29
Language collations	A suitable collation for controlling how text is sorted during indexing and query processing must be selected.	*Configuring Sorting through Collations* on page 39
Endeca dimensions	A separate version of each Oracle Commerce dimension must be created for each language.	*MDEX Engine Developer's Guide*
Oracle Commerce Guided Search indexed records	Properties of source data records must be mapped either to dimensions or to properties of the corresponding Oracle Commerce records.	*MDEX Engine Developer's Guide*

Component	Changes required for internationalization	For more information, see:
Search Terms	Search terms must be properly encoded before users of your Oracle Commerce Guided Search application submit them to a form.	*Encoding Search Terms* on page 23
Cartridge templates	If your implementation supports multiple locales, you can localize your custom templates	*Creating Cartridge Templates for Specific Languages* on page 43
Custom editors	Create language specific custom editors for Experience Manager.	*Creating Language-Specific Custom Editors* on page 55
Menus	Create language-specific menus and other aspects of the user interface that contain static text.	*Customizing Menus with Workbench* on page 55
HTML	Ensure that HTML pages take into account the requirements of the languages that they will display.	*Internationalizing HTML Pages* on page 43
Text manipulation	The following text manipulation features must be configured for use in particular languages: sorting, stemming, decompounding, thesaurus, wildcarding, spelling and did you mean, stop words, and keyword triggers.	*Managing Text in Different Languages* on page 47
Logging and reporting	Consider running a separate Logserver for each language, to make it easier to run language-specific logs and reports.	*MDEX Engine Developer's Guide*.

Part 2

Internationalizing Oracle Commerce Guided Search Data Records

Chapter 4

Character Encoding

This chapter explains how to configure your Oracle Commerce Guided Search implementation to use the character encodings that are best suited to the languages that your implementation supports.

Introduction

The text that an Oracle Commerce Guided Search application displays to its users is stored in memory as char arrays, c-strings, pascal strings, or other data structures. Before the application can display the text, it must convert the text into a format that it can display correctly and legibly. The process of converting the text is known as *encoding*.

Any process that reads and writes data must both encode and decode it. In particular, data must be encoded or decoded during I/O operations such as the following:

- Reading from disk
- Saving to disk
- Sending across a network
- Rendering a web page

Related Links

Character Encoding on page 21
This chapter explains how to configure your Oracle Commerce Guided Search implementation to use the character encodings that are best suited to the languages that your implementation supports.

Choosing the Right Encoding for a Language on page 22
Choosing the right encoding for text can minimize loss of information and ensure that your application renders text correctly and legibly.

Specifying Character Sets Through Java Manipulators (Forge) on page 22
You must specify the encoding for characters displayed in your application's user interface through a Java Manipulator component of the Forge pipeline.

Encoding Search Terms on page 23
You must ensure that search terms are properly encoded before users of your Guided Search application submit them to a form.

Specifying Character Encodings for HTML Pages on page 23
In each HTML page that your application displays, you must specify the correct character encoding using a Content-Type META tag. In addition, any links in the page must also encode these strings properly.

Choosing the Right Encoding for a Language

Choosing the right encoding for text can minimize loss of information and ensure that your application renders text correctly and legibly.

Unless you have reason to use other encodings, choose UTF-8 for:

- All of your Endeca indexing processes, such as Forge and Dgidx.
- Rendering English and most European languages. UTF-8 is optimized for these languages.
- UNICODE characters, which your application may display incorrectly if they are not encoded as UTF-8.

Note: Use the same encoding across all of your Endeca data processing/indexing components.

When to Use Encodings Other Than UTF-8

Use encodings other than UTF-8 only for reasons such as the following:

- Your data is in Hindi, Arabic, Chinese, Japanese, Korean or other languages for which UTF-8 is not a suitable or even a possible encoding. Some Korean glyphs are not supported by Unicode, for example.
- Encodings such as EUC, Shift JIS, HZ, and GB2312 have lower memory and conversion costs than UTF-8 for Chinese, Japanese, and Korean, as well as for certain cell phones.
- Encodings other than UTF-8 can reduce consumption of disk space for Chinese, Japanese, and Korean languages.
- You need to debug the indexing process using editors that support only EUC or Shift JIS.

Know the Encoding of Your Source Data

Make sure you know (or can determine) the encoding of all of your source data. Note the following:

- Web pages from web crawls can be in any of a wide variety of encodings.
- Some applications encode text in CP1252 and variants of the ISO-8859 encodings.
- Some documents are stored in encodings other than the ones that they declare; for example, web pages that declare their charset to be UTF-8 may in fact have been saved in ISO-8859-1 or CP1252.

Note: Make sure that all input sources, such as CAS, encode any text that they read from external sources using the same encoding that the external sources use for the text.

Specifying Character Sets Through Java Manipulators (Forge)

You must specify the encoding for characters displayed in your application's user interface through a Java Manipulator component of the Forge pipeline.

Java Manipulators

In Java Manipulators, you can specify Java routines that set the character encoding of your source data to UTF-8 as follows:

```
File f = new File(fileName);
FileInputStream fis = new fileInputStream(f);
InputStreamReader isr = new InputStreamReader(fis, "UTF8");
Buffered reader r = new (BufferedReader(isr);
```

Guided Search saves characters as UTF-8 by default.

For detailed information about how to create and configure Java Manipulators, refer to the *Developer Studio Online Help*.

Encoding Search Terms

You must ensure that search terms are properly encoded before users of your Guided Search application submit them to a form.

Specify encoding (such as UTF-8) for search terms in the following calls to the Presentation API:

- Statements that retrieve information from the `HttpServletRequest` object:
  - `request.setCharacterEncoding("UTF-8");`

  Statements that construct URL query strings:
  - `UrlGen urlg = new UrlGen(request.getQueryString(), "UTF-8");`

  Statements that create queries to the MDEX Engine; for example:
  - Java: `ENEQuery usq = new UrlENEQuery(request.getQueryString(),"UTF-8");`
  - .NET: `ENEQuery nequery = new UrlENEQuery(Request.QueryString.ToString(), "UTF-8");`

  For information about how to invoke the Presentation API to create and manage queries, refer to the *MDEX Engine Development Guide*.

Unicode Normalization of Search Terms

During indexing, text is normalized to NFC (Normalization Form Composition); that is, equivalent sequences of characters are converted to the same sequence of code points. For best recall, be sure to normalize your search terms to NFC before they are used in queries.

To normalize text, use a Normalizer object such as the one provided with the IBM International Components for Unicode (ICU) library:

```
import com.ibm.icu.text.Normalizer;

String nfc = Normalizer.normalize(searchTerms, Normalizer.NFC);
```

Lowercase Conversion of Search Terms

Uppercase characters in search terms are automatically mapped to lowercase characters. For example, searching for *WINES* is equivalent to searching for *wines*.

In some cases, uppercase characters can be converted to lowercase characters in more than one way, given a variety of local spelling conventions. For example, the German word *FLUSS* (river) can be converted either to *fluss* or to *fluß*.

You can pre-process the search terms in application code to conform to local spelling conventions before the search term is submitted.

Specifying Character Encodings for HTML Pages

In each HTML page that your application displays, you must specify the correct character encoding using a Content-Type META tag. In addition, any links in the page must also encode these strings properly.

The following example illustrates how to specify character encoding for an HTML page using the Java `URLEn¬coder` class:

```
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<a href="search.jsp?term=<%=URLEncoder.encode(searchTerm,"UTF-8") %>">
```

Chapter 5

Mapping Source Record Properties to Endeca Records

This chapter provides guidelines for mapping multiple language properties of source database records to properties or dimensions in Endeca records. For information about how to create Endeca dimensions and properties and map them to properties of source database records, refer to the *Oracle Commerce Developer Studio Help*.

Steps for Mapping Source Record Properties to Guided Search

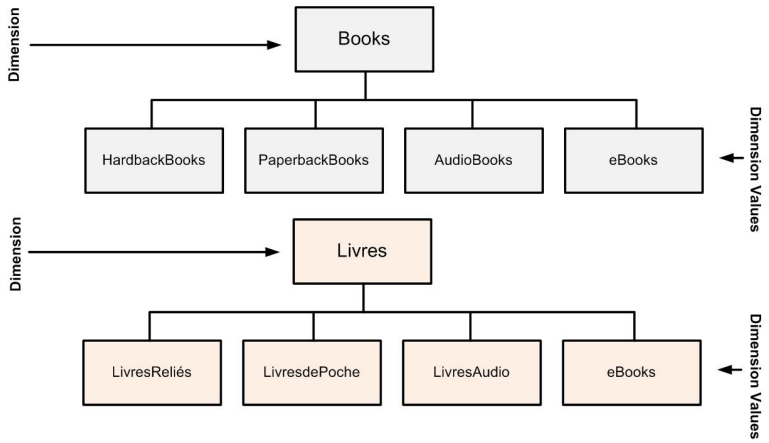
To map source record properties in different languages to Endeca dimensions or properties, follow these steps:

1. Identify the properties of your source database record that exist in more than one language. For example, the same set of logically related properties can exist in corresponding English and French versions, as follows:

Books	Livres
HardbackBooks	LivresReliés
PaperbackBooks	LivresdePoche
AudioBooks	LivresAudio
eBooks	eBooks

2. Decide whether each source database property should be mapped to an Endeca dimension or an Endeca property. Endeca dimensions refer to general categories of products and services, providing the logical structure needed for guided navigation and record searches. Endeca properties provide descriptions of products or services; your application can display these descriptions when the user accesses the records to which they apply. For more information about Endeca records, dimensions, and properties, refer to the *Oracle Commerce Guided Search Concepts Guide*.
   Note: For best results from Experience Manager and the boost/bury feature, dimension names should be NCN-compliant.
3. Use the Oracle Commerce Developer Studio to create dimensions and their dimension values, and map the dimensions and dimension values to the appropriate properties of your source database records. For example, create two dimensions, **Books** and **Livres**, and map the other, more specific, source database

record properties to values of these dimensions, as follows:



Note: Dimension names must be NCName-compliant. Dimension values can contain diacritical marks and extended characters.

4. Create Endeca properties and map them to the appropriate properties of your source database record.

Part 3

Analyzing and Sorting

Chapter 6

Language Analysis

This chapter describes how the MDEX Engine selects and applies a set of rules known as a language analysis when it indexes the text in your source records.

Indexing Languages with a Language Analysis

A language analysis is a set of rules that the MDEX Engine applies when it indexes text in any of the languages in a particular group of languages. The languages in this group have common characteristics that require special handling by an appropriate language analysis.

Every form of language analysis provides, at a minimum, tokenization: the breaking up of compound phrases into their constituent words or characters. Language analysis can optionally include stemming, which makes it possible to match inflected word forms that share a stem for example, to treat "family", "families", and "family's" as forms that match each other. Each language analysis includes a different set of other text management features, such as ignoring "stop words" (that is, common words without value for searches, such as "the"), and ignoring accents (diacritic folding).

The MDEX Engine can apply a language analysis to records, record properties, dimension tags on records, or to all record data processed by an MDEX Engine. Only one language analysis can be applied to any of these units at a time.

Oracle Commerce Guided Search supports two standard forms of language analysis, Latin-1 and OLT (Oracle Language Technologies), which are designed for use with different languages. Customers can create and use non-standard language analyses, if neither Latin-1 nor OLT meets their requirements. For detailed information about Latin-1 and OLT language analysis, see *Latin-1 and OLT Language Analysis* on page 65.

Related Links

Language Analysis on page 29
This chapter describes how the MDEX Engine selects and applies a set of rules known as a language analysis when it indexes the text in your source records.

How the MDEX Engine Chooses a Language Analysis on page 29
To choose a language analysis to use for processing a particular record, the MDEX Engine follows these steps:

How the MDEX Engine Chooses a Language Analysis

To choose a language analysis to use for processing a particular record, the MDEX Engine follows these steps:

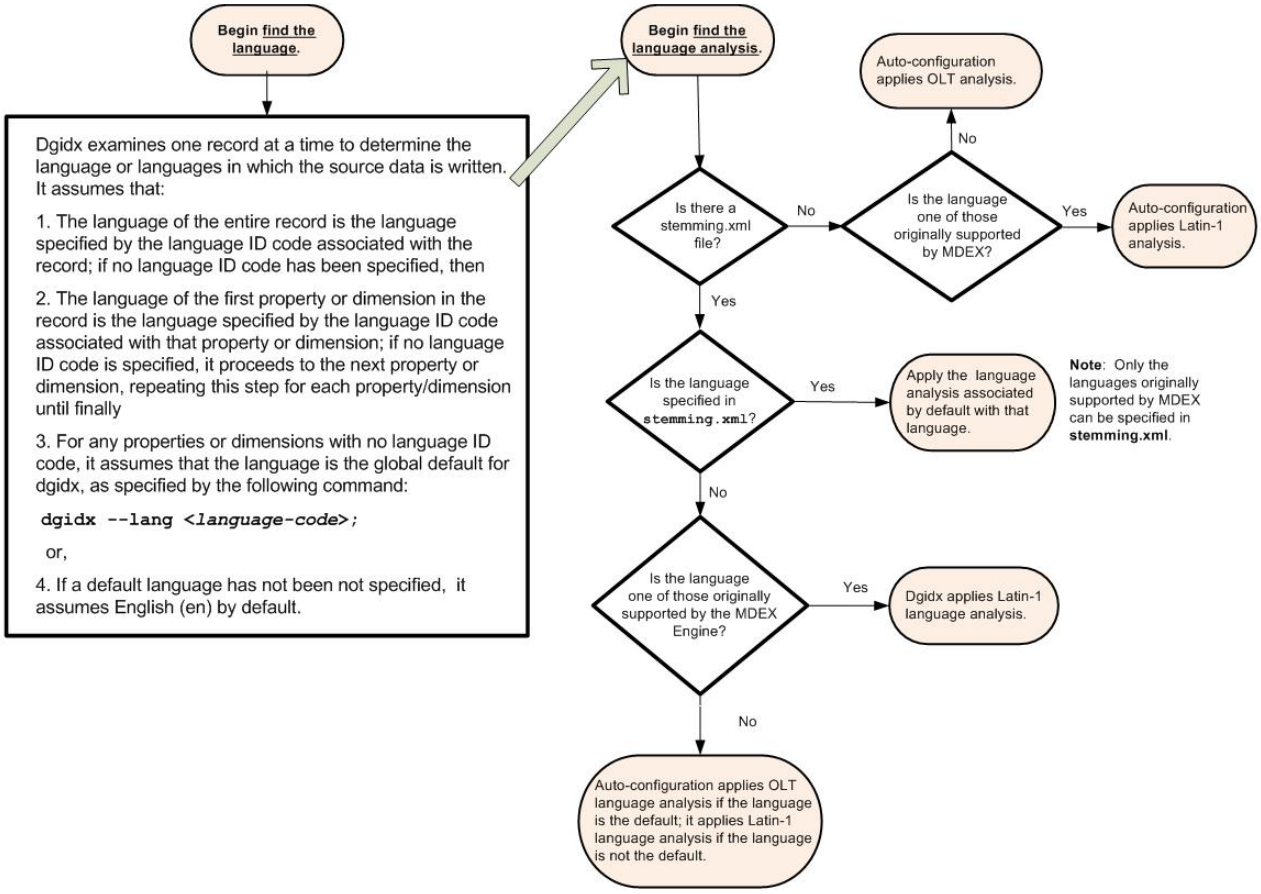
1. If a language ID code has been associated with the record, it assumes that all the record's content is in that language and proceeds to the next record.
2. If it does not find a language ID code for the record, it examines the first property in the record.
3. If it finds a language ID code for the first property, it associates that language with the property value and proceeds to the next property in the record.
4. If it does not find a language ID code for the first property, it proceeds to the next property.
5. When it has examined all properties, it applies the default language for MDEX as a whole (as specified by `dgidx --lang <`*`language code`*`>`) to any properties for which no language is specified; or, if no default language has been specified,
6. It applies en (United States English) to any properties for which a language is not specified.
7. It then proceeds to the next record, on which it repeats Steps 1 - 6.

For information about how language ID codes can be associated with records, properties, and dimensions, see *Assigning Language IDs globally, per record, and per property* on page 31. For information about dgidx, refer to the *Oracle Commerce Guided Search Administrator's Guide*.

Note: In the preceding sequence of steps, the MDEX Engine treats dimensions the same way that it treats properties.

When dgidx has determined the language of the record, property, or dimension, it consults the `stemming.xml` file (if one exists) to determine whether it contains an entry for that language. If an entry exists, dgidx uses information in the entry to decide which language analysis to apply to the language, and which features of that analysis to use. If no stemming file exists, or one exists but does not contain an entry for a particular language, an applicable default language analysis is applied. For information about `stemming.xml`, see *Specifying Language Analysis in stemming.xml* on page 35.

The following figure illustrates the process by which the MDEX Engine chooses a language analysis as described above:



Assigning Language IDs globally, per record, and per property

You assign language IDs at the following different levels:

- Per MDEX Engine, globally, to specify a default language for any records, properties, or dimension tags that have not been assigned a language. If no global default language is specified, English is assumed to be the global default.
- Per record. This is appropriate when different records contain different languages.
- Per dimension or property. This is appropriate when records contain dimensions or properties in different languages.
- Per query, which should be used in your front-end application if the language varies from query to query.

The language ID value that you assign to a record, property, or dimension must be a valid RFC-3066 or ISO-639 code, such as `en` (English), `de` (German), `ja` (Japanese), or `zh-TW` (traditional Chinese).

A full list of ISO-639 codes is available at:

`http://www.loc.gov/standards/iso639-2/php/code_list.php`

Assigning Language IDs Globally (per MDEX Engine)

Specify a global language ID using the `--lang` *<lang_code>* option on the Dgidx and Dgraph commands, where *<lang_code>* is the ISO-639 code for the language in the records; for example:

```
dgidx --lang en
dgraph --lang en
```

If you do not specify a global language ID, the MDEX Engine assumes by default that the language ID is `en` (English).

Assigning Language IDs per Record

Assign a language ID to each record if each record contains only one language, but different records contain different languages.

To specify a language ID for a record, create a Java manipulator in your Developer Studio pipeline, and configure it to add a property named `Endeca.Document.`*`lang_code`* to the record, where <*lang_code*> is the ISO-639 language ID code for the language in the records; for example: `Endeca.Document.ja`, which indicates that the language of the record is Japanese.

For more information about how to create and use Java manipulators, refer to the *Developer Studio Help*.

Assigning Language IDs per Property

Assign language IDs to individual properties in *`prefix`*`.languages.xml` files, where `prefix` is the name of your application; for example, `smithHardware.languages.xml`.

For example, the following excerpt from a languages file assigns language IDs to properties named "Property 1", "Property 2", and "Property 3":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LANGUAGE SYSTEM "languages.dtd">
<LANGUAGES>
<KEY LANGUAGE NAME="Property 1" LANGUAGE="en"/>
<KEY LANGUAGE NAME="Property 2" LANGUAGE="es"/>
<KEY LANGUAGE NAME="Property 3" LANGUAGE="ja"/>
</LANGUAGES>
```

Note: You must also create language-specific user interfaces to display information in particular languages. For information about how to do this, refer to the *Workbench User's Guide* and the *Assembler Application Developer's Guide*.

Properties that contain more than one language

Some properties are written predominantly in one language but contain words or phrases in another language or languages. You can assign to such a property the language ID of the language in which the property is written predominantly.

Any queries against the property must have the same language ID as the property. However, search terms in any of the languages that a property contains will return expected results.

Language-specific functions will be applied to search terms for the language specified in the property's language ID. Thus, if the language ID of a property is `ja`, any language-specific functions such as stemming that are applied to the property will be for Japanese; these functions will be applied to all words in the search terms, although they work well only for the language of the language ID.

Setting the Language of Queries

If your application accesses data in more than one language, you must specify the language of each particular query at query time. For example, if you set the language ID of a property to `ja`, any search run against that property must also specify the language id `ja`, or text management features such as word breaks will not function correctly.

The MDEX Engine (Dgraph) determines the language of the text of a query according to the following rule:

1. The language specified by the `setLanguageId` method (Java) or the `LanguageId` instance property (.NET) of the ENEQuery object; or, if no language has been specified, then:
2. The default language of the Dgidx.

Note: When OLT analysis is selected, the language of queries can be set only with `setLanguageId()`; setting the language on ENEQuery has no effect.

If all your data is in the same language, set the default language of the MDEX Engine to that language. For information about how to do this, see *Assigning Language IDs Globally (per MDEX Engine)* on page 31.

Properties searched by queries with different language IDs

Some properties will be searched by queries in different languages; that is, by queries with different language IDs. For example, a Partnumber property might be searched for in both Japanese and English.

In such a case, you can create a separate Partnumber property for each language. Each property name must be appended with the applicable language ID code; for example: `Partnumber_ja` (for Japanese searches) and `Partnumber_en` for English.

Chapter 7

Configuring How Text is Processed in stemming.xml

The `stemming.xml` file determines whether Dgidx performs stemming on particular languages. It also controls which language analysis (Latin-1 or OLT) applies to particular languages.

Specifying Language Analysis in stemming.xml

The `stemming.xml` file contains information that enables dgidx to select a language analysis for a particular language.

The `stemming.xml` file can contain entries only for the languages originally supported by the MDEX Engine. For a list of the originally supported languages and default language analysis applied to each, see *MDEX Engine Originally Supported Languages* on page 71.

If your source data includes one of the languages originally supported by MDEX and you need to enable stemming for that language, be sure that an entry for this language is included in the `stemming.xml` file. If `stemming.xml` does not contain an entry for a language, stemming is not enabled for that language and the default analysis is applied to that language.

In `stemming.xml`, the entry for a language is contained in a separate `<STEMMING>` element. Each subelement in the `<STEMMING>` element begins with STEM_*language-code*, where *language-code* identifies the language; for example, STEM_DE for German, STEM_EL for Greek, and STEM_HE for Hebrew. The subelements specify the following:

- Whether stemming is to be performed on that language.
- Whether a static stemming file is to be used.
- Whether compound matching is to be performed.

For example, the following entry, for American English, specifies that stemming is to be performed using a static wordforms file, and that compound matching is not to be performed:

```
<!DOCTYPE STEMMING SYSTEM "stemming.dtd"
<STEMMING>
  <STEM_EN_US ENABLE="TRUE"
              USE_COMPOUND_MATCHING="FALSE"
              USE_STATIC_WORDFORMS="TRUE" />
</STEMMING>
```

The following sections describe the subelements of the `<STEMMING>` element. For more information about `stemming.xml`, refer to the *Platform Services XML Reference*.

ENABLE

When ENABLE is set to TRUE, language analysis (in addition to tokenization) is enabled; the analysis includes not only stemming, but other functions that the analysis can perform, such as the use of a thesaurus, of stop words, and so on.

When ENABLE is set to FALSE, the language analysis performs only tokenization. A warning is displayed if ENABLE is set to FALSE and OLT language analysis is selected. The warning informs you that the setting of ENABLE in this case will be ignored, because OLT performs stemming unconditionally.

Note that the only stemming that Latin-1 analysis performs on English is to treat singular and plural wordforms as matches for each other; for example, to make "house" and "houses" match each other. This follows necessarily from the largely uninflected nature of English.

USE_STATIC_WORDFORMS

If set to TRUE, the stemming feature uses the static wordform dictionary files shipped with the MDEX Engine package. A static wordform dictionary file defines sets of inflected forms that are treated as matches for each other by the Guided Search feature; for example, the German word that means table, "Tisch", in its different grammatical cases, is specified as follows in the German wordforms file:

```
<WORD_FORMS>
  <WORD_FORM>tisch</WORD_FORM>
  <WORD_FORM>tisches</WORD_FORM>
  <WORD_FORM>tische</WORD_FORM>
  <WORD_FORM>tischen</WORD_FORM>
 </WORD_FORMS>
```

Default static wordform dictionary files are stored in `Endeca\MDEX\version\conf\stemming` (on Windows) and `usr/local/endeca/MDEX/version/conf/stemming` (on UNIX). You can update the default static wordform dictionary files, or create custom static wordform dictionary files to use in place of the default files. For information about how to do this, refer to the *MDEX Developer's Guide*.

If set to FALSE, dgidx generates the wordforms file from the source data dynamically.

Static wordform files are always used for stemming by Latin-1 analysis and are never used by OLT analysis. Thus, if USE_STATIC_WORDFORMS is set to TRUE, Latin-1 is selected; if set to FALSE, OLT is selected.

Note: Setting USE_STATIC_WORDFORMS to TRUE forces Latin-1 analysis to be selected for a language even if Latin-1 is not the better analysis for that language. For example, if you set USE_STATIC_WORDFORMS to TRUE for traditional Chinese, Latin-1 is applied, with the result that the Chinese text is not properly tokenized. Similarly, setting USE_STATIC_WORDFORMS analysis to FALSE forces OLT analysis to be used, which produces unsatisfactory results for some Western languages such as English.

USE_COMPOUND_MATCHING

If set to TRUE, the stemming feature matches compound words with any of their elements taken individually.

For example, when compound matching is enabled, the GERMAN word Bananenstecker (banana plug) can be matched either by "Banane" (banana) or by "Stecker" (plug). When compound matching is disabled, Bananenstecker is not matched by "Banane" or "Stecker", although it can be matched by inflected forms such as "Bananensteckers" (genitive singular).

Developer Studio and Custom stemming.xml Files

To change default values in stemming.xml, you can edit it manually – that is, you can edit it directly using a text editor, rather than editing it through Developer Studio. Manual edits that you make to `stemming.xml` can

be affected when you save your Developer Studio project and when you upgrade Developer Studio to the current version.

Note: The default values provided in `stemming.xml` by Developer Studio for the languages that it supports are suitable for almost all purposes.

Effect on stemming.xml of Saving Your Developer Studio Project

Developer Studio enables you to select the languages for which you want to enable stemming; the `stemming.xml` file is then written with default values for the languages that you select. You can change the default values only by editing `stemming.xml` manually.

Before you edit `stemming.xml` manually, always save and close your Developer Studio project. If you make manual edits to `stemming.xml` while your project is open in Developer Studio, those edits are overwritten when you save the project.

However, if you close your Developer Studio project before you edit `stemming.xml` manually, your edits are preserved, unless they conflict with default stemming values that are specified in the project. In this case, the edits are preserved as long as the stemming configuration in Developer Studio is not changed.

Effect on stemming.xml of Upgrading Developer Studio

When you upgrade Developer Studio to a newer version, you are prompted to save any existing Developer Studio projects to a new location. Your existing project, however, is preserved in its existing location; as a result, you have two projects, one using the older version of Developer Studio, and one using the new version.

The `stemming.xml` file for the newer, upgraded project has the same values as the original `stemming.xml` file. You can edit the upgraded version of `stemming.xml` manually or using Developer Studio. Any edits that you make to the upgraded version of `stemming.xml` do not affect the original version.

Note: Oracle recommends that you save the existing Developer Studio project as prompted when you open the existing project with a new version of Developer Studio.

Specifying non-default language analysis

With the exception of Chinese, Japanese, and Korean (the CJK languages), you can set the default language analysis for each language to either OLT or Latin-1 language analysis. CJK languages default to OLT analysis and cannot be configured to use Latin-1.

To change the default language analyzer for other languages:

1. For Dutch, English, English (UK), French, German, Italian, Portuguese, and Spanish, which default to Latin-1 analysis:
   a) Open the stemming file for your application.
      For example, `Endeca\apps\<app name>\config\pipeline\<app name>.stemming.xml`.
   b) In the entry for the language, set `USE_STATIC_WORDFORMS="FALSE"`.
   c) Save and close the file.
      This configures the language for OLT analysis.
2. For Arabic, Czech, Danish, Greek, Hungarian, Polish, and Russian, which default to OLT analysis:
   a) Navigate to the `MDEX\<version>\conf\stemming\custom` directory.
   b) Create a static stemming dictionary named `<lang id>_word_forms_collection.xml`.

c) Open the stemming file for your application.
   For example, `Endeca\apps\<app name>\config\pipeline\<app name>.stemming.xml`.
d) In the entry for the language, set `USE_STATIC_WORDFORMS="TRUE"`.
e) Save and close the file.
   This configures the language for Latin-1 analysis.

Note: The configuration for the `stemming.xml` file was designed to accept only a limited set of languages. These languages must be enabled explicitly in the file to set the language analyzer. Additional languages, including those listed below in Step 3, are automatically configured based on the presence or absence of a custom stemming dictionary.

3. For Catalan, Croatian, Finnish, Hebrew, Persian (Farsi), Portuguese (Brazil), Norwegian (Bokmal and Nynorsk), Romanian, Serbian, Serbian (Latin), Slovak, Slovenian, Swedish, Thai, and Turkish, which default to OLT analysis:
   a) Navigate to the `MDEX\<version>\conf\stemming\custom` directory.
   b) Create a static stemming dictionary named `<lang id>_word_forms_collection.xml`.
      This configures the language for Latin-1 analysis.

   The presence of the static stemming dictionary is sufficient to change the language analyzer to Latin-1.

Note: The Dgidx and Dgraph load custom dictionaries for all languages configured in the `stemming.xml` file.

Chapter 8

Configuring Sorting through Collations

Collations are used by your application to determine how it sorts the records returned by a customer's searches.

About language collations

Guided Search supports different collations for sorting in different languages. These include the Endeca collation, the Standard collation, and several language-specific collations. Guided Search uses the Endeca collation by default.

The Endeca Collation

The Endeca collation places lower case characters before the upper case versions of those same characters. For example, the Endeca collation sorts text as follows:

```
0 < 1 < ... < 9 < a < A < b < B < ... < z < Z
```

The Endeca collation is optimized for unaccented languages and ignores accents and punctuation. For this reason, in applications that use English as their global language, the Endeca collation performs better during indexing and query processing than the Standard collation. In applications that use non-Latin scripts or Latin scripts with accents, the Endeca collation may produce unexpected results for accented characters.

The Standard Collation

The Standard collation sorts data according to the International Components for Unicode (ICU) standard for the language you specify with `--lang` flag. For details about the standard collation for a particular language, see the Unicode Common Locale Data Repository at `http://cldr.unicode.org/`. In applications that include internationalized data, the Standard collation is typically the more appropriate choice because it accounts for character accents during sorting.

Language Specific Collations

In addition to the Endeca and Standard collations, dgidx and the dgraph support the following language-specific ICU collations:

- `de-u-co-phonebk`, a German collation that sorts according to phone book order rather than by dictionary order.
- `es-u-co-trad`, a Spanish collation that sorts the ch and ll characters in the traditional order rather than the standard order.
- `zh-u-co-endeca, zh-TW-u-co-endeca` For basic Latin characters, lowercase characters are placed before uppercase characters. Otherwise, characters are sorted by the numeric value of their UNICODE encodings (that is, by "code point" order).

- `zh-u-co-pinyin, zh-TW-u-co-pinyin`, an alphabetic sort of the Romanization of the readings of Chinese characters
- `zh-u-co-big5han, zh-TW-u-co-big5han`, which collates in the order of the big5han character encoding once used for Traditional Chinese. The encoding is now Unicode, but the collation order remains in use.
- `zh-u-co-gb2312han, zh-TW-u-co-gb2312han`, a collation defined by the GB2312 standard (mainland China) for Simplified Chinese. It is a mixture of pinyin for common characters and radical/stroke for less common characters.
- `zh-u-co-stroke, zh-TW-u-co-stroke`, a collation based on the total stroke count of the characters and is typically used with Traditional Chinese.
- `zh-u-co-unihan, zh-TW-u-co-unihan`, a collation defined by the Unified Han (Unihan) standard and based on (most significant first) radical/stroke, then Unicode block, and finally code point.

Note: Collations with the prefix zh-u-co- apply to Simplified Chinese, and those with the prefix zh-TW-u-co- apply to Traditional Chinese.

The following section explains how to specify the collation that you want to use for your data.

Specifying a global language ID and collation order

If most of the text in an application is in a single language, you can specify a global language ID by providing the `--lang` option and a <*`lang-id`*> argument to the Dgidx and dgraph components. The MDEX Engine treats all text as being in the language specified by <*`lang-id`*>, unless you tag text with a more specific language ID (that is, per-record, per-dimension, or per-query language IDs). The <*`lang-id`*> defaults to `en` (US English) if left unspecified.

For example, to indicate that text is English (United Kingdom), specify: `--lang en-GB`.

In addition to specifying a language identifier, you can optionally specify a collation order using an argument to the `--lang` option. A collation is specified in the form:

`--lang` <*`lang-id`*>`-u-co-`<*`collation`*>, where:

- <*`lang-id`*> is the language Id and may also include a sub-tag. If unspecified, the value of <`lang-id`> is `en` (US English).
- `-u` is a separator value between the language identifier portion of the argument and the collation identifier portion of the argument.
- `-co` is a key that indicates a collation value follows.
- `-`<*`collation`*> is the collation type of either `endeca`, `standard`, or in some cases, other language-specific ICU collations such as `phonebk` . If unspecified, the value of <*`collation`*> defaults to `en-u-co-endeca`.

For example, `--lang de-u-co-phonebk` instructs Dgidx and the dgraph to treat all the text as German and collate the text in phonebook order.

Dgidx sorts records by the value of particular properties and/or dimensions. The properties and dimensions used for sorting are specified by the `--sort` option of the Dgidx command. The `--sort` option also specifies whether ascending or descending sort order is used with each property or dimension. For information about the `--sort` option, refer to the *MDEX Engine Developer's Guide.* If you do not specify a particular sort order through the `--sort` option, Dgidx sorts records by their internal record IDs.

Part 4

Designing an Internationalized User Interface

Chapter 9

Designing an Internationalized User Interface

You must create a version of each of your cartridge templates for each of the languages that your Oracle Commerce Guided Search application supports.

Creating Cartridge Templates for Specific Languages

To create localized versions of your cartridge templates, follow these steps:

1. Create resource property files to store localized strings for each locale. Each resource property file name must follow this format:

   `Resources_<locale>.properties`

   where <*locale*> is the ISO language code. For example `Resources_fr.properties` indicates that French values are stored in it. Place these files in a locales folder for your custom template:

   `<app dir>\config\cartridge_templates\<template_identifier>\locales`

2. Specify values that do not change for locale (thumbnail URLs for example) in the single `Resources.properties` file or directly in the `template.xml` file.
3. In the template itself, use `${property.name}` notation in element content and attributes to reference a localized string in the `Resources_<locale>.properties`. Only content in the Description, ThumbnailURL, and EditorPanel sections can reference localized strings in the resources properties files. For more information about creating language-specific cartridge templates, refer to the *Assembler Application Developer's Guide.*

Internationalizing HTML Pages

When you design HTML pages for use by multi-lingual web applications, keep the following guidance in mind:

- Remove all text from images so that you do not have to provide a separate version of images for each language.
- Use the HTML Submit button rather than customized "submit" images.
- Do not use fixed font sizes. These prevent users from increasing font sizes in their browsers, which for some languages they may need to do to make the text legible. It may be necessary to create a separate style sheet for each language.
- Do not assume that any particular font is supported for all languages.

- Allow text to wrap.
- Place checkboxes and radio buttons in separate cells of tables, to ensure that they always align properly.
- Design your HTML pages to accommodate different text lengths.
- Use icons whose meanings are universally recognized.
- Do not assume that colors have universally recognized meanings. Red may not mean "stop" in some cultures, and green may not mean "go".

Diacritical Marks in SEO URLs

Search engine optimized (SEO) URLs can contain diacritical marks.

Unless you are aware of some language-specific recommendation to remove diacritical marks from URLs, leave them in the URLs.

Part 5

Managing Text in Internationalized Applications

Chapter 10

Text Management for Different Languages

This chapter describes how you can manage and manipulate text in different languages.

Managing Text in Different Languages

The set of text management features that Guided Search supports varies from language to language.

Related Links

Stemming Text

Stemming is the process of reducing words to their stem, base, or root form.

Note: This section applies only to Latin-1 analysis. OLT analysis uses a language specific OLT dictionary for stemming and decompounding.

Guided Search applications support stemming for the following languages:

- Simplified Chinese
- Traditional Chinese
- Dutch
- English
- French
- German
- Italian
- Japanese
- Korean
- Portuguese
- Spanish

All of these languages except Chinese, Japanese, and Korean have predefined stemming files. You can add terms to and remove them from the predefined stemming files; for information about how to do this, refer to the *MDEX Engine Developer's Guide*.

Stemming files are generated for Chinese, Japanese, and Korean, during ITL processing.

Decompounding Text

Some words are formed by joining together words that can stand on their own. These compound words can be broken up into their component words, so that the compound word is included in the search results for any of its component words. The process of breaking up a compound word in this way is known as *decompounding*.

In order to decompound a word, language analysis requires that all the components of the word be in the dictionary. For example, a language analysis decompounds the word "Bananenstecker" (banana plug) into "Bananen" and "Stecker" only if both of these words are in the dictionary.

Compound words are common in most Germanic languages (German, Norwegian, and Swedish), as well as in Japanese.

Creating a Thesaurus for a Multi-Lingual Application

Each MDEX Engine has only one thesaurus. The thesaurus is used for all records processed by the MDEX Engine, whatever their languages.

If your MDEX Engine processes records in more than one language, avoid adding words to the thesaurus that have different meanings in these languages. English and French in particular each include a large number of words that are spelled the same, or almost the same, as in the other language, but that have an entirely different meaning. For example, in English "chair" means a piece of furniture; in French, "chair" means "flesh".

Wildcarding Text in a Multi-Lingual Application

Wildcarding is the use of characters that can be matched by any other characters. Wildcarding is supported only by Latin-1 language analysis, and not by OLT analysis.

Configuring Language-Specific Spelling Correction

To prevent queries in one language from being spell-corrected according to the conventions of a different language, you must configure spelling correction for each particular language.

You configure spelling correction for particular languages when you tag your data with language IDs. Guided Search generates a language-specific dictionary for any data that has been tagged with a language ID. To find the correct spellings in the language specified by a particular language ID, the spelling correction feature consults the dictionary for that language.

Note: Spelling correction can be used only with languages that are written in alphabetic scripts. Thus, spelling correction is not supported for Chinese, and has limited application to Japanese and Korean, owing to the non-alphabetic nature of the scripts in which these languages are written.

If all properties within a Search interface are in the same language, spelling correction will correct words by suggesting other words in these properties. If you use record filters to return records only in a particular language, spelling correction will correct words only by suggesting words that occur in these records.

Selecting an Appropriate Mode for Spelling Correction

In addition to requiring a language-specific dictionary to reference, spelling correction also requires that dgidx be configured to use the proper spelling mode. Select a spelling mode option for dgidx by specifying one of the parameters to the dgidx `--spellmode` option listed in the following table.

Use this Spelling Mode . . .	. . . for this language or type of language
aspell (the default)	English and similar languages for which sound-alike corrections can be made, using phonetic rules. aspell does not perform corrections to non-alphabetic/non-ASCII terms such as café, 1234, or A&M.
espell	Non-English words or terms that are not words, such as such as part numbers; performs non-phonetic (edit-distance-based) corrections.
aspell_OR_espell	Languages that include both ASCII and non-ASCII characters and phrases. Aspell corrects ASCII words and Espell corrects other words.
aspell_AND_espell	Both modules suggest corrections and the user selects the best selection from the union of results.
disable	Chinese or other languages that use non-alphabetic scripts to which the concept of spelling does not apply.

For example, to select the `espell` mode, use the following command:

```
dgidx --spellmode espell
```

You can discover which spelling mode works best for an alphabetic language other than English by testing the following spelling modes with data in that language: `espell`, `aspell`, `aspell_OR_espell`, and `aspell_AND_espell`.

Note: In some cases, you may find it easier to create a separate Oracle Commerce Guide Search application for each language that you are targeting, rather than configuring a single application to manage all languages. For information about the advantages and disadvantages of each approach, see *How Many MDEX Engines Do I Need?* on page 13.

Specifying a Correction Mode in a Configuration File

Follow these steps to specify a correction mode in a configuration file. If such a configuration file exists, it overrides any parameter specified in the dgidx `--spellmode` option.

1. Using any standard text editor, create a file that contains the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SPELL_CONFIG SYSTEM "spell_config.dtd.">
<SPELL_CONFIG>
 <SPELL_ENGINE>
  <DICT_PER_LANGUAGE>
   <ESPELL/>
  </DICT_PER_LANGUAGE>
 </SPELL_ENGINE>
</SPELL_CONFIG>
```

2. Save the file as `<app name>_prefix.spell_config.xml`.

   For more information about the structure of a `spell_config.xml` file, refer to the *Platform Services XML Reference*. See also the `spell_config.dtd` in the MDEX Engine `conf/dtd` directory.
3. Store the file in the directory where you store your project's other XML instance configuration files.
4. Run a baseline update and restart the MDEX Engine with the new configuration file.

Stop Words in an Internationalized Application

A stop word is a commonly used word, such as "the", that a search engine has been programmed to ignore. Each MDEX Engine has only one stop word list. As a result, each stop word will be used for all records processed by the MDEX Engine, whatever their languages.

Thus, if you are using a single MDEX Engine for more than one language, provide a separate version of each stop word for each of the languages that your application supports.

Before you specify a stop word in one language, make sure that it does not appear with the same spelling but a different meaning in the other languages that your application supports. English and French in particular share many such "false cognates." For example, the French word for tea, "thé", can be mistaken for the English word "the", which is commonly designated as a stop word.

Merchandising Keyword Triggers

Keyword redirects send a user's search to a Web page (that is, to a URL).

Like dynamic business rules, keyword redirects use trigger and target values. The user's search is redirected if it contains a keyword (the trigger), and you have provided a rule that redirects any search containing that keyword to a particular URL (the target). These features are applied after navigation filtering.

If your application supports multiple languages and you intend to use a given keyword trigger in each language, you must create a separate rule for the keyword trigger in each language.

For example, if the word "pants" (English) triggers a rule, and the same rule should apply to queries in French and Spanish, then two other rules must be created: one triggered by "pantalones" (Spanish) and one triggered by "pantalon" (French).

For detailed information about how create keyword triggers, refer to the *MDEX Engine Developer's Guide*.

Part 6

Configuring Custom Editors and Workbench

Chapter 11

Creating Language-Specific Versions of Custom Editors and Workbench

Creating Language-Specific Custom Editors

If your implementation supports multiple locales, you can localize your custom editors.

To create language-specific custom editors, you must do the following things:

- Modify your editor's `pom.xml` file.
- Create resource properties files that contain localized strings.
- Modify the editor module.
- Add the `getMessage()` function to your custom editors to retrieve the localized strings.

For information about how to perform these tasks, refer to the *Assembler Application Developer's Guide*.

Customizing Menus with Workbench

If your implementation supports multiple locales, you can localize your custom Workbench extensions and menus.

You can store localized values for the following attributes:

- Extension names, descriptions, and icons

  Extensions enable you to incorporate Web applications related to your Guided Search implementation as plug-ins to Workbench. An extension can be as simple as a static Web page or it can provide sophisticated functionality to control, monitor, and configure your Guided Search applications. Extensions can be hosted on the same server as Workbench or on another server.

- Menu nodes, descriptions, and icons

For detailed information about how to use Workbench to customize these attributes, refer to the *Oracle Commerce Guided Search Administrator's Guide*.

Part 7

Logging and Reporting

Chapter 12

Logging and Reporting in an Internationalized Application

Logging and Reporting

Log messages produced by the Content Acquisition System (CAS), Forge, dgidx, and the MDEX Engine are in English and use the UTF-8 character encoding. Labels and other text in reports (apart from messages) are also in English and use UTF-8 character encoding.

The language and character encoding of reports and error messages are not configurable.

Note: Most common UNIX/Linux shells and terminal programs do not display UTF-8 by default; as a consequence, they display some valid UTF-8 characters as question marks (?). If question marks appear in log messages inexplicably, use the od (octal dump) command in Linux or a UTF-8 compatible display to display the log message in a different format. This will enable you to determine whether the questions marks are simply UTF-8 characters that the shell cannot display, or some other characters.

Part 8

Troubleshooting OLT Language Problems

Finding and Correcting Terms Unknown to OLT

MDEX enables you to generate reports that list the indexed terms (property values or dimension names) that are unknown to OLT. You can then correct the terms so that Oracle Language Technology (OLT) can recognize them and perform linguistic analysis on them.

Finding Indexed Terms That Are Unknown to OLT

This section describes how to generate vocabulary reports that list indexed terms (property values or dimension names) that are unknown to OLT, how to run queries on these reports, and how to correct problems so that OLT can recognize and process them. Unknown terms are reported only for the languages for which OLT analysis is enabled.

Terms can be unknown for any of the following reasons:

- Limitations of OLT: The terms are not included in the OLT dictionary, even though they are valid words in their proper languages. For information about how to customize your OLT dictionary to include unknown terms, see *Creating an auxiliary OLT dictionary* on page 67.
- Incorrect language assignments: The terms belong to a language other that the language associated with the records and/or properties where they occur. For example, "millésime" (French for "vintage") reported as "unknown" in a record incorrectly associated with the German language. For information about how to assign languages to records and properties, see *Assigning Language IDs globally, per record, and per property* on page 31.
- Non-linguistic entities: The terms are non-linguistic entities such as weights and measures, part numbers or stock keeping units (SKUs); for example: 12V, 10x15, mm², 110004A846K. OLT is not designed to recognize non-linguistic entities.

The reports can include the following information about each unknown term:

- The number of times that this term occurs in your indexed data.
- The language associated with the records and/or properties where the term occurs.

Generating a Vocabularly Report of Unknown Terms

You can generate a vocabulary report listing the unknown terms in all your indexed data at baseline update time. You can also automatically generate vocabulary reports of all the unknown terms in the data affected by partial updates, whenever you run the partial updates.

Note: Vocabulary reports are for audit and review only. You cannot modify your OLT dictionaries by editing vocabulary reports; for information about auxiliary OLT dictionaries, see *Creating an auxiliary OLT dictionary* on page 67.

Generating Vocabulary Reports at Baseline Update Time

You can specify that a vocabulary report be generated whenever you run a baseline update. To do this, add the `--vocabulary-report` option to the dgidx command line:

```
dgidx --vocabulary-report
```

If you are running dgidx from the Deployment Template, add an `<arg>--vocabulary-report</arg>` element to the `DataIngest.xml` file; for example:

```
<dgidx id="..." host-id="...">
  . . .
  <args>
    <arg>--vocabulary-report</arg>
    . . .
  </args>
</dgidx>
```

The report is written to a file whose name is of the form:

```
db_prefix.vocabulary_report.xml
```

where:

`db_prefix` is the name of the project; for example:

```
Discover.vocabulary_report.xml
```

The file is written to the directory specified in the `<output_dir>` element of the `DataIngest.xml` file; for example:

```
<output-dir>data/dgidx_output</output-dir>
```

Thus, the vocabulary report in this example would be written to a file with the following name in the following directory:

```
[appdir]/data/dgidx_output/Discover.vocabulary_report.xml
```

Note: In this guide, the abbreviation `[appdir]` stands for the directory that contains your application.

Generating Vocabulary Reports at Partial Update Time

To generate reports that include only the unknown terms added to your spelling dictionary through partial updates, add the `--vocabulary-report` option to the dgraph command line:

```
dgraph --vocabulary-report
```

A dgraph vocabulary report is generated whenever you run a partial update successfully. The report includes only those unknown terms that occur in records introduced or affected by the partial update.

If you are running Dgraph from the Deployment Template, add an <arg>--vocabulary-report</arg> element to the DgraphDefaults.xml file; for example:

```
<dgraph-defaults>
  <args>
    <arg>--vocabulary-report</arg>
    . . .
  </args>
  . . .
</dgraph-defaults>
```

The report is written to a file whose name is of the form:

```
<db_prefix>.vocabulary_report.v<VERSION>.xml
```

where:

db_prefix is the name of the project

<*VERSION*> is the generation number of the committed partial update.

The file is written to the directory specified in the <input_dir> element of the AuthoringDgraphClus¬ter.xml and LiveDgraphCluster.xml files; thus, the vocabulary report for a partial update might be written to a file with the following name in the following directory:

```
[appdir]/data/dgraphs/AuthoringDgraph/dgraph_input/Discover.vocabulary_re¬
port.v27.xml
```

Sample Queries on Vocabulary Report

The following XQuery scripts illustrate how to query vocabulary reports for commonly useful information.

Sample 1: Find the Fifty Most Frequently Occurring Unknown Terms

The following XQuery code queries vocabulary_report.xml for the fifty most frequently occuring unknown words. This information can identify instances of language misconfiguration and words needing OLT customization.

```
declare namespace ene="http://xmlns.endeca.com/ene/dgraph";
  let $x := doc("vocabulary_report.xml")
  let $sorted-unknowns :=
    for $term in $x//ene:terms[@class="unknown"]/ene:term
    order by number($term/@count) descending
    return <unknown>{$term/../../@lang} {$term/@count} {$term/@value}</unknown>
  (: Return the unknown terms as a table of count-lang-term tuples. :)
  (: Sort by frequency of occurence and limit to top 50. :)
  let $tab := "&#9;"
  for $unk in subsequence($sorted-unknowns, 1, 50)
  return fn:concat($unk/@count/string(), $tab, $unk/@lang/string(),
    $tab, $unk/@value/string())
```

The following is an example of the vocabulary report from a Swedish-English technical catalog that was analyzed by the Sample 1 XQuery and shows unknowns that fall into a variety of categories:

```
  159711 sv false
     496 sv mp3
     270 sv 12v
      55 sv creative
      51 sv microfiber
      36 sv gummiklätt
```

You might customize the OLT dictionary in response to this information as follows:

- The word "false" is English metadata in the product catalog that is being tokenized as Swedish. Because this metadata is never searched by customers, no customization of the OLT dictionary is needed.
- The words "mp3" and "12v" are common abbreviations that are only expected to match exactly and do not need customization.
- The words "microfiber", a technical term, and "gummiklätt", a compound, are candidates for customization of the Swedish OLT dictionary. Unless these words are added to the OLT dictionary, they will be matched only by an exact match (or thesaurus entry); matching by inflection or component parts will not occur.
- The English word "creative" may be in a property that is intended for English search but is mistakenly associated with Swedish. If this is the case, there will be more English words as unknowns. Analyzing the number of unknowns by property can be used to identify properties that may have a high number of unknowns due to a misconfiguration of the language assumed for the property.

Sample 2: Display the Properties that Include the Ten Most Frequently Occurring Unknown Terms

The following XQuery code queries `vocabulary-report.xml` for the properties with the ten most frequently occurring unknown words. This information is useful for identifying language misconfiguration.

```
declare namespace ene="http://xmlns.endeca.com/ene/dgraph";
  let $x := doc("vocabulary_report.xml")
  let $sorted-unknowns :=
    for $prop in $x//ene:terms[@class="unknown"]/ene:by_property
    order by number($prop/@count) descending

    return <unknown>{$prop/../../@lang} {$prop/@count} {$prop/@name}</unknown>
  (: Return the unknown terms as a table of count-lang-property tuples.:)
  (: Sort by frequency of occurence and limit to top 10. :)
  let $tab := "&#9;"
  for $unk in subsequence($sorted-unknowns, 1, 10)
  return fn:concat($unk/@count/string(), $tab, $unk/@lang/string(),$tab,
$unk/@name/string())
```

The following is an example of the vocabulary report from a Swedish-English technical catalog that was analyzed by the Sample 1 XQuery and shows unknowns that fall into several different categories:

```
   30446 sv ProductDescription_en
   28195 sv ProductStockStatus
   10286 sv ProductImageURL
     345 sv ProductDescription_sv
```

You might customize the OLT dictionary in response to this information as follows:

- The "ProductStockStatus" and "ProductImageURL" fields contain metadata that is not involved in text search, so the number of unknowns is not an issue.
- The "ProductDescription_en" and "ProductDescription_sv" fields are text searchable. In this case, the property "ProductDescription_en" was incorrectly configured as Swedish. Reconfiguration of the language associated with ProductDescript_en will correct the high unknown count.

Appendix A

Latin-1 and OLT Language Analysis

This appendix compares Latin-1 and OLT language analysis, including their effects on various features of Oracle Commerce Guided Search, and provides general guidance about which type of language analysis may be appropriate for your application.

Latin-1 language analysis

Latin-1 language analysis is available for all languages except Chinese (Simplified, Traditional), Japanese, and Korean. It is enabled by default for the following languages: Dutch, English, English (UK), French, German, Italian, Portuguese, and Spanish.

It supports the following features:

- Tokenization: Dividing text into words, phrases, symbols, or other meaningful elements delimited by spaces.
- Wildcard search: Searching for phrases that include characters (wildcards) that match all characters.
- Phrase search: Searching for exact matches of a particular string.
- Search characters: Searching for particular characters.
- Diacritic folding: Ignoring accent marks when indexing and searching: for example, treating "Furtwängler" and "Furtwaengler" as matching terms.
- Static stemming: Matching the base (uninflected) form of a word; for example, matching "box" to "boxes".
- Stop words: Common words (such as "the", "and", or "while") that have no value for searching.

It performs no form of analysis that is specific to any language.

Oracle Language Technology (OLT) language analysis

Oracle Language Technology analysis performs language-specific dictionary-based forms of linguistic analysis, including the following:

- Segmentation: Identifying word breaks in text from languages that do not use whitespaces as word delimiters. Formerly unseparated words must be contiguous to each other and in the same property. Note that Latin-1 analysis is unsuitable for languages that do not use whitespaces as delimiters.
- Tokenization: Breaking a stream of text up into words, phrases, symbols, or other meaningful elements.
- Orthographic normalization: Accounting for variations in the representation of words in languages that have standardized alternatives to diacritic marks (such as "ae" or "a" for ä in German); for example, treating "Furtwaengler" and "Furtwangler" as matching terms.

- Decompounding: Dividing compound word forms into their base terms; for example, dividing "Altertumswissenschaft" into "Altertums" and "Wissenschaft".
- Diacritic folding: Ignoring character accents in data when indexing and searching text.
- Dynamic stemming: Determining the base (uninflected) form of a word. The process is based on dictionary entries and language specific rules.
- Stop words: Common words (such as "the", "and", or "while") that have no value for searching.

A single MDEX Engine can process any number of the originally supported languages whose default language analysis is OLT; for example, a single MDEX Engine can process data in Arabic, Finnish, and Hebrew. However, among the languages that were not originally supported, a single MDEX Engine can process only one language whose default analysis is OLT.

The management of the originally supported languages can be configured in the file `stemming.xml`.

For a complete list of the languages supported by the MDEX engine, see *MDEX Engine Originally Supported Languages* on page 71.

Note: OLT analysis is only partially compatible with Oracle Commerce record and dimension search features; for example, it does not support wildcard search, phrase search, and search characters. If your application requires these search features, use Latin-1 analysis.

Different releases of the MDEX Engine may include different versions of OLT. To find out which version of OLT the MDEX Engine uses, enter the `--version` option for the Dgidx or Dgraph at the command line.

Note: Only one type of language analysis can be applied to any particular record, dimension, or property.

Auxiliary dictionaries for OLT analysis

You can optionally add an auxiliary dictionary to supplement the primary OLT dictionary for any supported language. This may be necessary if searches for terms that exist in your data are not producing the expected results.

The auxiliary dictionary is a UTF-8 encoded file that is line oriented and tab delimited. Each line in the file represents an entry to supplement the primary dictionary.

Entries to an auxiliary dictionary are of the following form:

```
COMMAND value1 value2 ...
```

Specify `STEM` or `COMPOUND` for *`COMMAND`* .

Using the STEM command

Each line beginning with `STEM` includes a term that represents the uninflected stem (or, lemma) of a word, and one or more attributes that identify the part(s) of speech (POS) of the word. The POS attributes must be separated from each other by commas (with no spaces). The command name STEM and the new term must be separated from each other and from the POS attbributes by tabs.

```
STEM  new_term1 POS,POS,POS,...
STEM  new_term2 POS,POS,POS,...
STEM  new_term3 POS,POS,POS,...
```

The POS attributes enable Guided Search to identify the possible inflectional endings of the new term in its given language.

You can specify the part of speech (POS) attributes by their full names or by abbreviations of their names (listed here in parentheses):

- `noun (N)` - A simple noun, like table, book, procedure
- `nounProper (propN)` - A proper name of a person, a place, and so on, that is typically capitalized, such as Zachary, Supidito, Susquehanna
- `verb (V)` - Any verb in its dictionary form, such as deconstruct, upsell, or skate
- `adjective (Adj)` - Modifiers of nouns, typically can be compared (green, greener, greenest), such as fast, trenchant, pendulous
- `adverb (Adv)` - Any general modifier of a sentence that may modify an adjective or verb or may stand alone, such as slowly, yet, perhaps
- `preposition (Prep)` - A word that forms a prepositional phrase with a noun, such as off, beside, from. Also used for postpositions in languages that have postpositions of similar function.
- `punct (Punct)` - Any non-letter symbol that is treated as a unit by itself, such as %, $,]
- `pronoun (Pro)` - Any pronominal form, including personal pronouns (I, they), demonstrative pronouns (those, this), relative pronouns (who, which, wherever)
- `interrog (Wh)` - An interrogative word, such as who, why, when, where, how
- `determiner (Det)` - Words that carry grammatical information about a noun group, for example definite/indefinite, such as the, a, an
- `particle (Part)` - Small, invariant words that convey grammatical information; also used for interjections.
- `conjunction (Conj)` - Conjunctions that introduce subordinate clauses, such as although, because, while; and conjunctions that introduce coordinate clauses, such as and, or, yet
- `numCardinal (Card)` - Cardinal numbers, like thirteen, 100, five
- `numOrdinal (Ord)` - Ordinal numbers, like thirteenth, 100th, fifth

For example, the following German auxiliary dictionary shows three entries. Each entry is marked with the attribute `N` to indicate it is a noun:

```
STEM aalglatt N
STEM ausrüster N
STEM verdränger N
```

Decompounding

You can manually configure an auxiliary dictionary to define components of compound words. This can be useful if existing language dictionaries do not reflect the usage of the language in a region or market, or if existing libraries have not kept up with changes to the language.

For example, the German orthography reform of 1996 introduced a standard set of rules for compound words, but these rules are not always followed. For this and similar cases, you may decide explicitly to configure dictionary entries that mark the divisions within compound words. For more information, see *Auxiliary dictionaries for OLT analysis* on page 66.

Segmentation

Word segmentation in languages that do not include spacing between words (Chinese, Japanese, Korean, and Thai) is handled through OLT and is based on the lemmas, or base words, defined in the stemming dictionary. If you are seeing incorrect search results for these languages, you can provide an auxiliary or custom OLT stemming dictionary to produce correct search results.

Creating an auxiliary OLT dictionary

To create an auxiliary dictionary:

1. Start a text editor that supports UTF-8 characters and enables you to edit the language that you want to supplement.
2. Create a new UTF-8 encoded file.
3. Add terms to the dictionary. Start each term on a separate line that begins with the command `STEM` or `COMPOUND`, followed by the word or character, any optional attributes, and then a carriage return.
4. Save the dictionary file with the filename `dictionary.<RFC3066 language code>.dict` in the `%ENDECA_MDEX_ROOT%\olt` directory on Windows or in `$ENDECA_MDEX_ROOT/olt` on UNIX.

   Note: The dictionary name should correspond to the `--lang` value you pass to the dgidx. If you are using a supported language that includes a region code, include it in the filename. For example: `%ENDECA_MDEX_ROOT%\olt\dictionary.zh-CN.dict`.

5. Re-index your data and specify the `--lang` flag to dgidx with appropriate `<lang Id>` value.
6. Restart the dgraph.

The dgidx and dgraph load custom dictionaries for all languages configured in the `stemming.xml` file.

Configuring decompounding in an auxiliary dictionary

You can configure a language-specific stemming dictionary to define the components of compound words. This process is known as *decompounding*. If decompounding is enabled (USE_COMPOUND_MATCHING="TRUE" in stemming.xml), then searching for a word matches both that word as well any compounds that include the word as a component.

Each of the individual components of a decompounded word must be a lemma -- that is, an uninflected word stem. If a lemma that you need to specify as a component of a compound word does not exist in your dictionary, you must add the lemma to your dictionary. You can add the lemma to your dictionary using the STEM command.

To customize decompounding:

1. Open the dictionary file that you wish to modify.

   For example, `%ENDECA_MDEX_ROOT%\olt\dictionary.de.dict`.

2. Locate or add the terms that you wish to configure for decompounding.

   For example, you may wish to customize the decompounding of the German word, "Binnenschiffahrt," which refers to transport along inland rivers.

3. Add the entry using a command of the form `COMPOUND` *`word stem1|stem2|… POS,POS …`*

   where:

   - *word* is the compound word that you are adding to the dictionary. You must specify the uninflected stem (lemma) of the word.
   - *stem1* is the first component of the compound word.

     Note: All components must be lemmas. For example, you can specify the German word "Arbeit" (work) as a component, because this form of the word is the word's lemma. You cannot, however, specify "Arbreits" (of work), because this form is not the lemma.

   - *stem2*, *stem3*, and so on are the remaining components of the compound word.
   - *POS*, the part of speech of *word*, such as N (noun), Adj (adjective), and so on. By specifying the part of speech of *word*, you enable the search feature to determine the set of inflectional endings to look for.

In the case of the above example, you could add the word in either or both of two ways: one that adheres to the German orthography reform standards of 1996, and one that reflects the earlier spelling of the word:

```
COMPOUND  Binnenschifffahrt   Binnen|Schiff|Fahrt N
COMPOUND  Binnenschiffahrt    Binnen|Schiff|Fahrt N
```

If a compound word is composed only of lemmas, you can add the word to your dictionary using either a COMPOUND command or a STEM command of the following form:

STEM word *stem1*|*stem2*|... POS,POS,...

For example, you can add the compound Dutch word appelsalade, which is composed of two lemmas, using either a STEM or a COMPOUND command, as follows:

```
COMPOUND appelsalade appel|salade N
STEM appel|salade N
```

You cannot use STEM to add a Dutch word such as sperziebonensalade ("green bean salad"), however, because one of its components, "bonen", ("beans") is an inflected form (plural), and thus is not a lemma. You can, however, add Sperziebonensalade using the following COMPOUND command:

```
COMPOUND sperziebonensalade sperzie|boon|salade N
```

where "boon" is the lemma of the Dutch word for "bean".

The above COMPOUND command enables sperziebonensalade to be found by searches for "boon" or "bonnen" (or "salade", "saladen" or "sperzie") if you specify USE_COMPOUND_MATCHING="TRUE" in `stemming.xml`. If USE_COMPOUND_MATCHING="FALSE", then defining the sperziebonensalade as a compound does not have any effect on searches (the searches do not use the components) but will not cause an error.

4. Save and close the file.

Mapping accented characters to unaccented characters

dgidx supports mapping Latin1, Latin extended-A, and Windows CP1252 international characters to their simple ASCII equivalents during indexing. You can optionally specify the `--diacritic-folding` flag to dgidx to map accented characters to simple ASCII equivalents. This mapping allows the dgraph to match Anglicized search queries such as *cafe* against result text containing international characters (accented) such as *café*.

Appendix B

Language Reference

MDEX Engine Originally Supported Languages

The following table lists the languages originally supported by the MDEX Engine, as well as the default language analyzer and language code for each.

The MDEX Engine supports other languages in addition to those that it supported originally, and it treats these languages differently when it chooses a language analysis to apply to them. For information about how the MDEX Engine chooses a language analysis for a language, see *Analyzing and Sorting* on page 27.

Language	Default Analyzer	Language Code
Arabic	OLT	`ar`
Catalan	OLT	`ca`
Chinese (Simplified)	OLT	`zh-CN`
Chinese (Traditional)	OLT	`zh-TW`
Croatian	OLT	`hr`
Czech	OLT	`cs`
Danish	OLT	`da`
Dutch	Latin-1	`nl`
English	Latin-1	`en`
English (United Kingdom)	Latin-1	`en-GB`
Finnish	OLT	`fi`
French	Latin-1	`fr`
German	Latin-1	`de`
Greek	OLT	`el`
Hebrew	OLT	`he`
Hungarian	OLT	`hu`
Italian	Latin-1	`it`

Language	Default Analyzer	Language Code
Japanese	OLT	ja
Korean	OLT	ko
Norwegian (Bokmal)	OLT	nb
Norwegian (Nyorsk)	OLT	nn
Persian (Farsi)	OLT	fa
Polish	OLT	pl
Portuguese	Latin-1	pt
Portuguese (Brazil)	OLT	pt-BR
Romanian	OLT	ro
Russian	OLT	ru
Serbian	OLT	sr
Serbian (Latin)	OLT	sr-Latn
Slovak	OLT	sk
Slovenian	OLT	sl
Spanish	Latin-1	es
Swedish	OLT	sv
Thai	OLT	th
Turkish	OLT	tr

Index

I

J

K

L

M

N

O

P

R

S

T

U

W

X

Z