Verity Locale Configuration Guide

Version 6.1.1

February 10, 2006
Part Number DM0710

Verity, Incorporated
894 Ross Drive
Sunnyvale, California 94089
(408) 541-1500

Verity Benelux BV
Coltbaan 31
3439 NG Nieuwegein
The Netherlands

Contents

Figures, Tables, and Listings

Preface

Welcome to the *Verity Locale Configuration Guide*. This book is for administrators and developers of Verity K2 applications. It is intended for readers who need to know how to administer or develop an application that supports indexing and search in multiple languages.

This preface contains the following sections:

Using This Book

This section briefly describes the organization of this book and the stylistic conventions it uses.

Version

The information in this book is current as of K2 Enterprise version 6.1.1. The content was last modified February 10, 2006. Corrections or updates to this information may be available through the Verity Customer Support site; see "Verity Technical Support" on page 17.

Organization of This Book

This book contains the following chapters and appendixes:

- **Chapter 1**, **"Installing Verity Locales,"** describes how install Verity locales.
- **Chapter 2**, **"Language Concepts,"** gives an overview of the Verity internationalization architecture, introduces language concepts related to text search, and illustrates how locale and character set are involved in indexing and searching.
- **Chapter 3**, **"Verity Locales,"** describes the language-handling characteristics of each Verity locale.
- **Chapter 4**, **"Configuring Locales,"** describes how to configure and use Verity locales to support indexing and search in languages other than English.
- **Chapter 5**, **"Locale Issues for Applications,"** gives suggestions for creating effective language-aware K2 applications.
- **Appendix A**, **"Locales, Character Sets and Languages,"** lists the locales, character sets, and language codes supported by Verity.
- **Appendix B**, **"Tokenization Delimiters,"** lists the characters that can be used as searchable symbols or as word delimiters to control indexing.
- **Appendix C**, **"Customizing Language Dictionaries,"** describes how to create customized stemming dictionaries for Asian and non-Asian languages.
- **Appendix D**, **"The Language ID Command Tool,"** describes how to use a command-line tool that identifies the language of a document.

Stylistic Conventions

The following stylistic conventions are used in this book.

Convention	Usage
Plain	Narrative text.
Bold	User-interface elements in narrative text: ■ Click **Cancel** to halt the operation.
Italics	Book titles and new terms: ■ For more information, see the *Verity K2 Getting Started Guide*. ■ An *index* is a Verity collection, parametric index, or recommendation index.
`Monospace`	File names, paths, and code: ■ The `name.ext` file is installed in: `C:\Verity\Data\`
`Monospace italic`	Replaceable strings in file paths and code: ■ `user` *`username`*
`Monospace bold`	Data types and required user input: ■ **`SrvConnect`** A connection handle. ■ In the **User Interface** text box, type **`user1`**.

The following command-line syntax conventions are used in this book.

Convention	Usage
`[ optional ]`	Brackets describe optional syntax, as in `[ -create ]` to specify a non-required option.
|	Bars indicate "either | or" choices, as in `[ option1 ] | [ option2 ]` In this example, you must choose between `option1` and `option2`.
`{ required }`	Braces describe required syntax in which you have a choice and that at least one choice is required, as in `{ [ option1 ] [ option2 ] }` In this example, you must choose `option1`, `option2`, or both options.
`required`	Absence of braces or brackets indicates required syntax in which there is no choice; you must enter the required syntax element.
`variable`	Italics specify variables to be replaced by actual values, as in `-merge` *`filename1`*
`...`	Ellipses indicate repetition of the same pattern, as in `-merge` *`filename1`*`,` *`filename2`* `[,` *`filename3`* `... ]` where the ellipses specify `,` *`filename4`*, and so on.

Use of punctuation—such as single and double quotes, commas, periods—indicates actual syntax; it is not part of the syntax definition.

Related Documentation

Most Verity books contain information related to languages and character sets. See also the following books for general information on Verity products and language-related features:

- *Verity K2 Getting Started Guide*
- *Verity K2 Developer Getting Started Guide*
- *Verity Query Language and Topic Guide*
- *Verity Collection Reference*

Verity Technical Support

Verity Technical Support exists to provide you with prompt and accurate resolutions to difficulties relating to using Verity software products. You can contact Technical Support using any of the following methods:

Telephone: (403) 294-1107

Fax: (403) 750-4100

Email: tech-support@verity.com

Web: `http://www.verity.com`

Product documentation, release notes, and document updates are available on the Verity Customer Support Site, at

`https://customers.verity.com`

It is recommended that you periodically check the Customer Support site for the existence of updates to this and other Verity product documents.

Access to the contents of the Customer Support site requires a user name and password. To obtain a user name and password, follow the signup instructions on the Customer Support site home page. You will need to supply your Verity entity ID and Verity license key.

1

Installing Verity Locales

This chapter gives instructions for running the Verity Locales installer, which installs one or more locales to provide language-specific indexing and searching capabilities.

This chapter includes the following sections:

Prerequisites

The following features must be present on the machine on which you run the Locales installer.

Supported Operating Systems

Locales and internationalization are supported on all Verity platforms, including Windows NT/Windows 2000, Solaris, AIX, HP-UX, and Linux.

Operating-System Localization

If the operating system on which Verity runs is localized to a different language from the Verity locale being used, query logging can in some cases be unreadable. If you plan to install and use only a single Verity locale, it is recommended that you also install an operating system that is localized to match that locale.

Verity Platform Installation

Before running the Locales installer, you already must have installed K2 Services, Verity Intelligent Classifier (VIC), or Verity Development Kit (VDK).

Installer Requirements

- The Locales installer is Java-based and requires JDK 1.4.2 or later.
- To install Verity Locales, you must have at least 1,000 MB of available disk space.
- The `JAVA_HOME` environment variable needs to be set to your JDK before you run the installer.

International Fonts

Strictly speaking, international fonts need not be present for installing locales or for using them to build collections. However, if you are building a collection and need to view the results, or if you are creating style files or BIFs to use in creating the collection, you'll need to have fonts available that can display the locale's characters. End-users' browsers, of course, must have fonts that can display the characters of any collections they might expect to search.

License Requirements

Each licensed locale, or set of locales, is enabled by a single license key. You cannot run the installer without providing a valid license-key string. Your Verity representative provides you with the appropriate license key for the locales and language features you are licensed for.

Multilanguage Locale and Single-Language Locales

The Verity Locales product includes

- The multilanguage locale
- Asian locales
- Advanced European locales

For the multilanguage locale, two levels of capability are available:

- The basic level is unlicensed. It provides advanced support for the English language, plus basic indexing and searching in any of the Unicode language ranges. This level of capability is available with every Verity installation and does not require running the Locales installer.

  At this level, non-English documents display properly but might not be indexed or searched correctly.

- The advanced level is installed as part of the Verity Locales product. Installation of this level requires a license key to be supplied when running the Locales installer. This level of capability provides linguistic capabilities for any of the European, middle-eastern, and Asian languages listed in Table 3-1 on page 64.

The license key you supply when running the installer also specifies which of the European and Asian single-language locales you are licensed for. These locales provide advanced support for the languages listed in Table 3-3 on page 67 and Table 3-5 on page 69, respectively. Only the individual locales for which you are licensed are installed.

A runtime license file is required by the multilanguage locale. It is named `locale.lic`, in the directory *`productDir`*`/common`, and it contains a single license string. If this license file is not present or does not contain a valid key, linguistic capabilities are not available.

All languages also support a client dictionary, which allows users to define their own tokens. For more information, see "Customizing Language Dictionaries" on page 147.

Legacy European Locales

These single-language locales are installed as part of the Verity Single-Language European Locales product, and they are available only to certain existing customers. Installation of these locales requires a license key to be supplied when running the Locales installer. These locales provide linguistic capabilities for the Western European languages listed in Table 3-7 on page 72.

Running the Locales Installer

For either UNIX or Windows platforms, and for either the Verity Locales product or the Verity Single-Language European Locales product, the DVD-ROM includes the locales installer program (launched with `install.sh` on UNIX platforms, `setupwin32.exe` for Windows) plus locale binary files and language data.

Before starting, see "Installer Requirements" on page 20. Then follow these instructions to use the installer.

1. Launch the installer:

   - **For the Verity Locales product:**

     Place the K2 product family DVD or the VDK DVD into the DVD-ROM drive of the machine that already has the installed Verity platform component (K2, VIC, or VDK). Open the DVD and navigate to the `locales` directory. Double-click `setupwin32.exe` (on Windows) or execute `install.sh` (on UNIX) to launch the Locales installer.

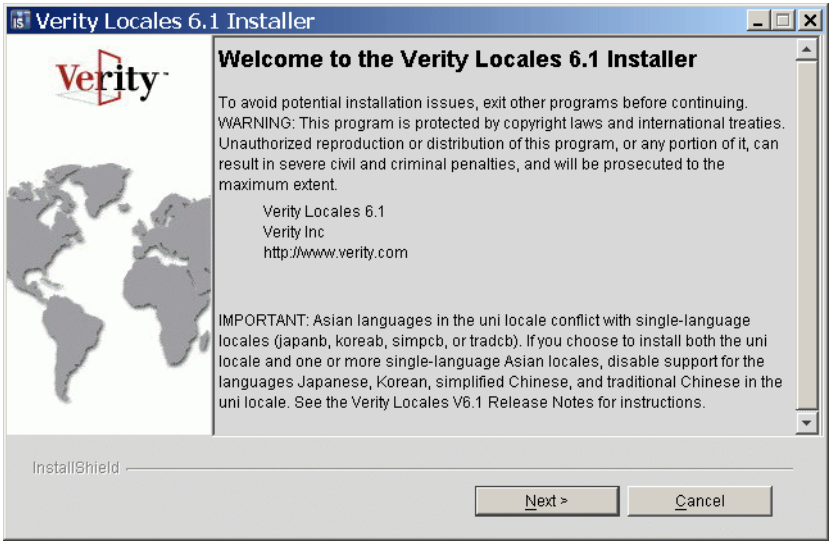
   - **For the Verity Single-Language European Locales product:**

     Place the Single-Language European Locales installation DVD into the DVD-ROM drive of the machine that already has the installed Verity platform component (K2, VIC, or VDK). Open the DVD and launch the Locales installer (double-click `setupwin32.exe` or execute `install.sh`).

   (To run the installer in console (text) mode on UNIX platforms, execute `install.sh -console`.)

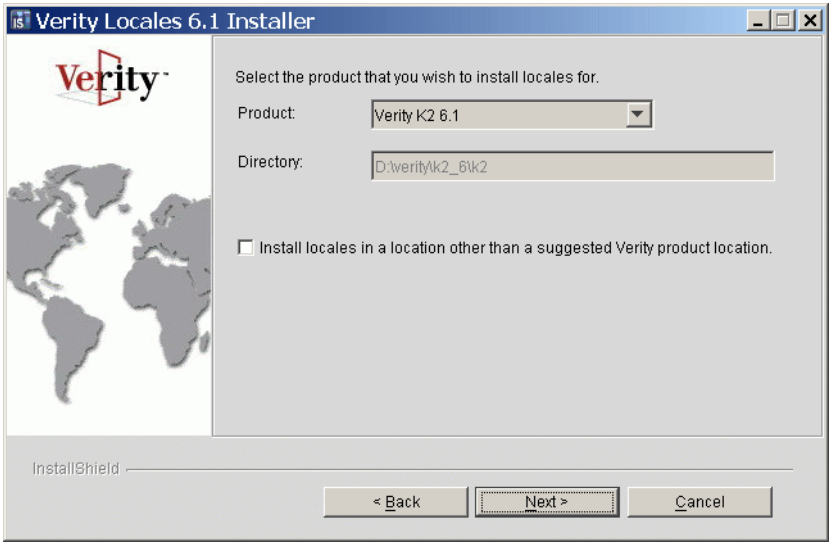
   The Welcome screen appears.

Note The following screens are from the Verity Locales installer. Screens for the Verity Single-Language European Locales installer are similar, although the list of licensed locales and languages is different.



2. Read the Welcome screen and click **Next**. The License Language screen appears.
3. Select the language for the license agreement, then read the agreement on the License Agreement screen.
4. Read the license agreement, click the **I accept...** button, then click **Next**.

   The Product Selection and Destination Folder screen appears. The default path shown in the Destination Folder area is the appropriate installation location based on the previously installed Verity component (K2, VIC, or VDK).

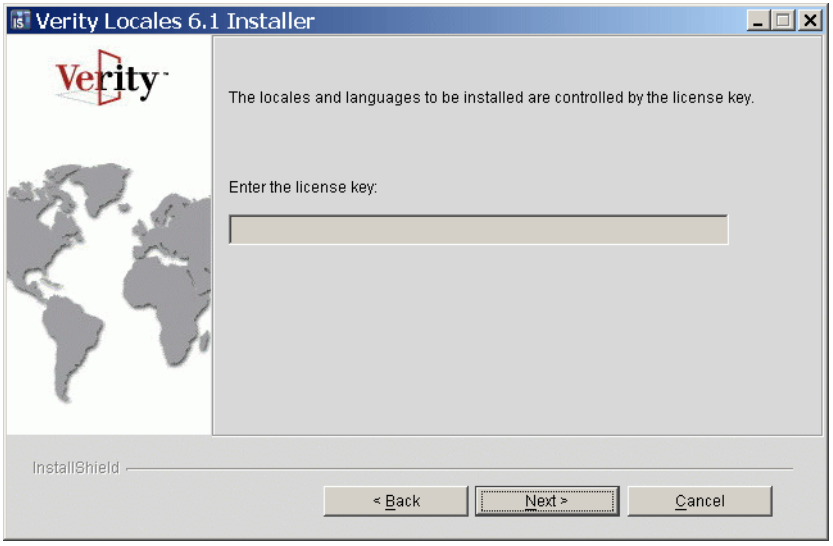


5. Accept the default installed Verity product as shown, or select an installed Verity product to associate these locales with.
6. Accept the default Locales installation directory as shown, or type in the desired directory. It should be the directory *`productDir`*, where *`productDir`* is the path to specific installed Verity product (such as `usr/verity/k2_6/k2` or `usr/vdk`).

   (It is also the directory immediately above the `common` directory.)
7. Click **Next**.

   A screen appears that cautions you to stop all K2 services before proceeding.
8. Click **Next**.

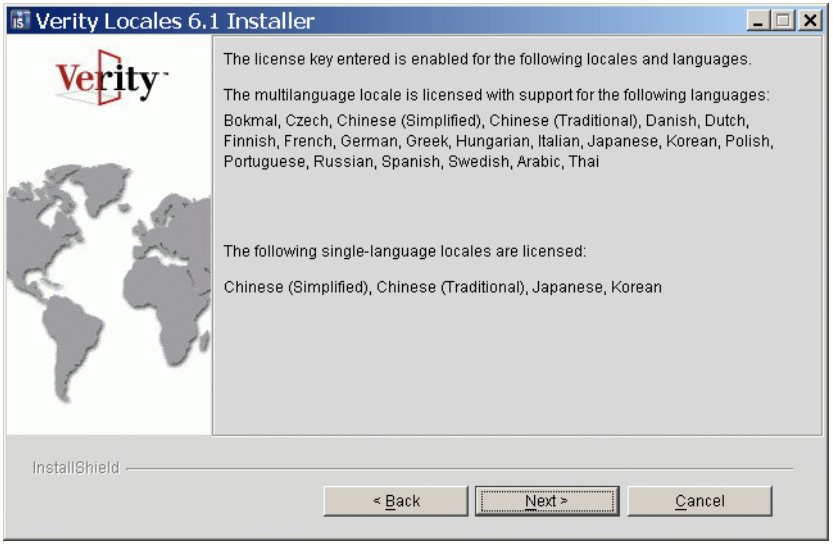
The License Key screen appears.



9. Type the license-key number for the locale or locales you are installing. Note that the key determines the set of locales and languages to be installed.

10. Click **Next**.

    The Locales Verification screen appears. It lists all the locales and languages enabled by the license key entered in the previous screen. (The list may include locales or languages not installed by the currently running installer.)

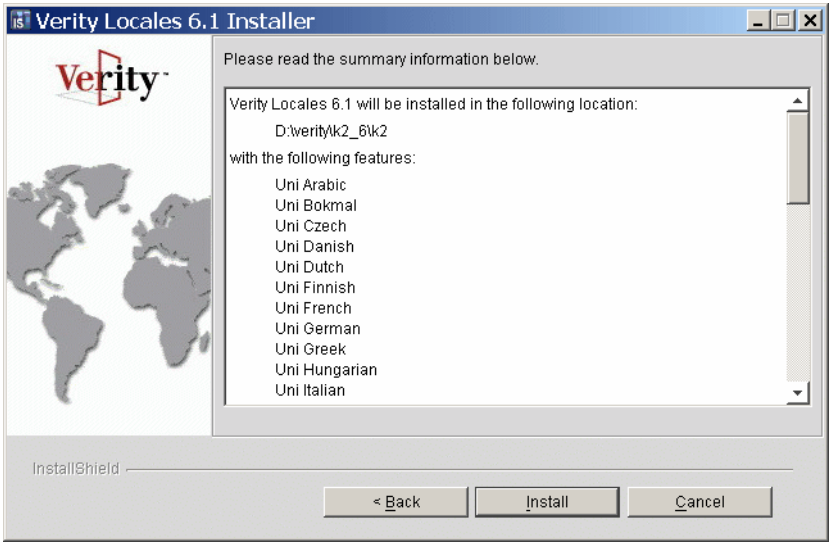


11. If the list of languages and locales is correct, click **Next**. Otherwise, click **Back** to correct your license-key entry before proceeding.

12. Click **Next**.

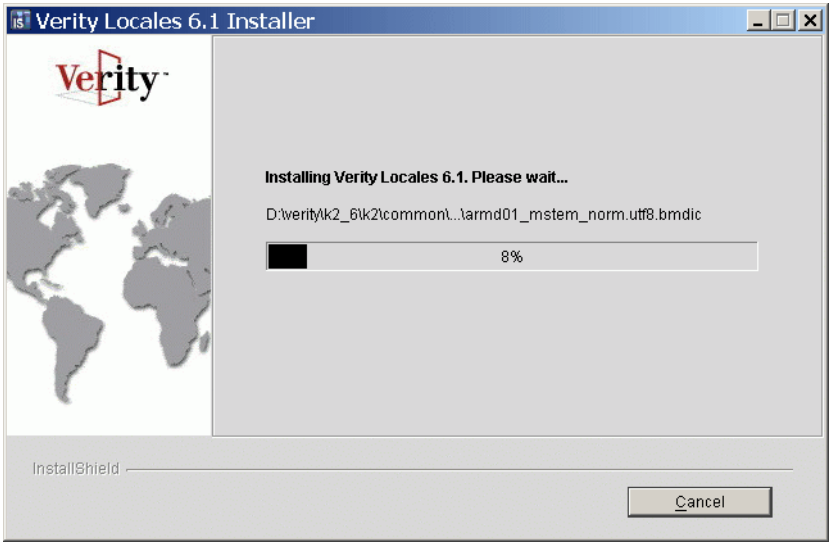
    The Summary screen appears, listing the components to be installed and showing the installation directory path.

    - For the Verity Locales installer, the list may include one or more languages for the multilanguage (`uni`) locale, and/or one or more Asian or advanced European single-language locales.
    - For the Verity Single-Language European Locales installer, the list includes one or more legacy European locales.

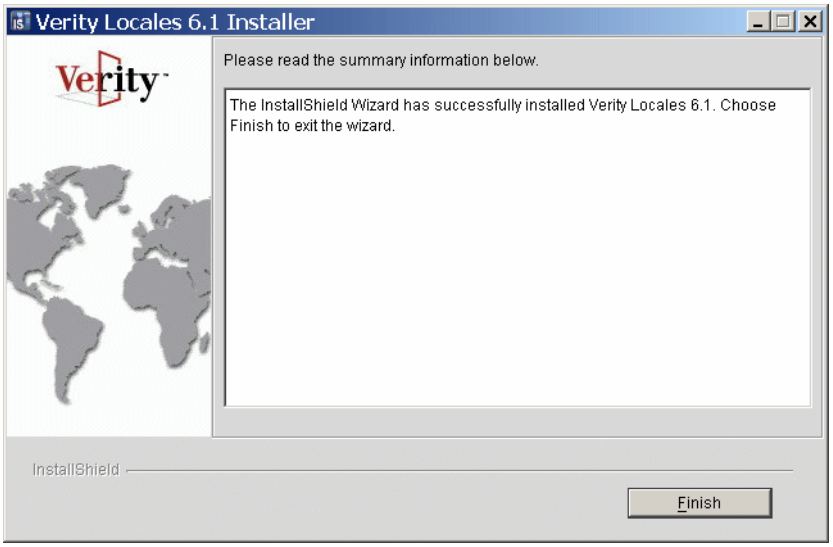


13. If the information is correct, click **Next**. Otherwise, click **Back** to correct the information before proceeding.

    The Copying Files screen appears. It displays a progress bar while it copies the files to your system.



If copying completes successfully, a confirmation notice appears.



14.Click **Finish** to complete the installation.

Installing Additional Locales or Languages

After installing one or more locales, you can run either of the Locales installers again to add more locales (or to add support for more languages to the multilanguage locale).

IMPORTANT Before running the Locales installer to add support for additional languages to the multilanguage locale, you must first shut down all K2 services.

Obtain a new license key from Verity that specifies the complete set of locales and languages that you will be licensed for. Then re-run the appropriate Locales installer and enter your new license key. The installer adds or removes locales to match the new key.

Uninstalling Locales

To completely remove all locales, you run one or both of the Locales uninstallers (depending on what you have installed). Take these steps:

1. Launch the appropriate uninstaller:

   - On Windows, open the Add/Remove Programs control panel and choose

     - Verity Locales 6.1 to remove all single-language locales (other than `englishv`) and all language support (other than English) from the multilanguage locale.

     - Verity Single-Language European Locales 6.1 to remove all legacy European locales.

   - On UNIX, make sure your `JAVA_HOME` variable is set and in the system path, then launch the uninstaller for your locales product:

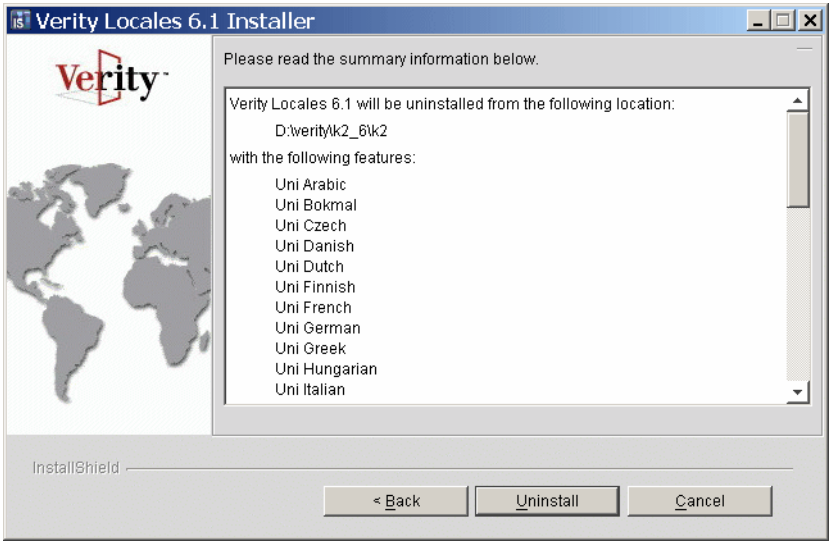
     - *Verity Locales:* launch the file `localeuninstaller.bin` in the directory *`installDir/product`*`/_uninst_loc`.

     - *Verity Single-language European locales:* launch the file `localeuninstaller.bin` in the directory *`installDir/product`*`/_uninst_eur`.

   The uninstaller Welcome screen appears.

2. Click **Next**.

The uninstaller Summary screen appears, listing the locales and languages to be removed from the specified directory.



3. To continue, click **Uninstall**.
4. On the Confirmation screen that appears after the files have been removed, click **Finish**.

Note On UNIX, if you have installed multiple times, there will be multiple uninstall directories (for example, `_uninst_loc1`, `_uninst_loc2`, and so on). Be sure to run the uninstaller that is in the latest uninstall directory.

2

Language Concepts

By its nature, textual information is language-specific. The words, sentences, paragraphs, and documents that make up a body of knowledge are expressed only within the context of one or more human languages. The fundamental building blocks for those expressions—characters and symbols—are numerous and highly specific to individual writing systems.

A useful information-retrieval technology must be able to process information in a large variety of writing systems, and it must be able to extract meaningful units of information (words, phrases, concepts, and so on) from many different languages.

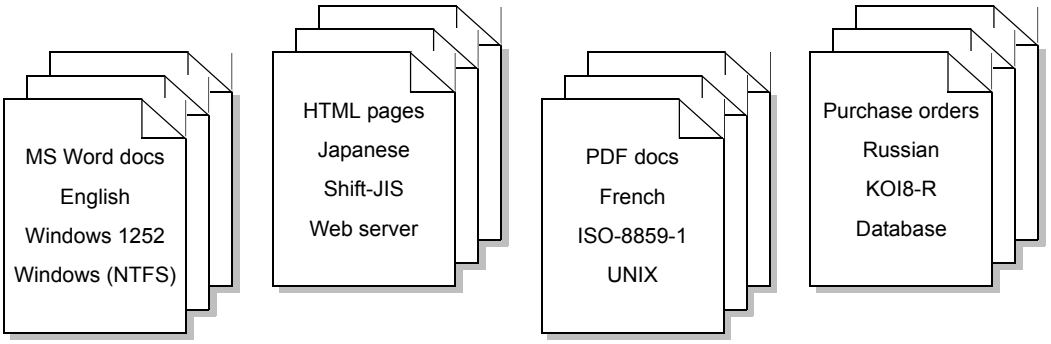
This chapter gives an overview of Verity's software architecture to illustrate the language-related issues that it addresses. The chapter includes the following sections:

Language and Encoding in Documents

Text information is stored on the world's computer systems in a great variety of languages and formats. The different languages, the different (and often proprietary) storage formats, and the different computer platforms involved present challenges for extracting searchable information.

Figure 2-1 shows examples of the kinds of document characteristics that Verity software needs to work with in order to extract and analyze their content.

Figure 2-1 Document types, languages, character sets, and repositories



The figure shows four kinds of document characteristics:

- **Repository type.** This is the platform or protocol involved in storing and retrieving the information. The examples shown here are file system (Windows or UNIX), Web server (HTTP protocol), and database (ODBC protocol).

  Verity software can access these and other types of repositories.

- **File format.** A given repository type can hold documents or information in many different formats. The examples shown here are Microsoft Word, HTML, PDF, and an example of the use of database tables to store information. (In a database, information is not typically stored in documents, so Verity constructs documents—such as individual purchase orders in this case— from that information.)

  Verity software can read hundreds of different file formats.

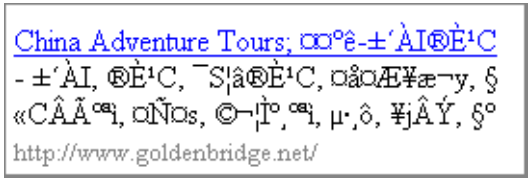
- **Character set**. The character set of a document includes its encoding—the numeric codes used to store the values of the individual text characters. Different languages and different platforms often use different character sets. The characters of a single language might be able to be implemented in several character sets and, conversely, a single character set can sometimes be used to store text in several languages.

Verity software can read document text stored in dozens of different character sets, and it can convert text from one character set into any other character set supported for that language.

- **Language.** Language here refers to the natural language (such as English, Japanese, French, Russian) of the words in the text.

  Verity provides basic (display and storage) support for approximately 100 languages, and it provides linguistically sophisticated support for at least 20 languages.

As an example of the importance of considering character set and character-set conversion when displaying text, consider the following fragment of an HTML document containing mixed Chinese and English text. This is the appearance of the text when the HTML browser's encoding is set to Windows 1252 (typical for English text):



The Chinese characters (top row) are indecipherable. If the browser encoding is now set to Big5 (typical for traditional Chinese text), the Chinese characters display correctly:

A language-aware application can use Verity functionality to track character encoding throughout the process of reading, analyzing, indexing, and displaying text in many different languages. It can convert character encoding whenever necessary to make sure users can read the information presented to them.

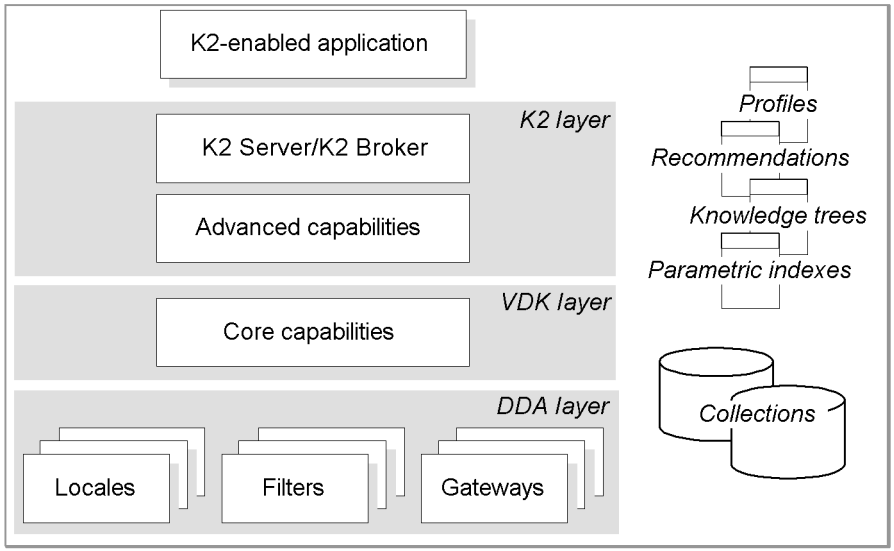
K2 Internationalization Architecture

Verity software is implemented in a modular, layered architecture that allows customers and developers to implement information-management solutions that are highly customized to the needs of their users.

Architecture Overview

As shown in Figure 2-2, Verity consists broadly of three layers of software, atop which sits an application developed by a Verity customer or partner. Each Verity layer is associated with creating specific data structures used by the application, some examples of which are shown in Figure 2-2.

Figure 2-2 Verity software architecture (simplified)



From the top down, the layers have these features:

- **Application layer.** This layer represents the customer's client application, which uses Verity technology to search for, retrieve, analyze, and present information to the user.

  A localized application can provide these capabilities in a language other than English. A highly capable language-aware application can support multiple non-English languages.

- **K2 layer.** This layer represents Verity's broker/server architecture, which provides scalable, distributed access to information, distributed Web-based administration, and a security service. It also creates Verity's most sophisticated information-analysis features, such as parametric indexes, recommendations, profile nets, adaptive ranking, and expert location.

  The client application typically connects to Verity through the broker/server interface. Language issues arise related to display characteristics of retrieved search results, viewed documents, and languages attached to high-level structures (such as topic sets).

- **VDK layer.** This layer represents the core capabilities of Verity's search technology—reading and indexing documents, creating and searching through indexed *collections* of information, and extracting *features* to build document summaries.

  Using linguistic information built into the DDA layer below it, this layer performs the language-specific tasks of converting document text into indexed, searchable units of information. It also interacts with the K2 layer to pass language-specific information, in specific character sets, to the application user.

- **DDA layer.** This is the driver layer, consisting of low-level code modules that interact with repositories, control the processing of their contents, and pass information back to the VDK layer. Three basic kinds of drivers exist at this level:

  - **Gateways.** *Gateways* interact directly with repositories. They read file-level information from the repositories into Verity and prepare it to be processed by document filters. Each type of repository, such as file system, Web server, or database, requires a gateway designed specifically for it.

  - **Filters.** *Document filters* take binary file content from gateways and convert it to field data and pure text *virtual documents*, ready for indexing. Some filters perform language- or character-set identification (Verity can work with source documents in a wide variety of character encodings), others perform character-set conversion and stripping of non-text data from the binary files. Each different file format, such as Microsoft Excel, PDF, or XML, requires a document filter that can read that format to extract document text.

  - **Locales.** Verity *locales* are code modules and data tables that control how the virtual documents created by document filters are to be turned into language-specific word indexes. For each language supported by Verity, there is one or more locales. Verity also implements a special *multilanguage locale*, which by itself supports over 90 languages.

    Locales are the core of Verity's support for internationalization. By installing and configuring locales, an administrator can give a language-aware client application the ability to work in languages other than English.

  Drivers at the DDA level are replaceable, plug-in modules. Verity its partners can help to extend the capabilities of applications by developing new DDA drivers.

Language and Character Set in Indexing

For Verity applications, indexing always occurs within the context of a locale (the *current locale*). This section and the following two sections illustrate how and where language analysis and character-set conversion occur in a language-aware Verity application. The illustrations also highlight the differences between using a single-language locale and the multilanguage locale.

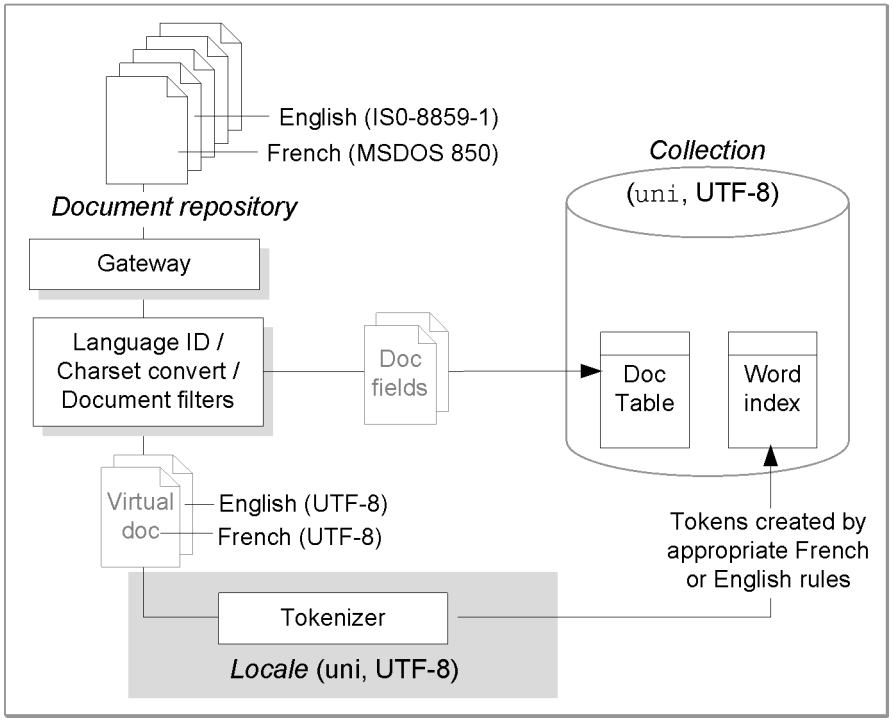
This section summarizes the processes that occur when documents in a repository are indexed into a collection.

Indexing in the Multilanguage Locale (uni)

Figure 2-3 shows the indexing process on the same repository when the current locale is the multilanguage locale. The document repository in this example contains both English documents written in the ISO-8859-1 character set and French documents written in the MSDOS 850 character set. A Verity spidering application (such as `vspider` or K2 Spider) or direct-indexing application (such as `mkvdk`) is performing the indexing.

1. The gateway reads in the file's contents.
2. The file is filtered:
   - The document filter opens the document.
   - The language-detection filter assigns a language to the document.
   - An auto-detection filter determines the character set of the content and converts it to the internal character set (UTF-8) of the current locale (`uni`).
   - The document filter creates the virtual document and writes descriptive information (including its language—`en` or `fr` in this case) to the collection's document table.
3. The document filter passes the virtual document to the Verity tokenizer, which uses language-specific rules (French rules for the French document, English rules for the English document) for tokenizing. Each token in the virtual document, along with index information noting all of its occurrences in the document, is added to the collection's word index.
4. If word stems are being indexed, both the English word stems and the French word stems—each marked for the language they apply to—are placed in the word index at the same time as the other tokens. See "Stemming" on page 47 for details of how this process occurs for the multilanguage locale.

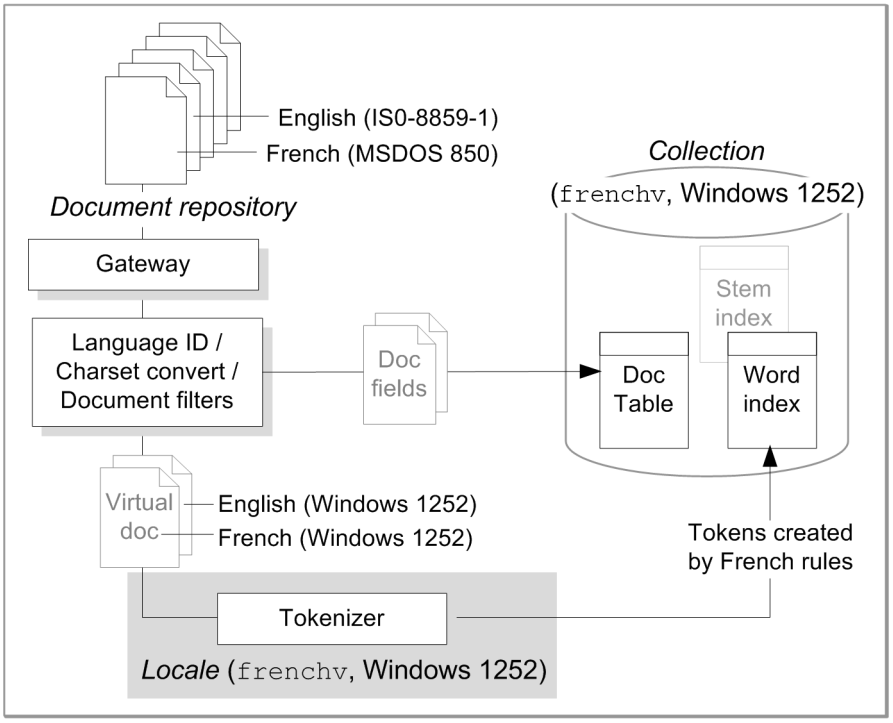
Figure 2-3 Language and encoding during indexing (multilanguage locale)



Indexing in Single-Language Locales

Figure 2-4 shows the indexing process when the current locale is a single-language locale, in this case `frenchv`. The process is similar to that with the multilanguage locale, except that the language information that is captured does not affect the indexing process.

Figure 2-4 Language and encoding during indexing (single-language locale)



1. The gateway for this kind of repository opens the file containing the French or English document and reads in its contents.
2. The file is filtered, or initially processed:
   - A document filter for the type of document that has been read in opens the document.
   - A language-identification filter (see "Language Identification" on page 44) assigns a language—`en` or `fr` in this case—to the document.
   - An auto-detection module (see "Character-Set Detection" on page 44) determines the character set of the content and converts it to the internal character set (Windows 1252) of the current locale (`frenchv`).
   - The document filter extracts the text from the document and creates a pure-text virtual document to be used for indexing. The filter also generates or extracts metadata (descriptive information) from the document—including its language, as determined earlier in this step—and writes it into fields in the collection's *document table*.

Both the virtual document and the field information are in the current locale's internal character set (Windows 1252 in this case).

3. The document filter passes the virtual document to the Verity tokenizer, which uses locale-specific rules (in this case `frenchv` rules) to break the document into words or other tokens for indexing. Each token in the virtual document, along with index information noting all of its occurrences in the document, is added to the collection's word index.

   The use of French rules for indexing in this example applies to every processed document, regardless of whether its actual language is French. With a single-language locale, all documents in a collection are indexed according to that language's rules.

4. If word stems are being indexed (see "Stemming" on page 47), a stem index (stemdex) is created in the collection during a second indexing pass, using, in this example, `frenchv` stemming rules.

Language and Character Set in Searching

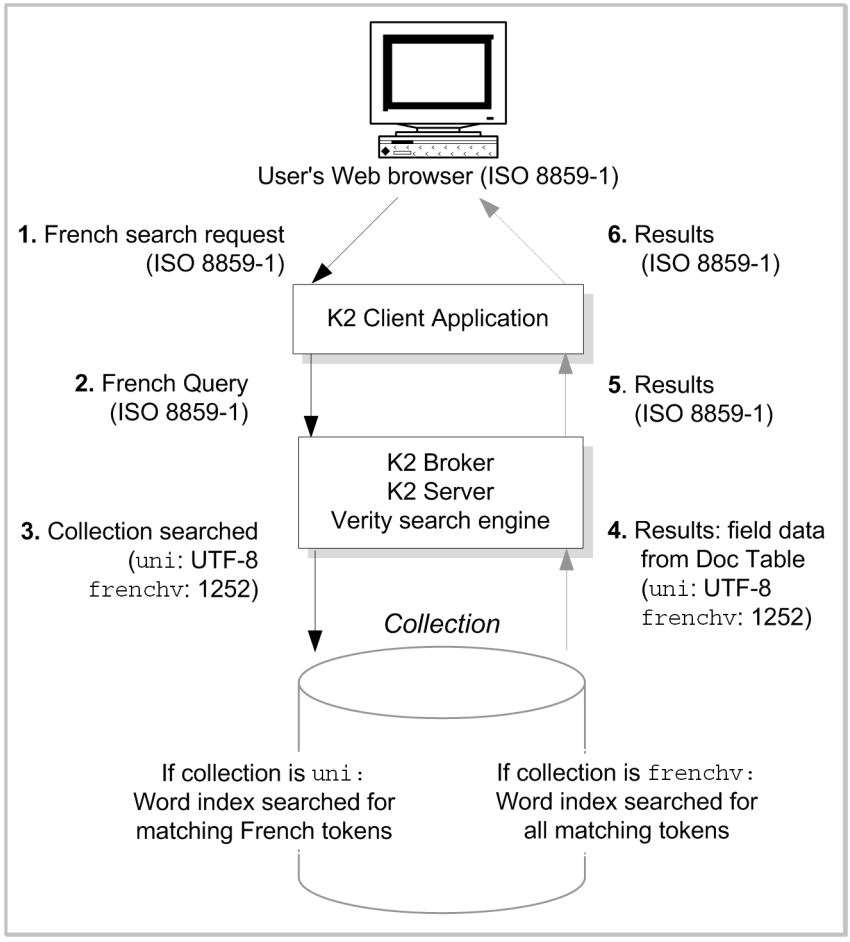
This section summarizes the locale- and character set-related processes that occur when application users search for documents in a collection.

Note that searching and display of search results involve the user's application and the collection; the document repository itself is not accessed.

Searching in the Multilanguage Locale (uni)

Figure 2-5 shows what happens during a search of the example multilanguage collection indexed as described in "Indexing in the Multilanguage Locale (uni)" on page 36. In this example, the user accesses the Verity application through a Web browser and keyboard whose character encodings are currently set to the ISO-8859-1 character set.

Figure 2-5 Language and encoding during searching



1. The user enters a search term in an application page (in this example, an HTML page in a browser). The browser sends the search term to the client application.
2. The client application constructs a VQL query string from the search term:
   - If a language-independent search is to be performed, the application can apply the VQL operator `<WORD>` to the string to ensure that all occurrences of the term in any language should be included in the results.
   - If a stemmed search is to be performed, the application applies the VQL modifier `<lang/fr>` to the string to specify that only occurrences of words with French stems should be included in the results.

(If the default language for searching has been set to `fr`, this operator is not required; see "Resetting the Default Session Language" on page 117.)

Note When a multilanguage collection contains Chinese and Japanese documents, searching with the `<lang/id>` modifier can return hits from documents written in both languages.

The application sends the string to the K2 Broker or K2 Server, specifying that it wants the results to be returned in ISO 8859-1.

3. The K2 Server passes the string to the Verity search engine, which converts the character encoding to UTF-8, then searches the collection's word index for all occurrences of the search term. If a stemmed search is being performed, the word index is searched for only French stems.
4. The Verity search engine converts the search results to ISO 8859-1 and returns them to the K2 Server.
5. The K2 Server sends the results to the client application.
6. The client application sends the search results to the user's browser.

Note A client application using the multilanguage locale and simultaneously serving multiple users in multiple languages might specify its own internal character set (UTF-8), and either require all users to work in that character set or else perform its own character-set conversion between the user's character set and UTF-8.

Searching in Single-Language Locales

Figure 2-5 on page 40 also shows what happens during a search of the example `frenchv` collection indexed as described in "Indexing in Single-Language Locales" on page 37. The process is similar to searching a multilanguage collection, except that the language of the search is not specified.

1. The user enters a search term in the browser page, which causes a search request to be sent to the client application. The user's submitted string is in ISO 8859-1 character encoding.
2. The client application constructs a VQL query string from the user's search term. The application sends the string to the K2 Broker or K2 Server and specifies that it wants the results returned in ISO 8859-1.
3. The K2 Server passes the string to the Verity search engine, which converts the character encoding to that of the current locale (`frenchv`) and searches the collection's

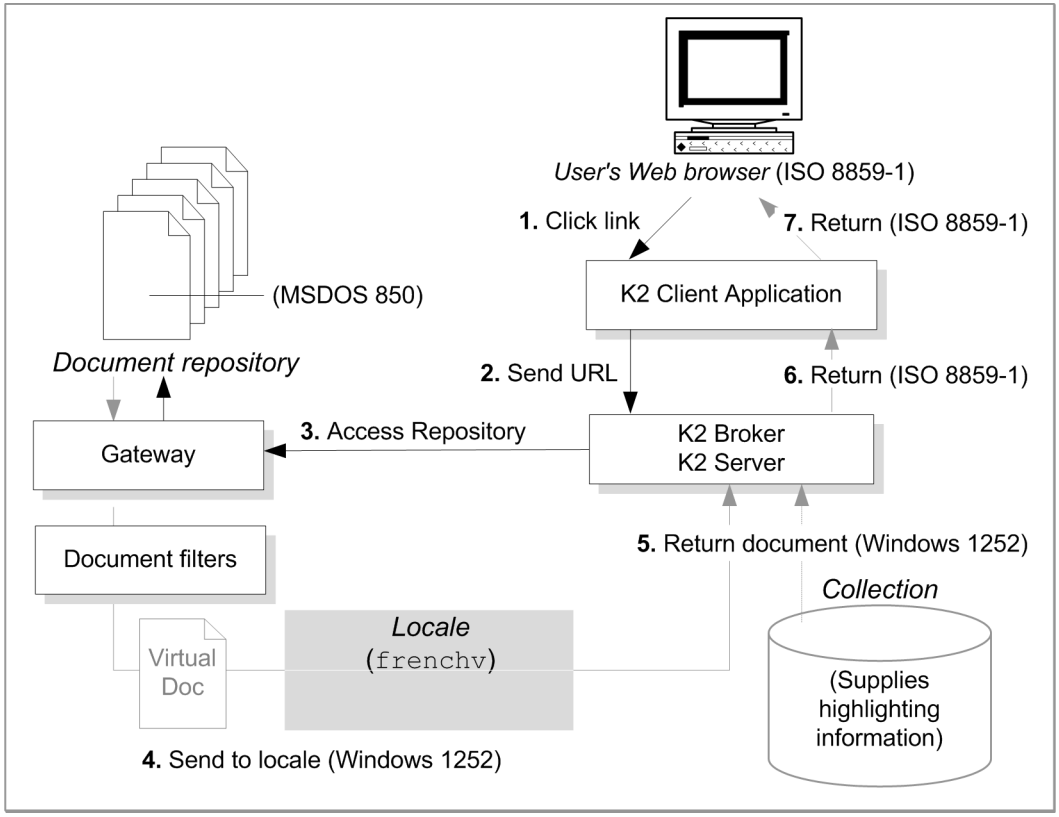
word index for all occurrences of the search term. If a stemmed search is being performed, the stem index is searched. In either case, all indexed documents, whether French or not, are included in the search.

4. Search results—consisting only of field information from the collection's document table—are returned to the search engine. These results are in the locale's internal character set (Windows 1252).
5. The search engine converts the results to client's internal character set (ISO 8859-1), and the K2 Server returns the results to the client application.
6. The client application sends the search results to the user's browser. (The client might specify ISO 8859-1 encoding in the HTML header sent to the browser, to ensure that the user's browser would switch to ISO 8859-1 to display the results.)

Language and Character Set in Document Retrieval

This section summarizes the processes that occur when an application user requests to view a document referenced in search results.

Figure 2-6 Language and encoding during document retrieval



1. The user clicks a document link in the search results displayed on the browser page. The browser sends a URL to the client application.
2. The client application forwards the URL to the K2 Broker or K2 Server.
3. The K2 Server accesses the document repository through the gateway, which opens the document in its own character encoding (MSDOS 850 in this example).
4. The gateway passes the opened file to the document filters, which convert the document to the locale's character set, create a virtual document and pass it to the locale.
5. The locale obtains word-location information from the collection (if it is using static highlighting to highlight the search term in the document), highlights the search terms in the document, and passes the document to the K2 Server.
6. The K2 Server converts the virtual document to the client's character set (ISO-8859-1) and passes the document to the client application.

7. The client application displays the returned document to the user.

Character-Set Detection

Some documents in a repository might carry specific information (such as in the `encoding` attribute of the `<?xml>` declaration in an XML document) that identifies the character set they are written in. Many documents, however, do not, and it is up to Verity to determine the actual character set of each incoming document.

In some cases, based on its configuration settings, the gateway might have prior knowledge of the character-set encodings of the documents it reads. If not, the Verity auto-detection process analyzes the incoming document data to determine its character set.

It is the responsibility of the gateway or the document filters—whichever determines the character set of an incoming document—to convert that document, if necessary, to the internal character set of the current locale. Verity's character-set conversion capability, like its auto-detection capability, is based on KeyView technology, which can detect and read over 250 common file formats.

Language Identification

Every collection is created in the context of a single locale. Most locales apply to a single language only. For those locales, documents in a repository that are not in the locale's language are usually not of interest for indexing. In this situation, even though auto-detection still attempts to identify the language of every file, all documents that are in a character set supported by the current locale are passed through to the locale for tokenization.

The multilanguage locale is a special case. It supports many languages, and it applies language-specific rules when tokenizing the text of a document. Therefore, it must be able to identify the languages of documents read in through the gateway.

The Verity *language-identification filter* is used with all locales to detect the language of incoming documents. The filter makes use of the document's character set (which in some cases is enough to identify the language), the presence of specific accented characters, and other language features to make the identification. The filter stores the document's language assignment as a two-character language code (see "Supported Language Codes" on page 135) in the `VLANG` field of the collection's document table.

Verity's tokenizer assumes in general that each document consists of text in a single language. Language identification assigns the language to a document based on a portion (by default, the first 2K bytes) of content. Portions of mixed-language documents that are not in the main language are processed as if they were.

Language-Related Indexing Features

Verity locales exist to provide support for language-aware search. Each locale provides rules, settings, tables of information, and functions that facilitate the construction of collection indexes that take into account the word structure, spelling, and parts of speech in that locale's language.

Sorting Order

For faster case-insensitive and accent-insensitive search, and for efficient search of related spellings, the word index in a collection needs to be sorted in an order that is specific to the language's set of characters. That sorting order can also be used to present search results to the user.

Each locale maintains a table of characters and their variants, with entries placed in the sorting order for that language. Typically, the sorting order groups all variants of a character together, like this:

A a À à Á á Â â B b C c Ç ç D d ...

With this ordering, all accented or capitalized variants of a word are adjacent to each other in the word index, making accent-insensitive and case-insensitive searching efficient.

Tokenization and Word Delimiters

In general, Verity collections store all of a document's individual words as the elements of the word index it creates from the document. More specifically, the Verity engine generates and stores all the document's *tokens*, which are character strings that occur between *delimiters* (white space or punctuation). This process of extracting tokens from a document is called *tokenization*.

Tokens are thus more than just the natural-language words in the document; they are the document's searchable units. For example, this English sentence

The blue/green used truck costs $2000 - $5000 more (plus taxes).

might be converted to

$2000
$5000
blue/green
costs
more
(plus
taxes)
The
truck
used

because, in this case, the blank space, period, and hyphen are considered tokenization delimiters but the forward slash, dollar sign, and parentheses are not. This is the default behavior for older versions of some locales, such as `english`.

The set of delimiters that controls tokenization is highly locale-dependent and, for most locales, is now customizable by the Verity administrator. For the example just given, if the administrator chooses to enable *simple tokens* behavior, which redefines nearly all symbols as delimiters, the following tokens would appear in the word index:

$
2000
5000
blue
costs
green
more
plus
taxes
The
truck
used

In this case, **blue**, **green**, **plus**, and **taxes** are now searchable words in the document.

The advantage of having more delimiters (and thus shorter tokens) is that more hits are returned from searches. Simple tokens is the default behavior for the Verity multilanguage locale, the Advanced European locales, the legacy European locales, and for Latin characters in the Asian locales.

In some situations, however, longer, more specific tokens may be more useful—such as in automatic classification, in which longer words (such as **blue/green** in this example) might make better category names (than just **blue** or **green**). For that reason, the simple-tokens behavior can be disabled.

See "Tokenization" on page 78 for instructions on specifying simple tokens and redefining tokenization delimiters.

Stemming

Stemming is a process by which Verity further breaks down a word by extracting its *word stem*, or main part, stripped of prefixes or suffixes. Indexing the word stems in a document allows for *stemmed search*—a search that finds all the words that share the supplied stem.

For example, suppose a document in English contains the words **houses**, **housed**, and **housing**. A regular search for the term **house** would find nothing. But a stemmed search would find all three words, because **house** is the stem for all of them.

(Verity locales use *inflectional* stemming, meaning that only stems of the same part of speech as the word being stemmed are extracted. In the above example, all are verbs.)

In the used truck example from the previous section, the stems **use** and **tax** would also be indexed, so that users searching for those terms would find the information about the used truck.

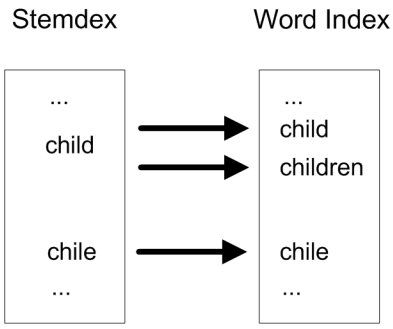
Note Verity also uses word stems when it automatically constructs higher-level indexing structures such as document summaries and clusters; see the *Verity Collection Reference* and the *Verity Intelligent Classification Guide* for more information.

Note Hyphenated words are not searched by any stemming operators. For example, a search query of **known** will not match **well-known**. However, a query of ***known** will match **well-known**. For more information, see "Tokenization and Word Delimiters" on page 45.

Verity locales use two different methods for stemming: one for single-language locales, and another for the multilanguage locale.

Stemming for Single-Language Locales

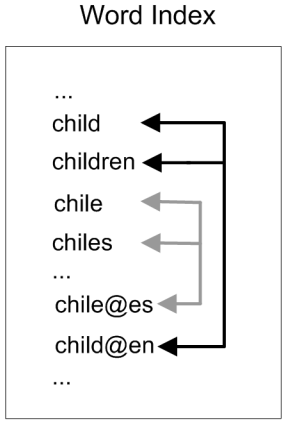
With single-language locales, stemming is performed as a separate process after indexing, and the word stems reside in a separate stem index (stemdex) that Verity creates inside the collection (see also Figure 2-4):



An entry in the stem index notes the locations of all words in the word index that share that stem. The word index in turn has the locations of those words in the document.

Stemming for the Multilanguage Locale

With the multilanguage locale, stemming is performed during indexing. Stems are marked according to language and are related to the regular tokens in the word index (see also Figure 2-3.):



In this example, each English word that is a variation of **child** is referenced by the stem **child@en**. Likewise, the word **chiles** (from a Spanish document) has **chile@es** as its stem. English searches for **child** will return occurrences for all words referenced by **child@en**; Spanish searches for **chile** will return all occurrences of **chile** and **chiles**.

This method can in some cases improve stemming quality, since the context (part of speech) of each word to stem is known. For example, the stem of the past-tense verb **saw** is **see**, but for the noun **saw**, it is just **saw**.

Words that are common to more than one language can have more than one word stem. For example, the word **llamas** in a word index could have the stems **llama@en** and **llama@es**, in which case both English and Spanish stemmed searches for **llama** would return all occurrences of **llamas**. However, a Spanish search for **llama** would also return occurrences of **llamadas**, whereas an English search would not.

Stemming is highly language-dependent, and each locale implements stemming according to its own rules. In the current locale architecture, stemming functionality is not customizable, although it can be enabled or disabled.

Normalization

Some locales support *normalization*, an indexing feature in which a single version of a character is used when alternate versions exist, and a single spelling is used for a word that has alternate spellings. Users searching a normalized collection for a word will then find all words with either the common spelling or any of the alternate spellings.

For example, in the Japanese language, both Katakana (phonetic) characters and ASCII characters occur in half-width and full-width versions, with different character codes. In the Verity Japanese locale (`japanb`) and in the multilanguage locale, the half-width versions are normalized to their full-width equivalents. A person searching for full-width Katakana word (ニュース, for example) will find all occurrences of both the full-width and half-width (ﾆｭｰｽ) version. As another example of Japanese normalization, Okurigana (Voice-marked Kanji) is indexed as non-marked Kanji.

Normalization applies to the tokens in the collection index itself, not to the original source documents. When viewing the documents through a Verity client, the user sees the actual spellings and the actual versions of the characters that occur in the source.

Decomposition of Compound Words

Some languages (notably German) include the concept of *compound words*, words created by the concatenation of several independent words in certain grammatical contexts. *Decomposition* is the process by which Verity breaks compound words into their constituent tokens.

For example, the German word for taxi driver is **taxfahrer**. During indexing, the word is decomposed into the subwords **taxi** and **fahrer**, and each subword is indexed separately.

A locale that supports compound words creates independent tokens for each compound word and for all subwords of the compound word. In the word index, the subwords are marked as having the same positions in the document as the compound word. Therefore, searching for either the compound word or any of its subwords will produce the same matches.

Decomposition is somewhat similar to stemming, in that it extracts smaller units from tokens. However, a compound word is considered a collection of words, whereas the words that share a stem are considered variations of the same single root word.

For some Asian languages, Verity supports user customization of word decomposition. For those locales, you can create a user dictionary that contains terms (such as proper names or industry-specific terms) that should be decomposed in a non-default manner or not decomposed at all. See "Customizing Word Decomposition" for details.

Japanese uses compound words that can be repeatedly decomposed. For example, the word 東京三菱銀行 (Tokyo Mitsubishi Bank) can be decomposed into 東京 + 三菱銀行 (Tokyo + Mitsubishi Bank), or more completely decomposed into 東京 + 三菱 + 銀行 (Tokyo + Mitsubishi + Bank). Each division represents a word. The compound word is a noun phrase.

Part-of-Speech Identification

Some Verity locales support part-of-speech identification during indexing. When it is used, each indexed token is analyzed to determine whether it is a noun, verb, adjective, number, and so on.

An extension of part-of-speech detection is *noun-phrase* extraction. Automatic detection of noun phrases is available for some locales, and high-level Verity tools use that capability to automatically extract document features and construct document summaries or clusters from a collection.

Part-of-speech information and noun-phrase extraction are used by Verity software to better support high-level constructions such as feature extraction, document summaries, document clusters, and automatic classification.

For some locales, noun-phrase extraction can be disabled for improved performance, if desired.

Number Handling

Some languages use traditional script for numbers as well as the common Latin versions. For example, Chinese, Japanese and Korean use Han script numbers as well as Latin numbers. The number nineteen can be written in Han script in several different ways, or as the Latin **19**.

For those locales that support number handling, performing a stem search with either a script number or its equivalent Latin number produces the same results.

In some languages, script numbers may also be used as non-numbers. For example, in Japanese, the word **Ichinomiya** is a place name that (when written in Japanese script) contains the Han number 1, but in this case the 1 does not represent a number value. A Verity locale converts a script number to Latin only if all characters in the word represent numbers (or day and month characters, in the case of date strings).

Language-Related Search Features

Verity locales provide several features to help users tailor their searches to provide more specific or more complete results, based on the specific characteristics of the language of the collection being searched.

Locale and Language in Search Queries

The Verity Query Language (VQL) supports the concept of language-specific searching. When searching a collection created with the multilanguage locale, a user can specify that only occurrences of the search term in documents of a particular language be returned.

The language of the user's searches can be specified as a default on a per-session basis, or it can be applied to an individual query string. VQL defines the `<lang/id>` modifier, which can be used in a query string to restrict a stemmed search to documents whose language is specified by a standard (usually two-letter) code *id*.

For example, this query

```
<lang/es>aceptar
```

searches for all occurrences of the Spanish word **aceptar**. If this were a stemmed search, it could find all occurrences of Spanish words whose root is **aceptar**, such as **acepté**, **aceptó**, and **aceptaron**. The same search could also find the word **aceptar** if it was a valid word in other languages.

You can use multiple instances of the `<lang/id>` modifier in a single query. This allows you to search for terms in more than one language without having to use multiple queries.

For example, this query

```
<lang/en> cat <and> <lang/fr> (chien, chat)
```

searches for the English word **cat** and the French words **chien** and **chat**. Note how parentheses are used to specify multiple words.

Use of language in search queries is described further in Chapter 5. Supported language codes are listed in "Supported Language Codes" on page 135.

Case-Insensitive Search

With case-insensitive search, a search term of **a** returns occurrences of both **a** and **A**. All locales by default specify case-insensitive searches. The Verity administrator can later reconfigure Verity to make searches case-sensitive, if desired. Also, use of the VQL operator `<CASE>` on a search term forces the search to be case-sensitive, even if case-insensitivity is enabled.

The Verity *auto-case* capability is a search convention in which search terms that are all one case (such as **next** or **NEXT**) are searched for case-insensitively, whereas mixed-case search terms (such as **Next** or **neXT**) are searched for case-sensitively. In this example, if auto-case were enabled, an occurrence of **NeXT** would be found by either of the first two search terms but not by either of the second two terms. By default, auto-case is disabled.

Accent-Insensitive Search

With accent-insensitive search, a search term of **a** might return **a**, **à**, **á**, and **â**. Accent-insensitive search is in most cases preferable to accent-*sensitive* search, in which each accented variation is treated as a separate character for searching. However, note these implications of using accent-insensitivity:

- Automatically extracted feature names (see "Part-of-Speech Identification" on page 50) will contain only unaccented versions of their characters.

- Collections created with an earlier, accent-sensitive version of a locale my need to be re-indexed to retain the same search behavior.

Accent-insensitive search is supported for non-Asian languages in the multilanguage locale, for the advanced European locales, and for the legacy European locales. (Accented text does not occur in the Asian locales.) Accent-sensitive search is not supported.

Symbol Search

Normally, punctuation, white space, and other non-alphanumeric characters are not searchable. In some Verity single-byte locales and in the multilanguage locale, however, you can configure the locale so that nearly any of the defined token delimiters are searchable.

For example, without symbol search, the phrase **©Verity Inc. 2003** would be indexed as

2003
Inc
Verity

(assuming that both © and **.** are specified as word delimiters), and a search for **©Verity** would produce no results. If © is made searchable, however, the word index would have these entries:

©
2003
Inc
Verity

and searches for **©**, **Verity**, or **©Verity** (as a phrase) would be successful.

Synonym Search

The Verity administrator can create a *thesaurus*, or dictionary of synonyms, to use with the collections created for a given locale. When the user conducts a synonym search, occurrences of the search word (for example, **run**) as well as any of its synonyms (such as **race**, **rush**, **hurry**, **bolt**, **dash**, **hasten**) are returned.

In VQL, you specify a synonym search with the `<THESAURUS>` operator.

For instructions on how to create or modify a thesaurus, see the *Verity Query Language and Topic Guide*.

Soundex Search

For those locales that support it, Verity allows the user to perform a *Soundex search*. In this type of search, occurrences of the search word and also of any similar-sounding words are returned. For example, searching for the name **Jean** would return occurrences of it plus any similar-sounding but differently spelled names, such as **Joan** or **Jane**.

In VQL, you specify a Soundex search with the `<SOUNDEX>` operator.

Soundex was originally developed for indexing proper names for census purposes. Currently, Verity supports Soundex search for the multilanguage locale (English language only), the `englishv` locale, and the `englishx` legacy locale.

Typo Search

For some locales, Verity supports *typo search*, a kind of "fuzzy search" that corrects for minor misspellings in the search query. In a typo search, occurrences of the search word and any words close to it in spelling are returned.

For example, if the user's search term is **juvinile**, the typo search facility might return all occurrences of **juvenile**. In addition, the client application might display a suggestion to the user, such as

Did you mean to search for juvenile?

The client application can configure the precision of the typo search by specifying how closely the spelling of the returned items must match the search term.

In VQL, you specify a typo search with the `<TYPO>` operator.

Typo search is not strictly related to language features, except that some locales support it and others do not.

Stop Words

A *stop-word* list is a list of terms to ignore in searching or in indexing. Typically, stop-word lists include very short and very common words (such as **a**, **an**, and **the** in English), but they also might include longer words such as long number strings, or possibly words that are too common to be useful as search targets (such as the term **Internet** in an indexed collection consisting entirely of documents related to the Internet).

The primary reason for using a stop-word list is that it can increase search speed and decrease the size (storage requirement) for an index. Verity provides support for four different stop-word lists, each with a different purpose or scope:

- `style.stp`. This stop-word file lists words that should not be indexed. Words on this list do not make it into a collection's word index, and therefore are not searchable. The `style.stp file is` collection-specific and supports the following:
  - regular expressions
  - case-sensitive matching (not case-insensitive)
  - only single-byte character sets

  Putting common words in this list can impair searching for phrases. For example, if the word **the** is on this list, searching for **Attack of the Clones** will return no results, even for a collection devoted to recent science fiction movies—unless **the** is also in the stop-word list that is applied to the search query itself (see `qp_inet.stp` and `vdk30.stp`, below).

  Instructions for using `style.stp` are in the index-tuning chapter of the *Verity Collection Reference*.

- `style.fxs`. This collection-specific stop word list is used by the feature-extraction process during indexing. Feature extraction is the automatic process of generating keywords and phrases that characterize a document, for the purpose of summarizing it or clustering it with other similar documents.

  This file augments `vdk30.stp`, but does not support regular expressions. It supports multibyte character sets and case-insensitive matching.

  Words listed in `style.fxs` might exist in the collection index, but they nevertheless are not used in generating keywords and phrases that constitute the document features. Those words might include proper names, single characters, and common short words.

  Instructions for using `style.fxs` are in the chapter on index tuning in the *Verity Collection Reference*.

- `qp_inet.stp`. This default stop word list for the Verity Internet-style query parser is locale-specific, and uses the same format as `vdk30.stp`. It contains words that the query parser will strip from query terms before conducting a search.

  Words listed in `qp_inet.stp` might include short words—articles, prepositions, and so on— to allow the parser to convert a natural-language question, such as

  Where can I buy sourdough bread in San Francisco?

  Into a search for its core terms:

  buy sourdough bread San Francisco

  The Internet-style query parser is described in the *Verity Query Language and Topic Guide*. It uses a stopword file (in *`locale_name`*`/qp_inet.stp`) to strip out very

common words in the query. The word **it** is in `english/qp_inet.stp`, so it is removed from the list of query terms in the English locale. If all the terms in the original query are removed (that is, if they are all listed in `qp_inet.stp`), the Internet-style query parser generates a dummy query (`<Word> #nil#`) that is intended to match zero documents.

You can edit the `qp_inet.stp` file (for example, to remove the word **it**).

- `vdk30.stp`. This locale-specific stop-word file is used, along with style.fxs, for feature extraction at indexing time. It is also used by the Verity Query By Example (QBE) parser to convert natural-language phrases into query terms, in a similar manner to the Internet-style query parser. In the multilanguage locale, this stop-word file only contains stop words common to all supported languages. You can specify locale-specific stopwords in the `stopword.`*`ID`* file. For more information, see "Creating a Stop-Word File" on page 105.

  The `vdk30.stp` file does not support regular expressions. It does support multibyte character sets and case-insensitive matching.

  For a multilanguage collection, you can define a language-specific stopword file in `stopword.`*`ID`*, located at *`productDir`*`/common/uni`, where *`ID`* is the language ID.

Two of these stop-word files, `style.stp` and `style.fxs`, are collection-specific; you need to set up different versions of them each time you create a collection in a different language. The other two files, `qp_inet.stp` and `vdk30.stp`, are locale-specific. Each locale has its own default implementation of `vdk30.stp`, thus providing language-sensitive stop words for QBE queries and feature extraction in any language.

Instructions for creating or customizing `vdk30.stp` are in Chapter 1 of this book. The QBE parser is described in the *Verity Query Language and Topic Guide.*

Also to consider is the VDK flag `VdkTokenWordInfoFlag_Noise`, used on VDK `WordInfo` tokens that tokenizers in locale drivers can set to indicate that the following word is a stop word. Such noise words will be ignored by the indexer, but will be included by the feature extractor unless specified in `vdk30.stp` or `style.fxs`. For more information, see the *Verity Developer's Kit Programming Reference*.

Limitations in Handling Source Documents

Certain language-related issues in some types of source documents either cannot be handled by Verity, or must be handled as special cases.

- **HTML/XML documents.** Some HTML and XML files include the language-specification attribute `lang` in some tags. Verity, however, ignores that specification, if it is present (unless it occurs in the `<HTML>` tag), and handles language assignment in this way:
  - *Single-language locales.* The file is indexed according to the language rules of the current locale.
  - *Multilanguage locale.* The language-identification filter assigns a language to the document based on its text content. The file is then indexed according to the rules of its assigned language.
- **Archive documents.** Verity can read and process compressed document archives, such as Zip files. Documents within the archives can be extracted and indexed.
  - *Single-language locales.* Documents in the archive are indexed according to the language rules of the current locale, regardless of the document's language.
  - *Multilanguage locale.* Documents in the archive are indexed according to their own languages. The language assigned to the archive file itself (as shown in the `VLANG` field in the collection's document table), however, is the language of the first indexable file encountered during processing.
- **Database-based documents**. Verity can assemble and then index virtual documents that it constructs from database-table columns. In some cases, the language of one column might be different from that of another. In single-language locales, the entire document is indexed according to the language rules of the current locale, so one or more columns could contain meaningless data.
- **PDF documents.** PDF documents can exist in many different character encodings. Verity includes two different document filters that convert PDF content differently, depending on the current locale.
  - *Adobe PDF filter.* The Verity PDF filter converts Latin 1-based PDF text to the Windows 1252 character set. Locale-based tokenization is not used.
  - *KeyView filter.* The Verity KeyView filter converts PDF text to the internal character set of a locale. The filter can be used with any locale if locale-based tokenization is desired.
- **Language identification.** During indexing to create a collection in the `uni` locale, when the language-detection filter cannot recognize the language of a document, it assigns the language value `un` to the document.

3

Verity Locales

This chapter describes the features of Verity locales, the software modules that give applications based on Verity technology the ability to work in many languages.

This chapter includes the following sections:

Locale Basics

All Verity locales share the characteristics described here.

Installed Location

An installed locale module is a set of data files and one or more executable library files. The data files are in the *locale directory*, at

productDir\common*locale_name*

where

- *productDir* is the path to the directory containing the component of Verity that has been installed (for example, `usr/verity//k2_6/k2` for K2 Services on UNIX, or `C:\Verity\Intelligent Classifier` for VIC on Windows).
- *locale_name* is the name of the locale (for example, `japanb`).

In a K2 Services installation, the library files are in the directory *os_platform*`\bin`,

where *os_platform* is the pathname of the operating system-specific directory (for example, `C:\verity\k2_6\k2\_nti40` for Windows) that holds executable Verity files.

The locale driver has a name of the form `loc_`*DriverName*`.so`, `loc_`*DriverName*`.sl`, or `loc_`*DriverName*`.a` on UNIX, `loc_`*DriverName*`.dll` on Windows.

Locale Definition File and Locale Configuration File

The file `loc00.lng`, in the directory *productDir*`\common\`*locale_name*, controls several aspects of locale behavior. The Verity administrator can edit that file to customize the locale.

For the multilanguage locale, the file `uni.cfg`, in the directory *productDir*`\common\uni`, controls most aspects of locale behavior. The Verity administrator can edit that file, as well as `loc00.lng`, to customize the locale.

See "Configuring Locales" on page 77 for details on editing `loc00.lng` and `uni.cfg`.

Note In previous Verity releases, the standard file for controlling tokenization behavior was `style.lex`, which is not associated with any particular locale and cannot handle tokenization of anything but 7-bit ASCII characters. In place of `style.lex`, you should use each locale's `loc00.lng` file (and `uni.cfg`, if appropriate) to control tokenization and other language-related features.

Internal Character Set and Supported Character Sets

Every locale module has a single internal character set. All collection indexes and all associated files (such as BIFs and style files) processed by the locale are stored in that character set. The internal character set for a locale is specified in the locale's `loc00.lng` file and cannot be changed.

All locales support other character sets in addition to their internal character set. Support for another character set means that collection data, query strings, and search results in that locale can be displayed or printed using one of the character sets specified as supported for that locale. Verity performs the necessary character conversion in such cases.

The internal character sets and the additional supported character sets for all locales are listed in "Verity Locales and Character Sets" on page 128.

Default Locales and the Session Locale

Every K2 or VDK application or command-line tool must establish a VDK session at run time, before accessing collection data or making API calls. Each VDK session includes a defined internal *session locale*—the locale that Verity applications and tools assume to be the locale of collections they access.

The session locale can be specified explicitly or it can be either of two default session locales:

1. If the application or tool explicitly specifies a locale when it establishes the session, that locale is the session locale.
2. If the application or tool does not specify a locale, Verity uses the *default installation locale*, if it exists, as the session locale.

   The default installation locale can be specified in the Verity configuration file (`verity.cfg`). Initially, it is not defined; however, you can define it by following the instructions in "Redefining the Default Session Locale" on page 102.
3. If the default installation locale is not defined, Verity uses the *system default locale* as the session locale. The system default locale is `uni`, except as noted in the next paragraph.

IMPORTANT If your installation is licensed for the `englishx` locale (see "Legacy European Locales"), and if you have installed that locale, it becomes the system default locale in place of `uni`.

When executing a command-line tool that uses the `-locale` option, or when making a function call that takes a `locale` or `internalLocaleDriver` parameter, note that if you do not explicitly pass a locale value, that is equivalent to specifying the default installation locale.

Default Session Language

When the session locale is `uni` (the multilanguage locale), VDK allows for the concept of a *default session language*. The default session language is the language to apply to search queries that do not explicitly specify a language (through the VQL `<lang/id>` modifier).

- For a VDK session, you can define the default session language using the `uni/id` form for the `-locale` option on command-line tools such as `rcvdk`. If you do not specify a default session language for the `uni` locale, the default session language is defined as `un` (unknown).
- When you are using a K2 tool (such as `rck2`) to search, you cannot redefine the default session language on the command line. Each collection's default session language is specified when it is registered with a K2 Server (for example, through the K2 Dashboard) and `rck2` cannot override that specification. If you need to change the default language for a registered collection, do this:

  a. Take the collection offline.

  b. Change its default session language, in either of these ways:

  - K2 Dashboard: specify the new default session language from the **VDK** tab on the collection's Edit Properties page.

  - `rcadmin`: Use the `indexvdkset` command to specify the new language (using the `uni/id` form for the `-locale` option).

  c. Bring the collection back online, then use `rck2` without specifying a language on the command line.

Installation and Licensing

In general, locales need to be installed before they can be used. Installing one or more locales means running the Locales installation program. When you run the installer, it prompts you to supply a license key for the locales and languages to be installed.

The Verity administrator cannot index or manage a collection whose language corresponds to an unlicensed or uninstalled locale or language. For locale installation instructions, see "License Requirements" on page 20. To obtain locale license keys, contact your Verity representative.

Built-In Locales

The following Verity locales are installed automatically when K2, VDK, or VIC is installed:

Verity locale name	Language
`uni`	Multiple languages (advanced support for English only; basic support for other languages)
`englishv`	English (advanced language support);
`english`	English (basic language support only)

These locales do not require a separate installation process. However, note these licensing requirements:

- Use of the `uni` locale, beyond advanced English language capabilities plus simple display and search of other languages, requires a separate locale license and installation.
- The `englishv` locale requires no license and supports only English. It provides advanced English language capabilities equivalent to those of the other Verity single-language locales.
- The `english` locale is a simple, built-in locale that requires no license and provides only limited support for the English language.

Locale Categories

Verity locales can be grouped into the following categories, based on internal character set, language characteristics, and supported indexing features:

- Multilanguage locale (the default locale; default language = English; has advanced support for English and many other languages)
- Advanced European locales (each has advanced support for a single language)
- Asian locales (each has advanced support for a single language)
- `english` (the built-in locale with only basic English-language support)
- Legacy European locales (each has advanced support for a single language; available only with the Single-Language European Locales product)

Verity provides the locales in these categories. Verity partners may make additional locales available. If you have a need for locales other than those described in this chapter, see your Verity representative.

The following sections describe the properties of the locales in each of the categories.

The Multilanguage Locale

The Verity multilanguage locale (`uni`) is a single locale module that supports many languages. It uses the UTF-8 character set, which is based on Unicode 3.2.

Note The default locale for K2 is the multilanguage locale, and the default supported language is English. If you are accessing a `uni` collection and need to use English indexing or searching rules, you do not need to specify the locale or language.

As delivered with Verity K2 and VDK, the multilanguage locale includes advanced linguistic support for the English language, plus basic tokenization support for over 100 languages (see Table A-3 on page 136). With a separate license and installation of the Verity Locales product, the `uni` locale can also provide advanced linguistic support for the languages listed in Table 3-1.

Table 3-1 Languages with advanced support in the multilanguage locale

European	Eastern European/Middle Eastern	Asian
Danish[a]	Arabic	Chinese (simplified)
Dutch	Czech	Chinese (traditional)
English	Greek	Japanese
Finnish	Hungarian	Korean
French	Polish	Thai[b]
German	Russian	
Italian		
Norwegian (Bokmal)[a]		
Portuguese		

Table 3-1 Languages with advanced support in the multilanguage locale (continued)

European	Eastern European/Middle Eastern	Asian
Spanish		
Swedish[a]		

a. Noun-phrase extraction not supported.
b. Language-specific tokenization only; no stemming or part-of speech extraction.

Note Several levels of licensing exist for the multilanguage locale. Use of each of the features described here for each language requires the appropriate licensing level. See "License Requirements" on page 20 for more information.

The multilanguage locale supports the indexing and search features noted in Table 3-2.

Table 3-2 Features of the multilanguage locale

Feature	Support
Character-set detection	Verity's internal auto-detection technology identifies the character set of source documents to be indexed. The locale accepts any source-document character set, converts to UTF-8.
Language identification	Verity's language-detection technology identifies the languages of source documents to be indexed. Language assignment is based partly on character-code ranges in Unicode. (Indexing rules are based on the language assignment.)
Sorting order	Sorting order is based on Unicode character value.
Tokenization	*White-space-separated languages (including Korean):* Simple-tokens behavior, in which nearly all non-alphanumeric characters can be word delimiters, is supported. Individual non-alphanumeric characters can also be treated as alphanumeric, if desired. *Chinese, Japanese, Thai:* Word-level tokenization used. *Chinese only:* Single-character tokenization can be added.
Stemming	*Western European Languages:* Supported. *Eastern European/Middle-Eastern languages:* Supported. *Japanese, Korean:* Supported. *Chinese, Thai:* Not applicable/not supported.

Table 3-2 Features of the multilanguage locale (continued)

Feature	Support
Normalization	*Japanese:* Half-width Kana indexed as equivalent full-width. Old Kanji indexed as equivalent New Kanji. ASCII indexed as equivalent double-byte Latin. Mixed Kanji/Kana words indexed as Kanji only. Hyphens removed from Kana. Okurigana supported for cases where the Okurigana Kanji stems are the same. **Note:** In a wildcard search, half-width–full-width Kana equivalence is not supported if the query term contains a voice-marked Kana—unless the voice-marked Kana is the leading character of a wildcard query.) *Chinese:* Traditional Chinese is indexed as Simplified Chinese.
Compound words	*Dutch, Finnish, German, Japanese:* Decomposition into subwords supported.
Part-of-speech	*Dutch, English, Finnish, French, German, Italian, Portuguese, Spanish, Chinese, Japanese, Korean:* Supported, including noun-phrase extraction. *Other languages:* Limited support (no noun-phrase extraction).
Number handling	*Chinese, Japanese, Korean:* Han script numbers indexed as Latin numbers, unless they appear in a non-numeric word.
Language-specific search	User can specify which language's rules to apply to a search query. Query operators must be in English only.
Case-insensitive search	*All Latin- and Cyrillic-based languages:* Supported and enabled by default. Auto-case capability also available. *Other languages:* Not supported.
Accent-insensitive search	Supported for languages that have accented characters.
Searchable symbols	All languages support search for symbol characters as defined by Unicode 3.2.
Synonym search	Supported for all languages, using a single thesaurus for all languages.
Soundex search	*English:* Supported. *Other languages:* Not supported.
Typo search	Supported.
Wildcard search	Supported.

Table 3-2 Features of the multilanguage locale (continued)

Feature	Support
Stop words	Stop-word lists for use in feature extraction and by the free-text query parser are provided. For more information, see "Stop Words" on page 54.
Date formatting	For date fields in a collection, only numeric or English date formats are supported.

Note The Verity tokenizer considers each document it indexes to be in only one language. In many cases, a document that is mainly in one language may have many words in other languages. In the `uni` locale, the tokenizer can process those words, called *foreign words*, according to the language rules defined by the Unicode language block those words' characters fall into.

One implication of this feature is that a language-specific stemmed search in the overall language of the document will not find occurrences of foreign words. (A literal search will still find any occurrences, regardless of language.)

Advanced European Locales

The single-language locales in this category serve the languages native to Western Europe. Use of these locales (other than `englishv`) requires a separate license and installation of the Verity Locales product.

The following table lists the currently available advanced European locales. Windows 1252 is the internal character set for all of these locales.

Table 3-3 Languages supported by the advanced European locales

Verity locale	Language	Verity locale	Language
`bokmalv`	Norwegian	`germanv`	German
`danishv`	Danish	`italianv`	Italian
`dutchv`	Dutch	`portugv`	Portuguese
`englishv`[a]	English	`spanishv`	Spanish

Table 3-3 Languages supported by the advanced European locales

Verity locale	Language	Verity locale	Language
`finnishv`	Finnish	`swedishv`	Swedish
`frenchv`	French		

a. Built-in; does not require installation of Verity Locales.

Note In the K2 Dashboard, the `englishv` locale is called **English (Advanced)**.

The advanced European locales support the indexing and search features described in Table 3-7.

Table 3-4 Features of the advanced European locales

Feature	Support
Character-set detection	Verity's internal auto-detection technology identifies the character set of source documents to be indexed. The locale accepts any compatible source-document character set, converts to 1252.
Language identification	Verity's language-detection technology identifies the languages of source documents to be indexed. (Indexing rules are based on the language assignment.)
Sorting order	`englishv` supports accent-sensitive searching. Other locales use case-insensitive and accent-insensitive sorting behavior based on Windows 1252 character set.
Tokenization	Simple-tokens behavior, in which nearly all non-alphanumeric characters can be word delimiters, is supported. Individual non-alphanumeric characters can also be treated as alphanumeric, if desired.
Stemming	Supported.
Normalization	No normalization applied.
Compound words	`dutchv`, `finnishv`, `germanv`: Decomposition into subwords supported.
Part-of-speech	`dutchv`, `englishv`, `finnishv`, `frenchv`, `germanv`, `italianv`, `portugv`, `spanishv`: Supported, including noun-phrase extraction. `bokmalv`, `danishv`, `swedishv`: Limited support (no noun-phrase extraction).
Number handling	No special number handling.

Table 3-4 Features of the advanced European locales (continued)

Feature	Support
Language-specific search	Search query always uses language rules of collection being searched.
Case-insensitive search	Supported and enabled by default. Auto-case capability also available.
Accent-insensitive search	Supported for locales that have accented characters.
Searchable symbols	With simple-tokens behavior, nearly all non-alphanumeric characters can be searched for as symbols.
Synonym search	Supported for all locales. Verity provides a simple default thesaurus for `englishv`.
Soundex search	`englishv`: Supported. *Other locales:* Not supported.
Typo search	Supported.
Wildcard search	Supported.
Stop words	Stop-word lists for use in feature extraction and by the free-text query parser are provided. For more information, see "Stop Words" on page 54.
Date formatting	For date fields in a collection, only numeric or English date formats are supported.

Asian (CJK) Locales

The single-language locales in this category serve the multiple-byte languages of East Asia: Chinese, Japanese, and Korean. Use of these locales requires a separate license and installation of the Verity Locales product.

The following table lists the currently available Asian locales and their internal character sets. For common names of the listed character sets, see "Locales, Character Sets and Languages" on page 127.

Table 3-5 Languages supported by the Asian locales

Verity Locale	Language	Charset
`japanb`	Japanese	sjis
`koreab`	Korean	ksc

Table 3-5 Languages supported by the Asian locales

Verity Locale	Language	Charset
`simpcb`	Chinese (simplified)	gb
`tradcb`	Chinese (traditional)	big5

Asian locales support the indexing and search features noted in Table 3-6.

Table 3-6 Features of the Asian locales

Feature	Support
Character-set detection	Verity's internal auto-detection technology identifies the character set of source documents to be indexed. If a document with an unknown character set is encountered during indexing, it is skipped.
Language identification	Verity's language-detection technology identifies the language of source documents to be indexed. (Indexing rules are based on the current locale.)
Sorting order	Controlled by internal character set. Not customizable.
Tokenization	`japanb`: Word-level tokenization used. `simpcb`, `tradcb`: Word-level tokenization used; single-character tokenization available. `koreab`: White-space separators control tokenization. *All locales*: tokenization of ASCII uses simple-tokens behavior.
Stemming	`japanb`, `koreab`: Supported. `simpcb`, `tradcb`: Not applicable.

Table 3-6 Features of the Asian locales (continued)

Feature	Support
Normalization	`japanb`: Half-width Kana equivalent to full-width Kana. Old Kanji equivalent to New Kanji. ASCII indexed as equivalent double-byte Latin. Mixed Kanji/Kana words indexed as Kanji only. Hyphens removed from Kana. Okurigana supported for cases where the Okurigana kanji stems are the same. **Note:** In a wildcard search, half-width–full-width Kana equivalence is not supported if the query term contains a voice-marked Kana—unless the voice-marked Kana is the leading character of a wildcard query.) `simpcb`, `tradcb`: Simplified text in a traditional document is indexed as traditional; traditional text in a simplified document is indexed as simplified. ASCII indexed as equivalent double-byte Latin. `koreab`: ASCII indexed as equivalent double-byte Latin.
Compound words	`japanb`: Deep decomposition of tokens, to recursively break down compound words, is supported.
Part-of-speech	Part-of-speech information is recorded at indexing. Limited noun-phrase capability is available.
Number handling	Han script numbers indexed as Latin numbers, unless they appear in a non-numeric word.
Language-specific search	Search query always uses language rules of collection being searched.
Case-insensitive search	Supported for all locales.
Accent-insensitive search	Not applicable.
Searchable symbols	Supported for all locales.
Synonym search	Supported for all locales.
Soundex search	Not supported.
Typo search	Not supported.
Wildcard search	Supported for all locales.
Stop words	Stop-word lists for use in feature extraction and by the free-text query parser are provided for all locales.
Date formatting	For date fields in a collection, only numeric or English date formats are supported.

The english Locale

The `english` locale is the original English-language locale provided with Verity K2 and VDK. It is a simple, table-driven locale that provides basic, customizable capabilities for tokenization of plain (7-bit) text in English.

Note In the K2 Dashboard, the `english` locale is called **English (Basic)**.

In general, Verity recommends that you do not use the `english` locale because of its limited tokenization and stemming capabilities, and because it does not support 8-bit (or larger) character sets; it is restricted to 7-bit characters, such as plain ASCII.

The `english` locale supports the indexing and search features described in Table 3-7.

Table 3-7 features of the english locale

Feature	Support
Character-set detection	Verity's auto-detection technology identifies the character set of source documents to be indexed. If a document with an unknown character set is encountered during indexing, it is assigned the `english` locale's internal character set (ASCII).
Language identification	Verity's language-detection technology identifies the language of source documents to be indexed. (Indexing rules are based on the `english` locale, not the document language.)
Sorting order	Defined by `SORTORDER` table.
Tokenization	Performed by Verity's internal lexer. Limited customization possible.
Stemming	Limited, rule-based.
Normalization	No normalization applied.
Compound words	Not supported.
Part-of-speech	Not supported.
Number handling	No special number handling.
Language-specific search	Search query always assumes English.
Case-insensitive search	Supported, using `TOUPPER` and `TOLOWER` tables.
Accent-insensitive search	Not supported.
Searchable symbols	Supported, using `style.lex` and `CTYPE` table.
Synonym search	Supported.

Table 3-7 features of the english locale (continued)

Feature	Support
Soundex search	Supported.
Typo search	Supported.
Wildcard search	Supported.
Stop words	Supported.
Date formatting	Supported.

The functioning of the internal lexer can be configured to a limited extent through the style file `style.lex`. In that file, you can specify which characters are to be considered white space, punctuation, or searchable alphanumerics (word components).

You can further customize the functioning of the `english` locale by modifying the file `acro20.lng`. That file contains the tables `CTYPE`, `TOUPPER`, `TOLOWER`, and `SORTORDER`, plus stemming rules. This type of customization typically requires the assistance of Verity Professional Services.

For more information on `style.lex`, see the chapter on index tuning in the *Verity Collection Reference*.

Legacy European Locales

The single-language locales in this category serve the languages native to Western Europe. Use of these locales requires a separate license and installation of the Verity Single-Language European Locales product.

Table 3-8 lists the available legacy European locales. Windows 1252 is the internal character set for all of these locales.

Table 3-8 Languages supported by the legacy European locales

Verity locale	Language	Verity locale	Language
`bokmalx`	Norwegian	`germanx`	German
`danishx`	Danish	`italianx`	Italian
`dutchx`	Dutch	`nynorskx`	Norwegian
`englishx`	English	`portugx`	Portuguese

Table 3-8 Languages supported by the legacy European locales

Verity locale	Language	Verity locale	Language
`finnishx`	Finnish	`spanishx`	Spanish
`frenchx`	French	`swedishx`	Swedish

Note In the K2 Dashboard, the `englishx` locale is called **English (backward compatible)**.

The legacy European locales support the indexing and search features described in Table 3-7.

Table 3-9 Features of the legacy European locales

Feature	Support
Character-set detection	Verity's auto-detection technology identifies the character set of source documents to be indexed. If a document with an unknown character set is encountered during indexing, it is assigned the locale's internal character set.
Language identification	Verity's language-detection technology identifies the language of source documents to be indexed. (Indexing rules are based on the current locale, not the document language.)
Sorting order	All locales use case-insensitive and accent-insensitive sorting behavior based on Windows 1252 character set.
Tokenization	Performed by all locales. All locales support simple-tokens behavior, in which nearly all non-alphanumeric characters can be word delimiters. Individual delimiters can also be removed from the delimiters list, if desired.
Stemming	All locales support stem indexing and search.
Normalization	No normalization applied.
Compound words	Decomposition into subwords supported by `finnishx` and `germanx`.
Part-of-speech	All locales support part-of-speech, including noun-phrase extraction.
Number handling	No special number handling.
Language-specific search	Search query always uses language rules of collection being searched.
Case-insensitive search	Supported by all locales and enabled by default. Auto-case capability also available for all locales.
Accent-insensitive search	Supported for all locales.

Table 3-9 Features of the legacy European locales (continued)

Feature	Support
Searchable symbols	All locales support defining "searchable non-alphabet" characters.
Synonym search	All locales support use of thesaurus for synonym search. Verity provides a simple default thesaurus for several of the locales.
Soundex search	Supported by `englishx` only.
Typo search	Supported by all locales.
Wildcard search	Supported by all locales.
Stop words	All locales support use of a locale-specific stop-word list for use in feature extraction and free-text queries. Verity provides a simple default stop-word file for each locale.
Date formatting	For date fields in a collection, all locales support dates with month and day names in the locale's language.

4

Configuring Locales

After installing one or more locales, you can use them immediately. However, you also can reconfigure certain aspects of their behavior to customize the language handling and search characteristics of your application.

This chapter describes how to perform those configurations. It also gives suggestions for setting locale and character set when using Verity tools.

Note After changing any of the locale-specific settings described in this chapter, you must re-index existing collections that use that locale. Otherwise, search results may be unpredictable.

This chapter includes the following sections:

Making Locale-Specific Settings

This section summarizes the kinds of locale-specific configurations you can make. The subsequent sections show how to make them for particular locales.

Note After making any of the changes described in this section to a given locale, you must re-index existing collections that use that locale. Otherwise, search results may be unpredictable.

Tokenization

Verity locales allow you to change certain aspects of tokenization behavior by making modifications to locale configuration files. Different locales handle tokenization customization differently.

Stemming

In a stemmed search (see "Stemming" on page 47), all variations of a search term's root word are returned. For stemmed search to function, the indexing process must extract and index the stems of all words that it encounters. For some languages, *multistemming*—the extraction of multiple stems for a single word—is supported.

K2 also supports customization of the dictionaries that control stemming.

Noun- and Noun-Phrase Extraction

During indexing, the Verity tokenizer can for some languages analyze words for part-of-speech information. The capture of nouns and the construction of noun phrases, which depends on part-of-speech identification, is used to extract concepts and document summaries from a collection (see "Part-of-Speech Identification" on page 50).

If your Verity installation does not make use of analysis of nouns or noun phrases, you might be able to improve indexing performance by ensuring that noun-phrase extraction is disabled. You might get further performance improvement by also disabling part-of-speech identification.

Conversely, if accurate feature and concept extraction is important, you should ensure that both part-of-speech identification and noun-phrase extraction are enabled.

For some languages in the `uni` locale, K2 supports customization of the dictionaries that control noun extraction.

Word Decomposition

In languages that have compound words, Verity locales by default create tokens for all of a compound word's subwords (as well as for the compound word itself). K2 also supports customization of the dictionaries that control word decomposition.

Search Characteristics

For some locales, you can change certain aspects of search behavior by making the modifications described in this section.

Case-Sensitive Search

All locales have built-in support for case-sensitive searching. For Asian locales whose native languages do not have the concept of case, case-sensitive searching is still supported for ASCII characters.

Enabling case-sensitivity is not strictly a locale issue. To disable or enable case-sensitive searching on a per-collection basis, use the `Casedex` value in the `$define` directive in the collection's `style.prm` file. For more information, see the index-tuning chapter of the *Verity Collection Reference*.

Auto-Case

As described in "Case-Insensitive Search" on page 52, auto-case is a Verity search feature in which query terms that are single-case (all uppercase or all lowercase) are matched case-insensitively, whereas mixed-case query terms are matched case-sensitively. You can enable or disable the auto-case capability for a given locale. (Auto-case does not apply to Asian locales.)

Thesauruses for Synonym Search

All Verity locales support the use of a thesaurus, or synonym list, for searching. In a synonym search, all occurrences of the search term plus any of its synonyms are returned (see "Synonym Search" on page 53).

To enable synonym search for a given locale, you need to implement a thesaurus containing the lists of synonyms. For some locales, Verity provides a basic thesaurus that you can use as-is or further customize; for other locales, you need to create your own thesaurus.

Only one thesaurus file is allowed per locale. If you implement a properly constructed thesaurus, give it the required name (`vdk30.syd`), and place it at the top level of your locale's directory, it will be used for synonym search.

Because it supports many languages, the multilanguage locale is a special case. See "Thesaurus Setup for uni" on page 88 for instructions on how to use a thesaurus with the multilanguage locale.

For detailed instructions on creating and installing a custom thesaurus, see the *Verity Query Language and Topic Guide*.

Configuring the Multilanguage Locale

Tokenization

When it indexes a document, the Verity tokenizer breaks words at whitespace and punctuation characters (see "Tokenization and Word Delimiters" on page 45). The multilanguage locale by default uses simple-tokens behavior, which means that the full set of delimiters listed in Table B-1 on page 143 is used to control tokens, subject to customizations described here.

You can change tokenization behavior in general by disabling simple tokens, or more specifically by

- modifying the set of symbols that are considered punctuation for tokenization purposes
- redefining certain characters as searchable symbols (see examples in "Symbol Search" on page 53)
- redefining certain symbols as alphabetic characters
- Redefining certain symbols as alphabetic, but only when they do *not* occur at the beginning or the end of a word

If you want to change the set of characters used for tokenization, or make any of the other changes just listed, you can apply the changes globally—to all languages for which the multilanguage locale provides linguistic support (see Table 3-1 on page 64)—or you can apply them to an individual language.

Global Changes to Tokenization

You can change the set of characters used across all languages for punctuation or symbols, and you can disable or enable simple tokens behavior for all languages at once.

Modifying Symbols and Punctuation

To globally change the set of characters used for punctuation or as symbols, take these steps:

1. Open the file `uni.cfg`, in the directory *`productDir`*`\common\uni`.
2. If there are characters that you want to specify as punctuation (unsearchable and not part of any word), locate the `treat_as_punctuation` block within the global `post-process` section. (If the block is commented out, uncomment the parts of it you are going to use.)

   Modify the block like this:

   - To specify individual characters, follow the `chars:` label with a space-separated list of the character codes. (All character codes must be Unicode.)
   - To specify a range of characters, follow the `range:` label with the first and last character codes in the range, separated by a space. (All character codes must be Unicode.)

   You can combine a range and a list of individual characters in a block, and you can have more than one `range:` statement.

IMPORTANT All character codes you specify must be in Unicode encoding. See Table B-1 on page 143 for a list of delimiters and symbols with their Unicode values.

3. If there are characters that you want to specify as symbols (searchable as words themselves but not part of any other word), locate the `treat_as_symbol` block, also within the global `post-process` section.

   Modify the block in the same manner as described in the previous step. Uncomment it if it is commented out, and specify individual character codes and/or code ranges.

4. If there are symbol characters (for example, _) that you want to be treated as regular alphabetic characters, specify them in the `treat_as_alphabetic` block, using individual codes or code ranges, just as with the other blocks.
5. If there are symbol characters (for example, **&** or **-**) that you want to be treated as alphabetic, but only when they do *not* occur at the beginning of a word, specify them in the `not_allowed_leading_char` block.

   Note If you specify a character in this block (or the next) and also in the `treat_as_symbol` or `treat_as_alphabetic` block, `treat_as_symbol` or `treat_as_alphabetic` takes precedence.

6. If there are symbol characters that you want to be treated as alphabetic, but only when they do not occur at the end of a word, specify them in the `not_allowed_trailing_char` block.
7. Save and close `uni.cfg`.

Disabling and Enabling Simple Tokens

If you disable simple tokens, a much smaller set of punctuation is used to control tokenization. Note that simple tokens is not necessarily the most desirable indexing behavior in all cases. For example, for the purpose of extracting document features for summarization, longer tokens are in general more desirable than shorter ones. In that case, disabling simple tokens might yield better results.

To disable or re-enable simple tokens, take these steps:

1. Open `uni.cfg`.
2. In the post-processing block near the top of the file, locate this line

   ```
   simple_tokens: yes
   ```

3. Either comment out the line, like this:

   ```
   #simple_tokens: yes
   ```

   or change the variable's value, like this:

   ```
   simple_tokens: no
   ```

4. Save and close `uni.cfg`.

Per-Language Changes to Tokenization

To make any of these changes on a per-language basis instead of globally, you need to construct a post-processing block for that individual language in `uni.cfg`. For example, this is the Spanish block in the default `uni.cfg`:

```
#Spanish
language: es
{
  driver: "unitera -lang es ...
}
```

Within that block, create a `post-process` block and add the changes you want for that language. For example, to force the ampersand (`&`) and dash(`-`) to be treated as ordinary alphabetic characters in Spanish, modify the Spanish block like this:

```
#Spanish
language: es
{
  driver: "unitera -lang es ...

  post-process:
  {
    treat_as_alphabetic:
    {
      chars: 0x26 0x2d
    }
  }
}
```

As another example, to disable simple tokens for Spanish only, modify the block like this:

```
#Spanish
language: es
{
  driver: "unitera -lang es ...

  post-process:
  {
    simple_tokens: no
  }
}
```

Note Per-language settings take precedence over global settings.

Enabling Single-Character Tokenization

For the Asian languages (simplified Chinese, traditional Chinese, Japanese, and Korean) in the multilanguage locale, you can force inclusion of every native-script character as a separate token (in addition to the normal word-level tokenization that occurs) by using the `-single_char` option in `uni.cfg`.

To make this change globally (for all four languages), you can uncomment the statement

```
#single_char: ja ko zh zt
```

in the global `post-process` section near the top of `uni.cfg`. To make it apply to only some of the languages, uncomment it and edit it to include only the languages you want it to apply to.

Because this statement is by default commented out in `uni.cfg`, single-character tokenization is not the default behavior.

Stemming

Stem indexing is enabled by default in the multilanguage locale.

Disabling Multistemming

You can specifically disable only multistemming in the multilanguage locale. There is no option for completely disabling stemming. To disable multistemming in the multilanguage locale, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\uni`.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver:  "locuni -decompound"  "locuni"
   ```

3. Add the option `-nomstem`, so the statement looks something like this:

   ```
   driver:  "locuni -decompound -nomstem"  "locuni"
   ```

4. Save and close the file.

To re-enable multistemming, remove the `-nomstem` option from the `driver` statement.

Customizing Stemming

For non-Asian languages, the multilanguage locale allows you to create a custom dictionary to control stemming. See "Custom Dictionaries for Non-Asian Languages" on page 150 for details.

Noun- and Noun-Phrase Extraction

In the multilanguage locale, noun extraction (part-of-speech identification) and noun-phrase extraction are enabled by default for those languages that can support them.

To disable these features in the multilanguage locale, take these steps:

1. Open the file `loc00.lng`, in the directory *`productDir`*`\common\uni`.
2. In the `locale` block, locate the `driver` statement, which by default looks something like this:

   ```
   driver:  "locuni -decompound"  "locuni"
   ```

3. To disable noun-phrase extraction, add the `-nonnp` option, leaving it like this:

   ```
   driver:  "locuni -nonnp -decompound"  "locuni"
   ```

4. To also disable part-of-speech identification, add the `-nopos` option to the line as well, like this:

   ```
   driver:  "locuni -nonnp -nopos -decompound"  "locuni"
   ```

5. Save and close the file.

To once more enable noun-phrase extraction and part-of-speech identification, remove the options from the `driver` statement.

If you have re-enabled noun-phrase extraction in the `uni` locale, you must also make or verify some settings in the `style.prm` file for any collection that you create with this locale:

1. If you want to store feature vectors in the collection (required for clustering, summarization, and recommendation), and if you want those feature vectors to be

based on nouns or noun phrase s, verify that one of the following lines is uncommented in `style.prm`:

```
$define DOC-FEATURES "NP"
```

or

```
$define DOC-FEATURES "NNP"
```

2. if you want the nouns and noun phrases themselves to be stored in the collection (required for thematic mapping), verify that the following two lines (one for noun storage, one for noun--phrase storage) are uncommented in `style.prm`:

```
$define NOUN-IDXOPTS ""
$define NPHR-IDXOPTS ""
```

These elements of `style.prm` are described more fully in the chapter on tuning collections in the *Verity Collection Reference*.

Marking All Words as Nouns

For languages in the `uni` locale for which part-of-speech identification is not available, it is still possible to support feature extraction or concept extraction at a lesser level. You can make a setting in the locale's `uni.cfg` file that will cause all words to get marked as nouns. The resultant extracted features or concepts can be valuable, if less accurate than they would be with true part-of-speech identification.

To make this change globally (for all linguistically supported languages), take these steps:

1. Open the file `uni.cfg`, in the directory *`productDir`*`\common\uni`.
2. In the `post-process` block near the top of the file, locate this statement:

```
#mark_as_noun: no
```

3. Uncomment the line and change `no` to `yes`, like this:

```
mark_as_noun: yes
```

4. Save and close `uni.cfg`.

To make the change on a per-language basis, create a `post-process` block for the desired language and place the edited statement into it.

Customizing Noun Extraction

The multilanguage locale allows you to create a custom dictionary to control part-of-speech identification. See "Custom Dictionaries for Non-Asian Languages" on page 150 for details.

Disabling Word Decomposition

To disable word decomposition for the multilanguage locale, take these steps:

1. Open the file `loc00.lng`, in the directory *`productDir`*`\common\uni`.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver:  "locuni -decompound"  "locuni"
   ```

3. To disable word decomposition, remove the `-decompound` option, leaving something like this:

   ```
   driver:  "locuni"  "locuni"
   ```

4. Save and close the file.

To reinstate word decomposition, replace the `-decompound` option in the `driver` statement.

Customizing Word Decomposition

The multilanguage locale allows you to create two kinds of custom files for controlling word decomposition:

- For the Asian languages Japanese, Korean, Simplified Chinese, and Traditional Chinese, you can create a user dictionary into which you can place words that you want decomposed in a non-standard manner. See "Custom Dictionaries for Asian Languages" on page 147 for details.
- For non-Asian languages, you use a command-line tool to create a custom language dictionary. See "Custom Dictionaries for Non-Asian Languages" on page 150 for details.

Search Characteristics

Auto-Case

For the multilanguage locale, auto-case is disabled by default. If you want to enable auto-case, take these steps:

1. Open the locale definition file `loc00.lng`, in the directory *`productDir`*`\common\uni`.
2. In the `locale-flags` block, locate the `AutoCase` entry:

```
locale_flags:
{
...
NoAutoCase: yes
...
```

3. Change the value of `NoAutoCase` from `yes` to `no`.
4. Save and close the file.

Thesaurus Setup for uni

A Verity K2 thesaurus file (`vdk30.syd`) can support only one language, and each Verity locale can be configured for only one thesaurus file at a time.

The multilanguage locale is by default configured to use an English thesaurus; the English-language version of `vdk30.syd` is in the directory *`installDir`*`/k2/common/uni`.

K2 also includes thesauruses for the following additional languages:

- Dutch
- English
- French
- German
- Italian
- Portuguese
- Spanish

- Swedish

Each of these thesauruses is a `vdk30.syd` file in its own language-specific subdirectory within the `uni` directory. For example:

`installDir``/k2/common/uni/nl/vdk30.syd` for Dutch

`installDir``/k2/common/uni/fr/vdk30.syd` for French

To support synonym search in language other than English, replace the English version of vdk30.syd in the directory *`installDir`*`/k2/common/uni` with a copy of the `vdk30.syd` file from one of these language-specific directories.

For information on creating custom thesauruses for additional languages, see the *Verity Query Language and Topic Guide*.

Special Setup Issues

This section describes some configuration requirements and some performance-improving suggestions that are unique to the `uni` locale.

Enabling Language Identification

The `uni` locale needs to assign a language to every document that it indexes. Therefore, the language-identification filter (`flt_lang`) must be used whenever a collection is created in the `uni` locale.

By default, the language-identification filter is enabled. To confirm that it is enabled, or to re-enable it if it has been disabled previously, take these steps:

1. Open the version of the universal-filter configuration file (`style.uni`) that you will use to create the collection. (The original file is in the directory *`productDir`*`\common\style`.)
2. Verify that the following line exists and is uncommented:

```
postformat: "flt_lang "
```

   If the line is there but commented, remove the comment mark (#). Make sure all other `postformat:` statements are commented out.

3. Save and close the file.

For more details on `flt_lang`, see the discussion on configuring the language-identification filter and `style.uni` in the chapter "Filtering and Formatting Documents" in the *Verity Collection Reference*.

Specifying the KeyView Filter for PDF Documents

The Adobe PDF document filter always generates tokens in the Windows 1252 character set, without making use of locale-specific tokenization behavior. This is incompatible with the `uni` locale, which requires UTF-8. Therefore, the KeyView filter must be used for processing PDF documents for the `uni` locale.

Note If you use the KeyView filter instead of the PDF filter for PDF documents, highlighting of search terms is not available when users view the documents in Acrobat Reader.

By default, the KeyView filter is specified for processing PDF documents. If, however, the setting has been changed, you can restore it by taking these steps:

1. Open the version of `style.uni` that you will use to create the collection. (The original file is in the directory *`productDir`*`\common\style`).
2. Locate the following lines:

```
  type: "application/pdf"
    /format-filter = "flt_pdf"
#   /format-filter = "flt_kv"
...
    /charset       = none
```

3. Comment out the PDF filter line, uncomment the KeyView filter line, and assign a character set, like this:

```
  type: "application/pdf"
#   /format-filter = "flt_pdf"
    /format-filter = "flt_kv"
...
    /charset       = UTF-8
```

4. Save and close the file.

Improving uni Indexing Performance

You can take the following steps to improve indexing speed for the `uni` locale:

- Verify that noun-phrase extraction is disabled.
  (See "Noun- and Noun-Phrase Extraction" on page 85.)
- Disable part-of-speech identification.
  (See "Noun- and Noun-Phrase Extraction" on page 85.)

- Disable multi-stemming.
(See "Stemming" on page 84.)
- Disable word decomposition.
(See "Disabling Word Decomposition" on page 87.)
- Configure language identification.
(See "Configuring Language Identification" on page 106.)

Configuring Advanced European Locales

Tokenization

The advanced European locales by default use simple-tokens behavior. To change this, or to modify the set of searchable symbols or alphabetic characters, follow the procedures described for the multilanguage locale (see "Tokenization" on page 80)—except that

- You edit the `uni.cfg` file found in *`productDir`*`\common\`*`localeName`*, where *`localeName`* is the name of the locale (such as `germanv`) to be customized.
- You supply all character codes in the 1252 character set. (Table B-1 on page 143 lists standard delimiters and symbols with their 1252 code values.)

Stemming

Stem indexing is enabled by default in the advanced European locales.

Per-Collection Stemming

There is no locale-specific control on stemming. If you want to disable or enable stemming for a collection built in an advanced European locale, use the `Stemdex` value in the `$define` directive in the collection's `style.prm` file.

1. Open the version of `style.prm` that you are using to create the collection. (The original default version is in the directory *`productDir`*`\common\style`.)
2. Locate the `$define WORD-IDXOPTS` directive. If it looks like this:

```
$define WORD-IDXOPTS    "Stemdex Casedex"
```

change it to this:

```
$define WORD-IDXOPTS    "Casedex"
```

3. Save and close the file.

For more information on `style.prm`, see the index-tuning chapter of the *Verity Collection Reference*.

Customizing Stemming

The advanced European locales allow you to create a custom dictionary to control stemming. See "Custom Dictionaries for Non-Asian Languages" on page 150 for details.

Noun- and Noun-Phrase Extraction

In the advanced European locales, noun extraction (part-of-speech identification) and noun-phrase extraction are enabled by default.

Note Noun-phrase extraction is not available in `bokmalv`, `danishv`, or `swedishv`.

To disable these features in an advanced European locale, take these steps:

1. Open the file `loc00.lng`, in the directory *`productDir`*`\common\`*`localeName`*, where *`localeName`* is the name of the locale (such as `frenchv`) to be customized.
2. In the `locale` block, locate the `driver` statement, which by default looks something like this:

```
driver:  "locuni -decompound -nostemsuffix"  "locuni"
```

3. To disable noun-phrase extraction, add the `-nonnp` option, leaving it like this:

```
driver:  "locuni -nonnp -decompound -nostemsuffix"  "locuni"
```

4. To also disable part-of-speech identification, add the `-nopos` option to the line as well, like this:

```
driver:  "locuni -nonnp -nopos -decompound -nostemsuffix"  "locuni"
```

5. Save and close the file.

To once more enable noun-phrase extraction and part-of-speech identification, remove the options from the `driver` statement.

If you have re-enabled noun-phrase extraction in an advanced European locale, you must also make or verify some settings in the `style.prm` file for any collection that you create with the locale:

1. If you want to store feature vectors in the collection (required for clustering, summarization, and recommendation), and if you want those feature vectors to be based on nouns or noun phrase s, verify that one of the following lines is uncommented in `style.prm`:

   ```
   $define DOC-FEATURES "NP"
   ```

   or

   ```
   $define DOC-FEATURES "NNP"
   ```

2. if you want the nouns and noun phrases themselves to be stored in the collection (required for thematic mapping), verify that the following two lines (one for noun storage, one for noun--phrase storage) are uncommented in `style.prm`:

   ```
   $define NOUN-IDXOPTS ""
   $define NPHR-IDXOPTS ""
   ```

These elements of `style.prm` are described more fully in the chapter on tuning collections in the *Verity Collection Reference*.

Customizing Noun Extraction

The advanced European locales allow you to create a custom dictionary to control part-of-speech identification. See "Custom Dictionaries for Non-Asian Languages" on page 150 for details.

Disabling Word Decomposition

To disable word decomposition for those advanced European locales (such as `germanv`) that support it, take these steps:

1. Open the file `loc00.lng`, in the directory *`productDir`*`\common\`*`localeName`*, where *`localeName`* is the name of the locale (such as `italianv`) to be customized.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver:  "locuni -decompound -nostemsuffix"  "locuni"
   ```

3. To disable word decomposition, remove the `-decompound` option, leaving something like this:

```
driver:  "locuni -nostemsuffix"  "locuni"
```

4. Save and close the file.

To reinstate word decomposition, replace the `-decompound` option in the `driver` statement.

Customizing Word Decomposition

In advanced European locales, stemming, decomposition, and part-of-speech extraction are handled with a dictionary that you can customize. To create a user dictionary to use with an advanced European locale, see "Custom Dictionaries for Non-Asian Languages" on page 150.

Search Characteristics

Auto-Case

In the advanced European locales, auto-case is disabled by default. If you want to enable auto-case, take these steps:

1. Open the locale definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`localeName`*, where *`localeName`* is the name of the locale (such as `dutchv`) to be customized.
2. In the `locale-flags` block, locate the `AutoCase` entry:

```
locale_flags:
{
...
NoAutoCase: yes
...
```

3. Change the value of `NoAutoCase` from `yes` to `no`.
4. Save and close the file.

Configuring Asian Locales

Tokenization

You can influence tokenization in an Asian locale by changing options in its `loc00.lng` file.

Disabling and Enabling Simple Tokens

For Asian locales, simple-tokens behavior is enabled by default. If you disable simple tokens, a much smaller set of symbols—just the standard set of punctuation marks—is used to control tokenization of Latin-based characters.

Simple tokens is not necessarily the most desirable indexing behavior in all cases. For example, for the purpose of extracting document features for summarization, longer tokens are in general more desirable than shorter ones. In that case, disabling simple tokens might yield better results.

To disable simple tokens, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver: "loc_basis -simple_tokens _-& -nostems" "loc"
   ```

3. To disable simple tokens, remove the `-simple_tokens` option, leaving something like this:

   ```
   driver: "loc_basis _-& -nostems" "loc"
   ```

4. Save and close the file.

To re-enable simple tokens, restore the `-simple_tokens` option in the `driver` statement.

Enabling Single-Character Tokenization

For the Asian locales `simpcb` and `tradcb`, you can force inclusion of every native-script character as a separate token (in addition to the normal word-level tokenization that occurs) by using the `-single_character` option in `loc00.lng`. This single-character behavior is the default. The driver statement in these two locales looks like this:

```
driver:  "locbasis -simple_tokens _-& -single_character" "loc"
```

You can disable or enable simple-tokens and single-character behavior independently of each other.

Stemming

Stem indexing is enabled by default in the Asian locales `japanb` and `koreab`. Stemming is not applicable to the Chinese locales.

You might wish to disable stemming for Japanese and Korean. Disabling stemming can speed indexing—at the expense of supporting stemmed search, of course.

To disable stemming in the `japanb` locale or the `koreab` locale, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which for the `japanb` locale should look something like this:

   ```
   driver:  "locbasis -simple_tokens _-&"  "loc"
   ```

3. Add the option `-no_stems`, so the statement looks something like this:

   ```
   driver: "locbasis -simple_tokens _-& -no_stems"  "loc"
   ```

4. Save and close the file.

To re-enable stemming, remove the `-no_stems` option from the `driver` statement.

Noun- and noun-Phrase Extraction

In Asian locales, noun extraction (part-of-speech identification) and noun-phrase extraction are enabled by default.

To disable noun-phrase extraction in an Asian locale, take these steps:

1. Open the file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.

2. In the `locale` block, locate the `driver` statement, which by default should look something like this:

   ```
   driver:  "locbasis -simple_tokens _-&" "loc"
   ```

3. Add the `-no_nps` option to the line, like this:

   ```
   driver:  "locbasis -simple_tokens _-& -no_nps" "loc"
   ```

4. Save and close the file.

To re-enable noun-phrase extraction, remove the `-no_nps` option from the `driver` statement.

If noun-phrase extraction is enabled in an Asian locale, you must also be sure it is enabled for each collection you create with that locale. In the `style.prm` file for the collection, verify that the following two lines (one for noun extraction, one for noun--phrase extraction) are uncommented:

```
$define NOUN-IDXOPTS ""
$define NPHR-IDXOPTS ""
```

These definitions are described more fully in the chapter on tuning collections in the *Verity Collection Reference*.

Customizing Word Decomposition

The Verity `japanb`, `koreab`, `simpcb`, and `tradcb` locales allow you to create a custom file, called a user dictionary, into which you can place words that you want decomposed in a non-standard manner. See "Custom Dictionaries for Asian Languages" on page 147 for the details of how to create the dictionary.

Improving Performance

You can take the following steps to improve indexing speed for the Asian locales:

- Disable noun-phrase extraction.
  (See "Noun- and noun-Phrase Extraction" on page 96.)
- Disable stemming.
  (See "Stemming" on page 84.)

Configuring Legacy European Locales

Tokenization

The legacy European locales by default have simple tokens enabled. To modify the set of token delimiters used, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver: "loc_xlt -simple_tokens
   -tokenized_as_alphabet -_& -searchable_non_alphabet..." "xlt"
   ```

   (Note that the `-tokenized_as_alphabet` option in this locale already specifies three characters—hyphen, underscore, ampersand—that are to be treated as alphabetical characters instead of token delimiters.)
3. If the `-tokenized_as_alphabet` option is not present, add it after the `-simple_tokens` option and follow it with the symbols that you want to remove from the list of token delimiters.
4. If the `-tokenized_as_alphabet` option is already present, add or remove symbols to change the list. Adding a symbol here means that it is *not* to be considered a delimiter.

   The full set of symbols available as token delimiters is listed in "Tokenization Delimiters" on page 143.
5. Save and close the file.

Note The symbol **+** is always treated as a delimiter, because it has special meaning in the Verity Query Language. However when **+** appears at the end of a word—that is, if it is followed by white space or another delimiter—it is not treated as a delimiter. This keeps terms such as such as **C++** from being split up.

See "Tokenization Example" on page 99 for an illustration of how these settings affect tokenization results.

Making Symbols Searchable

By default, non-alphanumeric symbols are not searchable. However, if simple tokens is enabled for a locale, you can make certain symbols searchable. (See examples in "Symbol Search" on page 53.)

This feature is fully supported only for the legacy European locales.

To make symbols searchable, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver: "loc_xlt -simple_tokens...
   -searchable_non_alphabet #$%¡§«°±»¿" "xlt"
   ```

   (Note that the `-searchable_non_alphabet` option in this locale already specifies ten characters—three ASCII and seven extended ASCII—that are to be treated as searchable symbols.)
3. If the `-searchable_non_alphabet` option is not present, add it after the `-simple_tokens` option (or after the `-tokenized_as_alphabet` option, if it is present) and follow it with the symbols that you want to be searchable.
4. If the `-searchable_non_alphabet` option is already present, add or remove symbols to change what can be searched.

   The full set of symbols available as token delimiters is listed in "Tokenization Delimiters" on page 143.
5. Save and close the file.

See "Tokenization Example" on page 99 l for an illustration of how these settings affect tokenization results.

Tokenization Example

The table in this section shows two examples of tokenization in a legacy European locale, applied to the following (nonsensical) content:

```
#12:34-56 webmaster@verity.com verity©2003 hi|bye C++ R&D
```

The table lists the results of tokenization for two settings:

- Without the `-simple_tokens` option

- With these three options:
  ```
  -simple_tokens
  -tokenized_as_alphabet -_&
  -searchable_non_alphabet #$%©®¢£¥™
  ```

(This is equivalent to the default simple-tokens behavior.)

The table also lists selected query strings that could be applied to the tokenized document, specifying for each whether the query will yield a search hit with simple tokens on or off.

Table 4-1 Tokenization example

Tokens generated			Search hit?	
(simple off)	(simple on)	Example queries	(simple off)	(simple on)
#12:34-56	12	12:34-56	Yes	Yes
	34	12:34	No	Yes
	56	34	No	Yes
	#	:#[a]	No	Yes
webmaster@verity.com	webmaster	webmaster@verity.com	Yes	Yes
	verity	webmaster	No	Yes
	com	verity.com	No	Yes
		com	No	Yes
		@[b]	No	No
verity©2003	verity	verity©2003	Yes	Yes
	2003	verity	No	Yes
	©	2003	No	Yes
		©[a]	No	Yes
hi|bye	hi	hi|bye	Yes	Yes
	bye	bye	No	Yes
		|[b]	No	No
C++	C++	C++[c]	Yes	Yes
R&D	R&D	R&D	Yes	Yes
		R	No	No
		D	No	No
		&[d]	No	No

a. By default, this symbol is searchable if simple tokens is on.
b. By default, this symbol is not searchable if simple tokens is on.

c. **+** is not a delimiter if followed by another delimiter.
d. **&** is by default a delimiter, but for this example it has been excluded from the delimiter list.

Per-Collection Stemming

Stem indexing is enabled by default in the legacy European locales. For these locales, there is no locale-specific control on stemming. If you want to disable or enable stemming for a collection built in a legacy European locale, use the `Stemdex` value in the `$define` directive in the collection's `style.prm` file.

1. Open the version of `style.prm` that you are using to create the collection. (The original default version is in the directory *`productDir`*`\common\style`.)
2. Locate the `$define WORD-IDXOPTS` directive. If it looks like this:

   ```
   $define WORD-IDXOPTS    "Stemdex Casedex"
   ```

   change it to this:

   ```
   $define WORD-IDXOPTS    "Casedex"
   ```

3. Save and close the file.

For more information on `style.prm`, see the index-tuning chapter of the *Verity Collection Reference*.

Per-Collection Noun- and Noun-Phrase Extraction

In legacy European locales, noun extraction (part-of-speech identification) and noun-phrase extraction are enabled. You can disable them on a per-collection basis by using the `$define` directive in the collection's `style.prm` file.

1. Open the version of `style.prm` that you are using to create the collection.
2. Locate the following directives:

   ```
   $define NOUN-IDXOPTS ""

   $define NPHR-IDXOPTS ""
   ```

3. Comment them out, like this:

   ```
   #$define NOUN-IDXOPTS ""

   #$define NPHR-IDXOPTS ""
   ```

4. Save and close the file.

For more information on `style.prm`, see the index-tuning chapter of the *Verity Collection Reference*.

Changing Search Characteristics

Auto-Case

For the legacy European locales, auto-case is enabled by default. If you want to disable auto-case, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale-flags` block, locate the `AutoCase` entry:

```
locale_flags:
{
...
NoAutoCase: no
...
```

3. Change the value of `NoAutoCase` from `no` to `yes`.
4. Save and close the file.

Making Other Language-Related Settings

Redefining the Default Session Locale

If your installation has special requirements, you can optionally redefine the default session locale for Verity applications and tools (see "Default Locales and the Session Locale" on page 61). You might want to do this as a convenience if all the collections at your installation are in the older Verity locale `english`, rather than the multilanguage locale (`uni`), which is the installed default.

Note You can use this technique to switch the default session locale between `english` and `uni` only; use of any other locale as the session default is not supported.

The default locale that you can change is the default installation locale, which can be specified for K2 installations in the Verity configuration file (`verity.cfg`). Take these steps to change it:

1. Open `verity.cfg`, in the directory *`productDir`*`/common`, where *`productDir`* is the path to the directory containing the component of Verity that has been installed (for example, `usr/verity/k2_6/k2` for K2 Services on UNIX, or `C:\Verity\Intelligent Classifier` for VIC on Windows).
2. In the `[GENERAL]` section of the file, locate the following entry (if it exists):

   ```
   locale=uni
   ```

3. If the entry exists, change it to

   ```
   locale=english
   ```

4. If the entry does not already exist, add the changed version of it to the file.
5. Save and close the file.

Changing Formatting

For all locales, you can make the text-formatting changes described here.

Changing Date Formatting

As installed, each locale includes a date-ordering convention. The convention specifies the order in which the elements of a date (day, month, and year) must appear in date fields in a collection.

You should not have to change the setting for this convention; the default ordering is the most common one used for the locale. But if you do need to implement a non-default ordering at your installation, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate (or create) the `dateInput` entry:

   ```
   locale:
   ```

```
{
...
dateInput: "DMY"
...
```

3. Change the value of `dateInput` to specify the date-ordering you want:

   `DMY` (Day–Month–Year)
   `MDY` (Month–Day–Year)
   `YMD` (Year–Month–Day)
   `YDM` (Year–Day–Month)

4. Save and close the file.

Changing the Decimal Separator

As installed, each locale provides a decimal separator—the symbol used to set off the decimal portion of a number in collection fields. The symbol is either a period (**.**) or a comma (**,**), whichever is most appropriate for the locale.

You should not have to change this value. But if you do need to use a non-default decimal separator at your installation, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`productDir`*`\common\`*`locale_name`*.
2. In the `locale` block, locate (or create) the `decimal` entry:

```
locale:
{
...
Decimal: "comma"
...
```

3. Change the value of `Decimal` from `comma` to `dot`, or from `dot` to `comma`, as appropriate.
4. Save and close the file.

Setting Up Synonym Search For a Locale

All Verity locales support the use of a thesaurus, or synonym list, for searching. In a synonym search, all occurrences of the search term plus any of its synonyms are returned (see "Synonym Search" on page 53).

To enable synonym search for a given locale, you need to implement a thesaurus containing the lists of synonyms. For some locales, Verity provides a basic thesaurus that you can use as-is or further customize; for other locales, you need to create your own thesaurus.

Only one thesaurus file is allowed per locale. If you implement a properly constructed thesaurus, give it the required name (`vdk30.syd`), and place it at the top level of your locale's directory, it will be used for synonym search.

For detailed instructions on creating and installing a custom thesaurus, see the *Verity Query Language and Topic Guide*.

Creating a Stop-Word File

As noted in "Stop Words" on page 54, a stop-word list is a list of words to ignore in searching (or in indexing). Verity provides for several kinds of stop-word files.

The file `vdk30.stp`, in the directory *`productDir`*`\common\`*`locale_name`*, contains locale-specific stop words to be used by the VQL free-text query parser during searching, and by the feature-extraction process during indexing (in conjunction with the file `style.fxs`). Most Verity locales include a default stop-word list.

`vdk30.stp` does not prevent words from getting into the word index; that job is the responsibility of the stop-word file `style.stp`. What `vdk30.stp` contains are words that should not be considered when the indexer extracts document features for creating automatic document summaries and clusters.

To implement a locale-specific stop-word file, take these steps:

1. Create a text file in the internal character set of the locale. The filename must be `vdk30.stp`.
2. Optionally add comment lines (each starting with #) at the top, naming the document and specifying its locale and character set, like this:

```
# Polish stoplist
# charset iso-8859-2
#
```

3. Enter each word into the stop list. Note these requirements
   - Enter only one word per line
   - Words are case-sensitive. You must list all case variants.
   - The order of the words is not important.

- Enter only literal words. Regular expressions are not supported.
- If you are creating a stop-word list for the multilanguage locale, note that the `vdk30.stp` file only lists stop words common to all supported languages. It must be in the UTF-8 character set.

  To specify stop words for a particular supported language in the multilanguage locale, use the `stopword.`*`ID`* file, located in the directory

  `productDir``/common/uni/`*`ID`*`/stopword.`*`ID`*

  where *`ID`* is the language ID, as listed in "Supported Language Codes" on page 135. The `stopword.`*`ID`* file must use the UTF-8 character set.

Words you might want to exclude from feature extraction (and therefore include in `vdk30.stp`) are proper names plus any words that would not make good topic or concept titles (single characters and common short words, for example).

4. Save and close the file.
5. Move the file to your locale's directory:

`productDir``\common\`*`locale_name`*

where *`productDir`* is the installation directory of the Verity component (such as `C:\Verity\K2`), and *`locale_name`* is the name of the directory (such as `polish`) containing the locale for which you are creating the stop-word file.

Note If you are creating a stop-list file for a locale (like `polish`) that has a default stop list provided by Verity, move the default stop-list file from the *`locale_name`* directory, or else rename it, before adding your new stop-list file. It is recommended that you do not permanently remove the default stop-list file.

For more information on `style.fxs` and `style.stp`, see the chapter on index tuning in the *Verity Collection Reference*.

Configuring Language Identification

By default, language identification occurs as a part of indexing in all locales (see "Language Identification" on page 44). The language-identification filter processes each incoming document and assigns a language code to it.

Note You perform basic configuration of the language-identification filter by editing the `style.uni` file for your collection. For instructions, see the discussion of the universal filter in the document filters chapter of the *Verity Collection Reference*.

Language identification can have a negative effect on indexing performance. The filter compares each document to a set of defining information for every supported and enabled language, then assigns the highest-scoring language to the document.

Note The language-identification filter does not have to compare a document to any language data if the document already contains unambiguous language-assignment information. For example, if an HTML document contains the following metatag:

```
<meta http-equiv="Content-Language" content="id">;
```

or

```
<html lang="id">
```

the language-identification filter uses that information directly, instead of analyzing the document content.

If you know that all documents you will analyze will be in a specific subset of the Verity-supported languages, you may be able to improve indexing performance by applying language identification to only those specific languages. Furthermore, if detection is not required for any of your documents, you can disable language identification altogether.

By default, the language-identification filter is enabled for a small subset of the available languages. You can adjust that set of languages as described next.

Adjusting the Set of Languages to Identify

The languages that the language-identification filter compares with incoming documents are listed in the file `langlist.cfg`, in the directory *`productDir`*`\common\langid`. That directory also contains the *language-data files*—the files containing the language-defining information—of all Verity-supported languages.

In addition, each locale uses its own `langlist.cfg` file to determine that locale's character sets and possible languages. The `langlist.cfg` file for an individual locale is available at *`productDir`*`\common\`*`locale_name`*. The `langlist.cfg` file for one locale may be different from those of other locales.

The directory *`productDir`*`\common\`*`uni`* does not contain a `langlist.cfg` file. For the uni locale, the system default at *`productDir`*`/common/langid/langlist.cfg` is used.

Following is an example of a `langlist.cfg` file:

```
da-1252.lm
de-1252.lm
en-1252.lm
es-1252.lm
fi-1252.lm
fr-1252.lm
it-1252.lm
ja-eucjp.lm
ja-sjis.lm
ko-ksc.lm
nl-1252.lm
nb-1252.lm
nn-1252.lm
pt-1252.lm
sv-1252.lm
zt-big5.lm
zh-gb.lm
da-utf8.lm
de-utf8.lm
es-utf8.lm
fi-utf8.lm
fr-utf8.lm
it-utf8.lm
nl-utf8.lm
nb-utf8.lm
nn-utf8.lm
pt-utf8.lm
sv-utf8.lm
```

Each entry in the list is the name of a language-data file in the `langid` directory. Each filename typically specifies the language code (see "Supported Language Codes" on page 135) and character set (see "Supported Source-Document Character Sets" on page 130) to which it applies.

IMPORTANT Do not modify the contents of any of the language-data files referenced in `langlist.cfg`.

The `langlist.cfg` file includes three keywords:

- `MINSIZE` defines the minimum size, in bytes, of document content. Any document whose content (excluding noisy data such as punctuation, numbers, and so on) is smaller than the value of `MINSIZE` will be considered to have an unknown language.
- `MINSCORE` defines the minimum score of a document. Any document whose detection score is less than the value of `MINSCORE` is assumed to have an unknown language.

  The format is:

  `MINSCORE: minimum_score`

  The *`minimum_score`* can be any value between 0 and 1. A value outside this range is treated as 0.
- `UNKNOWN` defines the language ID and character set to assign to any document whose language is unknown. Its format is as follows:

  `UNKNOWN: <language ID>:<charset>`

  For example,

  `UNKNOWN: en:1252`

  defines `en` as the language ID to assign to an unknown document and `1252` as the character set.

  If you disable this keyword, the language ID assigned to those documents will be 'un'.

To remove a language/character-set combination from consideration for language identification, simply remove its line from `langlist.cfg`. To add another language, add a line for it to `langlist.cfg`, like this:

1. In the same directory as `langlist.cfg`, open the file `langlist.all`. (`langlist.all` is a version of `langlist.cfg` that lists all languages supported for identification.)
2. From `langlist.all`, copy the line(s) for the languages you want identified and paste those lines into `langlist.cfg`:

```
de-1252.lm
de-850.lm
en-1252.lm
ja-eucjp.lm
ja-sjis.lm
```

In this example, all but German, English, and Japanese have been removed, and German (cp850 character set) has been added. Documents will now be compared against only German, English, and Japanese language-data files in order to make a language assignment.

3. Save and close `langlist.cfg`. (Do not make changes to `langlist.all`.)

Note If you want to enable language-identification for *all* supported languages, you can rename or save a copy of your original `langlist.cfg`, then save a copy of `langlist.all` as `langlist.cfg`.

Disabling Language Identification

If you know that language identification is unnecessary for indexing your collections, there is no need to incur its potential negative performance impact. For example, if all documents that you index will be in one language only, and the collections you create will be in the locale for that language, you can disable language identification completely.

IMPORTANT Do not disable language identification if you are using the multilanguage locale. The `uni` locale requires language identification, even if there is only one language to identify.

You can disable language identification by any of these methods:

- Delete the entire `langid` directory from the *`productDir`*`\common` directory.

  If the `langid` directory is missing, the language-identification filter assumes that no language identification is desired.

- Disable the language-identification filter itself:

  a. Open the version of the universal-filter configuration file (`style.uni`) that you will use to create the collection. (The original file is in the directory *`productDir`*`\common\style`.)

  b. Verify that the following line exists:

```
postformat: "flt_lang "
```

  c. If the line exists (and is uncommented), comment it out:

```
#postformat: "flt_lang "
```

  If the line doesn't exist, do nothing. If it is there but already commented, do nothing.

  d. Save and close the file.

- Empty or remove the `langlist.cfg` file, located at *`productDir`*`\common\langid`.
- Empty or remove the `langlist.cfg` file for a given locale, located at *`productDir`*`\common\`*`locale_name`*.

Specifying Locale and Character Set in Tools

This section contains suggestions and reminders for successfully creating and maintaining non-English collections and other types of indexes. For detailed information on any of these topics, see the books referenced in each section.

Note The locale (and internal character set) of a collection apply to the fields and data in the collection itself, not necessarily to the languages and character sets of any of the documents indexed by the collection. A collection in one locale can have index information on documents in several languages and character sets.

Specifying Locale and Character Set On the Command Line

To create or manipulate collections or other types of indexes, you can use a variety of command-line tools, including the following:

- `mkvdk`, `rcvdk`, and `rck2` to directly create and modify collections.
- `k2spider_srv`. To create a collection using the K2 Spider.
- `vspider`. To create a collection using the Verity Spider.
- `mkpi`. To create and modify parametric indexes.
- `mkre`. To create and modify recommendation indexes.
- `mktopics`, `mkprf`, and `mksyd` to create and manipulate topic sets, profile nets, and thesauruses, respectively.
- `rcadmin` (`indexvdkset` command) to register and attach collections and other indexes.

With any of the above command-line tools, you include the `-locale` or `-charmap` options (or arguments, for `rcadmin`) for the following purposes:

- `-locale` *`locale_name`*

  Assigns the locale *`locale_name`* to the collection being created (or, alternatively specifies the locale of the collection or topic set upon which the index or profile net you are creating or accessing is based). The value of *`locale_name`* must be the name of a locale for which you are licensed. Verity locale names are listed in "Verity Locales and Character Sets" on page 128.

  This option is not required if this collection (or the collection on which this index or profile net is based) is in the default locale. See "Default Locales and the Session Locale" on page 61.

- `-charmap` *`charset_name`*

  Specifies that the character set *`charset_name`* is to be used to display information from this collection, index, or profile net. *`charset_name`* must be the name of one of the supported character sets for the locale specified in *`locale_name`*. "Verity Locales and Character Sets" on page 128 lists the supported character sets for each Verity locale and indicates which one is the internal one.

  This option is not required if you want to display collection, index, or profile-net information in its locale's internal character set.

Specifying Locale and Character Set In the K2 Dashboard

When you create a collection though the K2 Dashboard, you need to fill in the **locale** field in the Add Collection wizard. The drop-down list displays all the licensed and installed Verity locales available to you; from the list, choose the locale you want for your index.

If you use the K2 Dashboard to try to view a collection in a locale whose character set is not the current Dashboard character set, you may need to change the Dashboard encoding setting to view the collection successfully. The encoding you choose must be one of the supported character sets of the collection's locale.

For more information on using Verity Spider, see Part III of *Verity K2 Indexers Guide*.

Setting BIF Character Encoding

If you are using a Bulk Insert File (BIF) to add, modify, or delete documents in a collection or to specify field values for the documents in a collection, note that the character set of the BIF itself must match the character set that the collection's locale uses.

For example, if the collection is in Russian and uses the character set KOI8-R, create your BIF in a word processor that uses KOI8-R encoding.

Note You can convert a file from one character encoding to another (within a given locale) by running the `codeconv` command-line tool. `codeconv` is installed with Verity, in the directory *`productDir`*`\`*`os_platform`*`\bin`. For instructions on using `codeconv`, see "Code Conversion Command-Line Tool" on page 140.

Specifying Character Set In Style Files

In setting up the style files for your non-English collection, note these locale and character-set issues.

- **`style.uni`**. This style file configures the universal document filter. For each recognized document type, `style.uni` specifies which individual document filter is to process the document and, optionally, what character set the document is to be converted to.

  If `style.uni` specifies no character set for that document type, the individual document filter is responsible for performing any required character-set conversion. You can therefore use `style.uni` to force a desired character conversion for a given file type, if appropriate.

  For more information on `style.uni`, see the chapter on document filters in the *Verity Collection Reference*.

- **`style.tde`**. This style file contains rules specifying how fields are to be extracted from processed documents when using the `mkvdk` command-line tool to create a collection. Using the `charmap` modifier in the `datamap` section of `style.tde`, you can optionally specify a character-set name if you want `mkvdk` to display the extracted fields in a character set other than the internal character set of the collection's locale.

  For more information on `style.tde`, see the chapter on searching documents by fields in the *Verity Collection Reference*.

5

Locale Issues for Applications

This chapter gives hints and suggestions for creating language-aware applications that can make maximum use of Verity's internationalization technology.

For detailed instructions on using the Verity K2 and VDK APIs in application development, see the Verity documentation referenced in this chapter.

This chapter includes the following sections:

Language Concerns in Verity Applications

This section contains notes and suggestions regarding implementing language-awareness in applications that use Verity technology.

Runtime Concerns

For VDK applications, locale, character set, and language are settings that can be defined on a per-session basis.

Session Locale and Character Set

Before making calls to the VDK API, your application first needs to create a VDK session. You can create a session in either of two ways:

- Use the `VdkSessionNew` function to create a new session from scratch. Calling this function requires you to fill out a complete `VdkSessionNewArgRec` structure and pass it to the function.
- Use the `VdkSessionSpawn` function to create a new session by spawning it from an existing one. The new session inherits the characteristics of its parent session, other than those you define in a `VdkSessionSpawnArgRec` structure that you pass to the function.

A VDK session always has its own defined session locale and character set. The two functions allow you to specify the locale and character set for the session, using these two structure members:

Member name	Type	Description
`internalLocale`	`VdkCText`	(`VdkSessionNew` only) The locale of the new session. Its value must be the name of one of the supported Verity locales listed in "Verity Locales and Character Sets" on page 128. If this value is null, the session uses the default locale (see "Default Locales and the Session Locale").
`charMapDriver`	`VdkCText`	(`VdkSessionNew` and `VdkSessionSpawn`) The name of the character set for the new session. Strings sent to VDK during this session must use this character set, and all text returned from VDK will be in this character set. This value must be one of the supported character sets for the session locale. If this value is null, the session uses the internal character set for its locale (in the case of `VdkSessionNew`) or the locale of its parent session (in the case of `VdkSessionSpawn`). "Verity Locales and Character Sets" on page 128 lists the supported character sets for each Verity locale and indicates which one is the internal character set.

During a VDK session, your application can obtain information about the session by calling the `VdkGetSessionInfo` function. The information you can obtain includes the session's locale and character set, specified in the `localeName` and `charMapDriver` members of the `VdkSessionGetArgRec` structure. `localeName` and `charMapDriver` are identical to the `internalLocale` and `charMapDriver` members just described for `VdkSessionNewArgRec`.

`VdkSessionNew`, `VdkSessionSpawn`, and `VdkGetSessionInfo` are described in the session suite chapter of the *Verity Developer's Kit Programming Reference*.

Resetting the Default Session Language

When the session locale is the `uni` locale, VDK supports the concept of default session language (see "Default Session Language" on page 62). Your application can specify a default session language in the locale specifier you pass to `VdkSessionNew` in the `VdkSessionNewArgRec` structure. Specify the language using a two-character language code following the locale name, as in `uni/fr`. See "Supported Language Codes" on page 135 for a list of the Verity-accepted language codes.

Locale and Character Set Concerns

This section notes programming situations in which locale and character-set assignments can be made.

Getting Locale Information for a Collection or Parametric Index

If your application supports parametric-index searches on collections that might not be in the session locale or character set, the application might need to be able to determine what locale and character set are used by those collections and by indexes accessing those collections.

In Java, you can use these calls:

- Use the methods `getLocale` and `getCharSet` of the class `com.verity.parametric.server.Collection` to obtain the collection locale.
- Use the methods `getLocale` and `getCharSet` of the class `ParametricIndex` to obtain the index locale.

For more information, see the discussion of the parametric selection Java API in the *Verity Parametric Developer Guide*. See also the `VParametric` JavaDocs.

Getting and Setting Client Locale for Parametric Searches

To support parametric-index searches on a collection with a non-default locale or character set, your application needs to be able to read, and possibly set, the client browser's character set.

In Java, you can use the methods `getLocale`, `setLocale`, `getCharSet`, and `setCharSet` of the class `VParametricSearch` to perform these tasks. For more information, see the discussion of the parametric selection Java API in the *Verity Parametric Developer Guide*. See also the `VParametric` JavaDocs.

Locale and Character-Set Conversion for Gateways

If you are writing a gateway driver to read data from a repository and pass it on to the Verity engine for indexing, note that your driver is expected to determine, if feasible, what the current locale is and—if necessary—perform any required character conversion.

For documents whose language and character set the gateway understands, the gateway is expected to pass text to Verity in the internal character set of the current locale. To accomplish that requirement, your driver can obtain the value of the `charSetName` member of the `VgwSessionNewArgRec` structure, then pass that value to Verity in I18N library's `V18NConvert` function, which determines the text's current character encoding and performs the conversion (if necessary).

For instructions on developing a gateway, see the *Verity Gateway Developer's Kit Programming Reference*.

Language-Specific Searching

Your application can allow the user to specify a specific language for stemmed searches.

Defining a Search Language

If your application uses the `uni` locale, it can tokenize and stem the user's query string according to the language that the user wants to search in. Your application can pre-define the session language (as noted in "Resetting the Default Session Language" on page 117), or it can add the VQL modifier `<lang/id>` to the query string, where *id* is the code that defines the language of the search.

For Chinese, Japanese and Korean-language searches in `uni`, you must use the `<lang/id>` modifier to receive search results.

If the application uses a query parser such as the Internet-style query parser, which doesn't take VQL operators or modifiers, the user's query is actually expanded according the rules defined in the configuration file *productDir*/common/*locale_name*/basic.iqp. A global modifier can be specified in a rule in that file.

Language Concerns for Fields and Zones

Unlike text search of a document's content, searching a document's fields does not involve language information. Applying the <lang/*id*> modifier has no effect on search results from fields.

In a document with zones, the Verity engine assumes that all zones are in the same language. Search results based on a given value of <lang/*id*> might be incorrect for zones in a different language than the one specified by *id*.

Localized Operators

VQL operators, modifiers, and keywords are not available in more than one localized version. If your application exposes those operators to end users in a multilanguage environment, they can appear in one language only. The uni locale uses English for VQL keywords; see "Localized Operators" on page 119 for more information.

Locale-Based Tokenization in a Custom Query Parser

By calling the VdkQParserNew function, your VDK application can specify that a non-default query parser is to be used for conducting searches. The specified parser can be one of the built-in non-default parsers (VdkParser_BoolPlus or VdkParser_FreeText), or it can be a custom parser that you have developed.

When calling VdkQParserNew, you can use the localeTok member of the VdkQParserNewArgRec structure to flag whether the new parser should tokenize query strings based on the rules of the current session locale. If you set the flag to off, strings are tokenized according to the default session locale.

VdkQParserNew is described in the query parser suite chapter of the *Verity Developer's Kit Programming Reference*.

Supporting Search in Multiple Languages

An application can be designed to simultaneously handle search requests in different languages from multiple users and apply those searches to multiple collections in different locales. One approach to handling this requirement is to define a separate VDK session for each user and language, in which the session locale and character set are tied

to the user's language and browser encoding. Another approach is for the application to use the `uni` locale and UTF-8 encoding for communication with all users and all collections.

Localization Concerns

This section discusses locale-specific formatting, layout, and performance issues that your application might need to address.

Date-Format Restrictions for uni Locale and Asian Locales

One of the field types supported for indexing is `Date`; a collection's document table can have any number of fields of type `Date`. For `Date` fields in a collection in a single-language locale, Verity supports both numeric formats (such as **11-11-2003**) and text formats in the locale's language (such as **11Septiembre 2003**).

For the `uni` locale, text-based dates might be expected to be in any number of languages, but Verity supports only a single language per locale. Therefore, by default, the `uni` locale supports only numeric formats and English text formats for `Date` fields.

For the Asian locales `simpcb`, `tradcb`, `japanb`, and `koreab`, the text formats have not been localized, so they too support only numeric or English date formats.

It is possible to localize the `uni` locale to display textual dates in any (single) language other than English. See "Messages and Operators for a Locale" on page 122 for more information.

Implications of UTF-8 Character Set for uni Locale

If your application is to function correctly with the multilanguage locale, it needs to allow for differences between the UTF-8 character set and other character sets.

In UTF-8, a single character can occupy up to 4 bytes. Furthermore, the length of a string cannot be calculated as a fixed multiple of the number of characters it contains. For string display and text highlighting in UTF-8 text, it is your application's responsibility to make sure that it is using the UTF-8 character set and that it is correctly calculating string sizes and highlighting boundaries.

Locale-Influenced Sorting of Search Results

In some situations, you may want your application to display search results using a sort order other than document score or document ID (the defaults). If you sort results based on a string value (such as name, geographic location, or color), note that the resultant order depends on the sort-order table for the locale of the collection being searched.

Performance Issues for uni and Asian Locales

Indexing with the Asian locales and the `uni` locale can be slower than with single-byte locales. For `japanb`, `koreab`, and `uni`, you might be able to improve performance by disabling stemming (see "Improving Performance" on page 97). For the `uni` locale, you might also get improvements by taking the steps described in "Improving uni Indexing Performance" on page 90.

Levels of Feature Support in the Multilanguage Locale

This section discusses application-development issues specific to the multilanguage locale.

Multilanguage Document Clustering

Verity clustering algorithms currently assume that all documents to be clustered are in the same language. With the `uni` locale, there is the potential of putting unrelated documents in different languages into the same cluster.

No NGram Support

The `uni` locale does not support creation of ngram indexes for Asian languages (Chinese, Japanese, Korean). An ngram index is a search accelerator for Typo and Wildcard searches. For more information, see the discussion of the `mkvdk` tool in the *Verity Collection Referencee*.

Extra Word Variants in Indexes

The VDK Assists suite (see the *Verity Developer's Kit Programming Reference*) provides application access to a collection's various indexes (word index, zone index, zone-attribute index, stem index, case index, Soundex index, knowledge-base assist, and fuzzy assist). In the `uni` locale, an individual word can appear in this set of lists in more than one way—as the word itself, as a subword, and as a word stem. It is the responsibility of your application to filter out unwanted duplicates, if they occur.

Don't search for *_nn* (uni locale)

The Unicode character code 0x80 is undefined. The multilanguage locale uses this character as part of a stem word. Users should not be permitted to use it together with two lower case ASCII letters as the ending letters of a search term. (This is a very rare case.)

Localized Error Messages and Operators

Part of localizing a K2 application involves translating text that the end-user might see. Translating your application's user-interface text is an important issue but is not discussed here; this section considers Verity-supplied text strings—VDK error messages and VQL operators.

Messages and Operators for a Locale

Verity supplies a translated set of user-visible error strings and VQL operators for most locales. The translated strings are in the files `vdk30.rsd` and `vdk30.rst`, in the directory *`productDir`*`\common\`*`locale_name`*.

- **Multilanguage Locale**. A single set of error messages and operator names must cover all supported languages. The current locale architecture does not allow for separate, language-specific message sets within a single locale. Therefore, as shipped, the `uni` locale uses English for all the messages and operators.
- **Advanced European Locales**. Some but not all error messages are translated. Therefore, if you are licensed for those locales, it is possible for users of your application to see some messages in the locale's language and others in English.
- **Asian Locales**. The error strings have not been localized. Therefore, as shipped, the Asian locales use English for all the messages and operators.
- **Legacy European Locales**. Some but not all error messages are translated. Therefore, if you are licensed for those locales, it is possible for users of your application to see some messages in the locale's language and others in English.

`vdk30.rsd` and `vdk30.rst` are compiled files; the majority of their content is not human-readable. They cannot be edited directly. If your localized Verity application requires customized or non-English messages and/or operators for a given locale, please contact Verity Professional Services to discuss creation of custom localized resource files.

Translating Topic Sets

If your application package ships with topic sets that you have generated with English names, you might need to translate the names, or you might need to regenerate the topic sets, after localization.

Topic sets are described in the *Verity Query Language and Topic Guide*.

Testing and Troubleshooting

Logging and Debugging

This section describes some of the logging and debugging tools you can use to decipher problems related to locales.

Set Up Logging of Asian or uni Locale Activity

If you are using the `uni` locale or the Asian locales, you can turn on logging of locale activity by adding the `-log` option to the `driver` statement in the locale's `loc00.lng` file, like this:

```
driver:  "locuni -simple_tokens -log"  "locuni"
```

With this setting, all locale API activity and results are recorded in a file whose name starts with `LocUn`, stored in the directory `/var/tmp` (Solaris), `/tmp` (Linux), or the directory specified by the `%TEMP%` system variable (Windows). Reviewing this log might help you to pinpoint problems with the text input to the tokenization process or with the generated tokens or stems.

Only the most recent 500 actions are kept in the log. It is therefore most useful for inspecting the last few actions prior to a system crash.

Use Standard Diagnostic Logging

You can also use Verity's standard diagnostic logging package to log locale-based activity for the uni locale or other locales. Diagnostic logging records all activity, not just locale-based actions, and stores everything in a single large file.

The diagnostic package is an internal Verity capability that is not normally available to customers. If you have a special need to use it, please contact Verity Technical Support.

Note You should use diagnostic logging only for problem isolation. When enabled, it has a significant impact on performance.

Troubleshooting Language Problems

This section lists some problematic symptoms that might occur when you test your K2 client application, and it gives suggestions for how you might narrow the problem to find the solution.

Searching Problems

- **All searches fail completely.**
  - Make sure the collection is not empty.

    Does a null search produce hits? If not, the indexing might have failed or the collection might not be attached/loaded correctly into your application.
  - Make sure that the document you expect to find was indexed.

    Try indexing the same document again, using a higher message verbosity. Inspect the output for problems.
- **Search fails for specific terms in specific documents.**
  - Does a literal, exact-word search (enclosed in double quotes) succeed?

    If yes, there may be a problem in the stemming:

    – Did the locale and/or language used for indexing match the language of the document? Inspect the VLANG and CHARSET fields in the collection's document table. For example, run `rcvdk` and specify the fields to display:

```
rcvdk
x
fields vdkvgwkey 50 VLANG 10 CHARSET 10
s
r
```

    Or, run the language-identification command-line tool on those documents to see whether the right language assignment is made:

```
langid doc_file_path
```

– Did search use the correct language for stemming the query term? (Was the wrong `<lang/id>` specification used in the query, or is the current default session language not what you expect?)

- Does a search term containing the exact cases and accents succeed?

  If yes, the locale configuration may be different from what you expected (for example, accent-sensitivity may be enabled).

- Does a wildcard search, or a wildcard search that looks for a shorter string, work better?

  If yes, the settings for tokenization or stemming may be different from what you expect. Is simple-tokens behavior enabled? Is the set of token delimiters what you expect? Is stemming disabled?

- Does search succeed only for plain ASCII search terms?

  If yes, the character-set assignment might be incorrect or the locale might be wrong for the document.

- Is your search term in the word index at all?

  For multibyte languages in particular, you might want to use `didump` to print out the contents of the word index, to verify whether your search term is in the index.

  If your search term is not in the index, but similar terms exist near where it would be, it may be that tokenization or stemming is not functioning as you expect it to.

Incorrect Display of Accented or Multibyte Characters

- Is it true for documents in all languages, or only specific ones?

  [`uni` locale] If true only for specific documents, language or character-set identification may have failed for those documents. Inspect the VLANG and CHARSET fields for those documents in the collection's document table. For example, run `rcvdk` and specify the fields to display:

```
rcvdk
x
fields vdkvgwkey 50 VLANG 10 CHARSET 10
s
r
```

  Or, run the language-identification command-line tool on those documents to see whether the right language assignment is made:

```
langid doc_file_path
```

- Do the standard Verity templates work and display characters correctly?

  If yes, the search application might be configured incorrectly. Verify whether your search application uses an output character set that is incompatible with the locale's internal character set. Compare your application templates with the standard Verity templates.

  If no, the indexing/collection configuration might be incorrect. Verify that the locale used for indexing was the correct locale for those documents. (You can inspect the file `*.abt` in the collection's `assists` directory to verify the locale used for indexing.) Also verify that the style files (such as `style.uni`) contain the correct character-set settings.

- Do characters in fields display incorrectly, but the document summaries are fine?

  If yes, field-generation may be specified incorrectly. If you are using a BIF to populate fields, verify that its character encoding matches the character set of the locale. (Note that you can convert the character set of a BIF using the `codeconv` command-line tool.)

  If you are not using a BIF, verify that the gateway and document filter(s) are working properly for documents that have this problem. Check for errors in the character-set settings in the style files (such as `style.uni` or `vgw`*`xxxx`*`.`*`yyy`*).

- Is the character encoding of your client-browser set correctly?

  The browser encoding must be compatible with the character set of the collection. (The collection's character set is the internal character set of its locale. The browser encoding must be set to one of the supported character sets for that locale.) If your returned Web page automatically sets the user's browser encoding, are you using the correct `HTTP-equiv` settings for `charset`?

Command-Line Tools Return Different Results

- *Is search successful wi*th *`rcvdk`* or *`rck2`*, *but not through your K2 client application?*

  If yes, a locale or character-set error may have occurred in loading the collection into the K2 process, or your search-application might be requesting the output in an incompatible character set.

- *Is search unsuccessful with `rcvdk`* or *`rck2`*?

  If yes, make sure the `-locale` option specifies the locale of the collection and the `-charmap` option specifies a character set compatible with that locale.

A

Locales, Character Sets and Languages

This appendix lists the Verity-supported languages, Verity locales, and character sets that can be used for indexing, searching, and viewing in a localized environment. For more information on the features and usage of Verity locales, see "Verity Locales" on page 59 and "Installing Verity Locales" on page 19.

This appendix includes the following sections:

Verity Locales and Character Sets

The table in this section lists the names of the Verity locale modules that you can install and use for creating collections and other indexes. The table also lists, for each locale,

- Its internal character set (the character set it uses to process and store all its data)
- The additional character sets that can be used to display the locale's information and source documents.
- How it is licensed and delivered:
  - *Built-in*. Provided with a default K2/VDK installation.
  - *Verity Locales*. Installed with the Verity Locales product.
  - *Verity Single-Language European Locales* (legacy). Installed with the Verity Single-Language European Locales product.

Locale and character set are used as options, function parameters, and structure members in many Verity tools and APIs. Note the following usage conventions:

- **locale option.** When specifying a locale in a Verity command option or function parameter, use the Verity locale name (column 1 in the table).

  For example, to specify Greek as the locale for a collection you are creating with the `mkvdk` tool, use the option `-locale greek`.

- **charmap option.** When specifying a character set in a Verity command option or function parameter, use one of the Verity-defined character-set names (column 3 or 4 in the table). You can specify any of the supported character sets for the locale.

  For example, when using the `rcvdk` tool to view contents of a collection in the `arabic` locale, if you want the output to use the MS-DOS character set, use the option `-charmap 1256`.

Note The character-set names listed here are the specific Verity names that you must supply for the `charmap` option or parameter. For example, to indicate the UTF-8 character set, the value of charmap must be `utf8`, not `UTF-8`. See the next section, "Supported Source-Document Character Sets," for common aliases and re-spellings of these and other character sets.

Table A-1 Verity locales and character sets

Verity locale	Language	Internal character set	Other supported character sets
Built-in:			
`uni`	*Advanced support:* English *Basic support:* Many languages[a]	**utf8**	
`englishv`	English (advanced support)	**1252**	437, 850, 8859
`english`	English (basic support)	**1252**	437, 850, 8859
Verity Locales:			
`uni`	*Advanced support:* Multiple languages[b] *Basic support:* Many languages[a]	**utf8**	
`bokmalv`	Norwegian	**1252**	437, 850, 8859
`danishv`	Danish	**1252**	437, 850, 8859
`dutchv`	Dutch	**1252**	437, 850, 8859
`finnishv`	Finnish	**1252**	437, 850, 8859
`frenchv`	French	**1252**	437, 850, 8859
`germanv`	German	**1252**	437, 850, 8859
`italianv`	Italian	**1252**	437, 850, 8859
`portugv`	Portuguese	**1252**	437, 850, 8859
`spanishv`	Spanish	**1252**	437, 850, 8859
`swedishv`	Swedish	**1252**	437, 850, 8859
`japanb`	Japanese	**sjis**	eucjp, iso2022_jp
`koreab`	Korean	**ksc**	
`simpcb`	Chinese (simplified)	**gb**	big5
`tradcb`	Chinese (traditional)	**big5**	gb
Verity Single-Language European Locales (legacy):			
`bokmalx`	Norwegian	**1252**	437, 850, 8859
`danishx`	Danish	**1252**	437, 850, 8859
`dutchx`	Dutch	**1252**	437, 850, 8859
`englishx`	English	**1252**	437, 850, 8859

Table A-1 Verity locales and character sets (continued)

Verity locale	Language	Internal character set	Other supported character sets
finnishx	Finnish	**1252**	437, 850, 8859
frenchx	French	**1252**	437, 850, 8859
germanx	German	**1252**	437, 850, 8859
italianx	Italian	**1252**	437, 850, 8859
nynorskx	Norwegian	**1252**	437, 850, 8859
portugx	Portuguese	**1252**	437, 850, 8859
spanishx	Spanish	**1252**	437, 850, 8859
swedishx	Swedish	**1252**	437, 850, 8859

a. See Table A-3 on page 136.
b. See Table 3-1 on page 64.

Supported Source-Document Character Sets

The table in this section lists the character encodings that Verity can read and convert when indexing source documents from a document repository. Verity converts text in any of these character sets into a locale's internal character set for processing and storage in a collection.

The character sets that Verity uses internally are listed in the previous section, "Verity Locales and Character Sets."

Table A-2 Supported source-document character sets

Encoding Name	Typical aliases and alternate spellings	Comment
1250	Cp1250, Windows-1250	Central and Eastern European (Windows)
1251	Cp1251, Windows-1251	Cyrillic (Windows)
1252	8859, Cp1252, Windows-1252	Western European (Windows)
1253	Cp1253, Windows-1253	Greek (Windows)

Table A-2 Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
1254	Cp1254, Windows-1254	Turkish (Windows)
1255	Cp1255, Windows-1255	Hebrew (Windows)
1256	Cp1256, Windows-1256	Arabic (Windows)
1257	Cp1257, Windows-1257	Baltic (Windows)
1258	Cp1258, Windows-1258	Vietnamese (Windows)
8859-1	latin1, Iso-8859-1, iso8859-1	Western European (ISO)
8859-2	latin2, Iso-8859-2, Iso8859-2	Central-Eastern European (ISO)
8859-3	latin3, Is-o8859-3, Iso8859-3	Turkish, Esperanto, Maltese (ISO)
8859-4	latin4, Iso-8859-4, Iso8859-4	Estonian, Latvian, Lithuanian (ISO)
8859-5	cyrillic, Iso-8859-5, Iso8859-5	Russian, Cyrillic European (ISO)
8859-6	arabic, Iso-8859-6, Iso8859-6	Arabic (ISO)
8859-7	greek, Iso-8859-7, Iso8859-7	Greek (ISO)
8859-8	iso-visual, iso-logical, hebrew, Iso-8859-8, Iso8859-8	Hebrew (ISO)
8859-9	Iso-8859-9, Iso8859-9	Turkish (ISO)
8859_14	Iso-8859-14, Iso8859-14	Celtic (ISO)
8859_15	Iso-8859-15, Iso8859-15	Revised Latin 1 (ISO)
DOS-720	cp720	Arabic (DOS)
big5	cp950, Big5, Windows 950, IBM 950	Chinese (traditional)

Table A-2 Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
Cns11643	Cns11643	Chinese (national code for Taiwan)
cp037	Cp037	(EBCDIC code page 037)
cp10000	cp10000	(Microsoft Macintosh Roman)
cp1006	Cp1006	Urdu (Pakistan) (IBM AIX)
cp1026	Cp1026	(EBCDIC code page 1026)
cp424	Cp424	Hebrew (EBCDIC)
cp437	Cp437, MSDOS 437	Latin US (DOS)
cp500	Cp500	(EBCDIC code page 500)
cp737	Cp737, IBM 737	Greek (DOS)
cp775	Cp775, IBM 775	Baltic (DOS)
cp850	Cp850, IBM 850	Latin 1 (DOS)
cp851		Greek (DOS)
cp852	Cp852, IBM 852	Eastern European/Latin 2 (DOS)
cp855	Cp855, IBM 855	Cyrillic (DOS)
cp856	Cp856, IBM 856	Hebrew (IBM PC–old)
cp857	Cp857, IBM 857	Turkish (DOS)
cp860	Cp860, IBM 860	Portuguese (DOS)

Table A-2 Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
cp861	Cp861, IBM 861	Icelandic (DOS)
cp862	Cp862, IBM 862	Hebrew (DOS)
cp863	Cp863, IBM 863	Canadic (DOS)
cp864	Cp864, IBM 864	Arabic (DOS)
cp865	Cp865, IBM 865	Nordic (DOS)
cp866	Cp866, IBM 866	Cyrillic 2 (DOS)
cp869	Cp869, IBM 869	Greek 2 (DOS)
cp874	tis620, Tis-620, Cp874, IBM 874	Thai
cp875	Cp875	(EBCDIC code page 875)
euc-tw		Chinese (traditional) (euc encoding of CNS 1643-1992)
euc-cn		Chinese (simplified) (euc encoding of GB 2312-80)
euc-jp	Euc_jp	Japanese
Euc-ksc	euc-kr, Euc_kr, Windows-949	Korean
gb	euc-gb, Gb, Gbk, Cp936,Windows-936, GB2312, Gb2312-80	Chinese (simplified)
gb12345		Chinese (traditional) (variant of GB2312)
GB18030		Chinese 4-byte character set
x-iscii-as		ISCII Assamese
x-iscii-be		ISCII Bengali
x-iscii-de		ISCII Devanagari

Table A-2 Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
x-iscii-gu		ISCII Gujarathi
x-iscii-ka		ISCII Kannada
x-iscii-ma		ISCII Malayalam
x-iscii-or		ISCII Oriya
x-iscii-pa		ISCII Panjabi
x-iscii-ta		ISCII Tamil
x-iscii-te		ISCII Telugu
iso-2022-cn	Iso2022cn, Iso-2022-cn	Chinese (7-bit) (GB2312 pus CNS 11643) This encoding is not supported in the langlist.cfg file for character set detection or for XML. However, it is supported in plain text and HTML if metadata indicates that `iso-2022-jp` is fully supported.
iso-2022-jp	Iso2022jp, Iso-2022-jp	Japanese (7-bit)
iso-2022-kr	Iso2022kr, Iso-2022-kr	Korean (7-bit) This encoding is not supported in the langlist.cfg file for character set detection or for XML. However, it is supported in plain text and HTML if metadata indicates that `iso-2022-jp` is fully supported.
ISO 8859-10	iso_8859-10, iso8859-10	Nordic
ISO 8859-13	iso_8859-13, iso8859-13	Baltic
koi8r	cp878, Koi8_r, Koi-8r	Russian
ksc	cp949, cp1363, IBM 1363	Korean
Shift-jis	Sjis, cp932, shift-jis, sjis, Windows-932	Japanese
tis620	620	Thai

Table A-2 Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
Unicode	Unicode, UTF-16, iso-10646, Utf-16, utf-16be, utf-16le, UnicodeBig, UnicodeBigUnmarked, UnicodeLittle / UnicodeLittleUnmarked /Utf-16	(All languages)
Utf8	Utf8, 65001, Utf-8, utf8	(All languages)
VISCII		Vietnamese

Supported Language Codes

The table in this section lists the ISO 639 and ISO 639-1 two-character and three-character codes for the languages supported by the Verity multilanguage locale (`uni`) and the Verity language-identification command-line tool.

- In a bulk import file (BIF) used to create or modify a collection in the multilanguage locale, the value of the language specification field (`VLANG`) must be a language code from this table.

  Note When a document specifies a user-defined name for its language, rather than one of the standard codes, that name is preserved in the `VLANG` field. It is not mapped to the standard two-letter code.

  A user-defined name can be any string, language, abbreviation, or spelling. These original names are useful for diagnostic purposes. When the user-defined `VLANG` value cannot be determined by the Verity multilanguage locale (`uni`), it is treated as 'un' (unknown).

  For more information on BIF format, see the *Verity Indexers Reference Guide*.

- Input to, and output from, the Verity language-identification command-line tool specifies language in terms of the language code.

  For more information on the language-identification command-line tool, see "The Language ID Command Tool" on page 157.

- When searching for text in the multilanguage locale, the client can specify which language rules to apply to the search by using a language code in the `<lang/id>` query operator (for example, `<lang/de>` for German).

  Note Only the codes flagged in column 3 of the table below are accepted in `<lang/id>`.

  For more information on this and other query operators, see the *Verity Query Language and Topic Guide*.

- When specifying the default session language for a VDK session using the `uni` locale, the caller of `VdkSessionNew` supplies a locale identifier of the form `uni/id`, where *id* is the language code.

  Note Only the codes flagged in column 3 of the table below are accepted in `uni/id`.

  For more information on the default session language, see "Resetting the Default Session Language" on page 117.

Table A-3 Verity-supported language codes

Language	Code	OK for <lang/*id*> and uni/*id*?
Afrikaans	af	
Albanian	sq	
Arabic	ar	Yes
Armenian	hy	
Basque	eu	
Belarusian	be	
Bengali	bn	
Breton	br	
Bulgarian	bg	
Catalan	ca	
Cherokee	chr	
Chinese (simplified)	zh	Yes
Chinese (traditional)	zt	Yes
Croatian	hr	

Table A-3 Verity-supported language codes (continued)

Language	Code	OK for <lang/*id*> and uni/*id*?
Czech	cs	Yes
Danish	da	Yes
Dhivehi (Thanna)	div	
Dutch	nl	Yes
English	en	Yes
Esperanto	eo	
Estonian	et	
Ethiopic	gez	
Faroese	fo	
Farsi	fa	
Finnish	fi	Yes
French	fr	Yes
Frisian	fy	
Gaelic	ga	
Galician	gl	
Georgian	ka	
German	de	Yes
Greek	el	Yes
Greenlandic	kl	
Gujarati	gu	
Hausa	ha	
Hebrew	he	
Hindi	hi	
Hungarian	hu	Yes
Icelandic	is	
Indonesian	id	
IsiNdebele	nde	
IsiXhosa	xh	
IsiZulu	zu	

Table A-3 Verity-supported language codes (continued)

Language	Code	OK for <lang/*id*> and uni/*id*?
Italian	it	Yes
Japanese	ja	Yes
Kannada	kn	
Khmer	km	
Korean	ko	Yes
Kurdish	ku	
Lao	lo	
Latvian	lv	
Lithuanian	lt	
Luxembourgish	lb	
Macedonian	mk	
Malay	ms	
Malayalam	ml	
Maltese	mt	
Mongolian	mm	
Myanmar	bms	
Nepali	ne	
Norwegian (Bokmal)	nb	Yes
Norwegian (Nynorsk)	nn	
Oriya	or	
Panjabi (Gurmukhi)	pa	
Pashto	ps	
Philippine (Tagalog, Hanunoo, Buhid, Tagbanwa)	phi	
Polish	pl	Yes
Portuguese	pt	Yes
Rhaeto-Romance	rm	
Romanian	ro	
Russian	ru	Yes
Sami	se	

Table A-3 Verity-supported language codes (continued)

Language	Code	OK for <lang/*id*> and uni/*id*?
Serbian	sr	
Sesotho	st	
SesothoSaLeboa	stl	
SeTswana	tn	
Sinhala	si	
SiSwati	ss	
Slovak	sk	
Slovenian	sl	
Somalian	so	
Spanish	es	Yes
Swahili	sw	
Swedish	sv	Yes
Syriac	syr	
Tamil	ta	
Telugu	te	
Thai	th	
Tibetan	bo	
TsiVenda	ven	
Turkish	tr	
Ukranian	uk	
Urdu	ur	
Vietnamese	vi	
Welsh	cy	
XiTsonga	ts	
Yiddish	yi	
(All others)	un	

Code Conversion Command-Line Tool

Verity requires a fixed character set for BIF, style, and OTL files used by a locale. However, organizations often create and save documents using many different character sets. Verity provides the Code Conversion command-line command-line tool to help you convert files to a single character set. You identify a source file, its character set, and the character set you want to convert to. The command-line tool produces a converted document.

Running codeconv

To run the Code Conversion command-line tool, use the following command line:

`codeconv` *`src_charset src_file tgt_charset tgt_file [Unicode_endian]`*

where

- *`src_charset`* is the character set of the source file. Can be any encoding name or alias listed in "Supported Source-Document Character Sets" on page 130.
- *`src_file`* is the pathname to the file whose character set you want to convert.
- *`tgt_charset`* is the character set you want to convert to. Can be any encoding name or alias listed in "Supported Source-Document Character Sets" on page 130, as long as it is compatible with (applicable to the same language as) the character set specified in *`src_charset`*.
- *`tgt_file`* is the pathname to the file that is to hold the converted results.
- *`Unicode_Endian`* (optional) specifies, for 16-bit Unicode character sets, whether the storage order is little-endian (`0`) or big-endian (`1`). The default value is `0`.

The source and target character sets must be in the same language. For example, if the source character set is ISO Russian, the target character set cannot be Japanese.

A symbol that exists in one encoding standard might not exist in another. In this case, the command-line tool displays a question mark (?) in the converted file. For example, if a source document contained the Microsoft encoding for the Euro currency symbol, and the target character set did not provide an encoding for this symbol, the target file would show a question mark for each occurrence of the Euro symbol.

Example

In the following command-line example, the command-line tool is converting the file `russian.txt`, which uses character set 1251, to the character set UTF-8. The resulting file is named `target.txt`:

```
codeconv 1251 russian.txt utf8 target.txt
```

Supported Character Sets

The Code Conversion command-line tool supports the character sets listed in "Supported Source-Document Character Sets" on page 130.

Limitations

The source and target files must be textual, such as `.txt`, `.htm`, or `.xml` files. The command-line tool cannot convert files in proprietary formats such as Word, WordPerfect, Excel, or PowerPoint.

Conversion from GB to BIG5 is not supported.

B

Tokenization Delimiters

This appendix lists the tokenization delimiters applied by default when simple-tokens behavior is enabled. Individual symbols in this table can be removed from the list of delimiters and/or made searchable as symbols. See "Tokenization" on page 80 (for the multilanguage locale), "Tokenization" on page 91 (for single-language locales), "Tokenization" on page 95 (for Asian locales), and "Tokenization" on page 98 (for legacy European locales).

Table B-1 Available tokenization delimiters and symbols

Character Code		Description
1252	Unicode	
21	U+0021	EXCLAMATION MARK
22	U+0022	QUOTATION MARK
23	U+0023	NUMBER SIGN
24	U+0024	DOLLAR SIGN
25	U+0025	PERCENT SIGN
26	U+0026	AMPERSAND
27	U+0027	APOSTROPHE
28	U+0028	LEFT PARENTHESIS
29	U+0029	RIGHT PARENTHESIS
2A	U+002A	ASTERISK
2B	U+002B	PLUS SIGN
2C	U+002C	COMMA
2D	U+002D	HYPHEN-MINUS

Table B-1 Available tokenization delimiters and symbols (continued)

Character Code		Description
1252	Unicode	
2E	U+002E	FULL STOP
2F	U+002F	SOLIDUS
3A	U+003A	COLON
3B	U+003B	SEMICOLON
3C	U+003C	LESS-THAN SIGN
3D	U+003D	EQUALS SIGN
3E	U+003E	GREATER-THAN SIGN
3F	U+003F	QUESTION MARK
40	U+0040	COMMERCIAL AT
5B	U+005B	LEFT SQUARE BRACKET
5C	U+005C	REVERSE SOLIDUS
5D	U+005D	RIGHT SQUARE BRACKET
5E	U+005E	CIRCUMFLEX ACCENT
5F	U+005F	LOW LINE
60	U+0060	GRAVE ACCENT
7B	U+007B	LEFT CURLY BRACKET
7C	U+007C	VERTICAL LINE
7D	U+007D	RIGHT CURLY BRACKET
7E	U+007E	TILDE
80	U+20AC	EURO SIGN
82	U+201A	SINGLE LOW-9 QUOTATION MARK
84	U+201E	DOUBLE LOW-9 QUOTATION MARK
85	U+2026	HORIZONTAL ELLIPSIS
86	U+2020	DAGGER
87	U+2021	DOUBLE DAGGER
88	U+02C6	MODIFIER LETTER CIRCUMFLEX ACCENT
89	U+2030	PER MILLE SIGN
8B	U+2039	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
91	U+2018	LEFT SINGLE QUOTATION MARK

Table B-1 Available tokenization delimiters and symbols (continued)

Character Code		Description
1252	Unicode	
92	U+2019	RIGHT SINGLE QUOTATION MARK
93	U+201C	LEFT DOUBLE QUOTATION MARK
94	U+201D	RIGHT DOUBLE QUOTATION MARK
95	U+2022	BULLET
96	U+2013	EN DASH
97	U+2014	EM DASH
98	U+02DC	SMALL TILDE
99	U+2122	TRADE MARK SIGN
9B	U+203A	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
A1	U+00A1	INVERTED EXCLAMATION MARK
A2	U+00A2	CENT SIGN
A3	U+00A3	POUND SIGN
A4	U+00A4	CURRENCY SIGN
A5	U+00A5	YEN SIGN
A6	U+00A6	BROKEN BAR
A7	U+00A7	SECTION SIGN
A8	U+00A8	DIAERESIS
A9	U+00A9	COPYRIGHT SIGN
AA	U+00AA	FEMININE ORDINAL INDICATOR
AB	U+00AB	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
AC	U+00AC	NOT SIGN
AD	U+00AD	SOFT HYPHEN
AE	U+00AE	REGISTERED SIGN
AF	U+00AF	MACRON
B0	U+00B0	DEGREE SIGN
B1	U+00B1	PLUS-MINUS SIGN
B2	U+00B2	SUPERSCRIPT TWO
B3	U+00B3	SUPERSCRIPT THREE
B4	U+00B4	ACUTE ACCENT

Table B-1 Available tokenization delimiters and symbols (continued)

Character Code		Description
1252	Unicode	
B5	U+00B5	MICRO SIGN
B6	U+00B6	PILCROW SIGN
B7	U+00B7	MIDDLE DOT
B8	U+00B8	CEDILLA
B9	U+00B9	SUPERSCRIPT ONE
BA	U+00BA	MASCULINE ORDINAL INDICATOR
BB	U+00BB	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
BC	U+00BC	VULGAR FRACTION ONE QUARTER
BD	U+00BD	VULGAR FRACTION ONE HALF
BE	U+00BE	VULGAR FRACTION THREE QUARTERS
BF	U+00BF	INVERTED QUESTION MARK

C

Customizing Language Dictionaries

This appendix describes how to create customized language dictionaries for controlling stemming, decomposition, and in some cases part-of-speech extraction, in the uni locale, the englishv locale, and the single-language Asian locales.

This appendix includes these sections:

Custom Dictionaries for Asian Languages

For the languages Japanese, Korean, Simplified Chinese, and Traditional Chinese in the the multilanguage locale, and for the Asian locales `japanb`, `koreab`, `simpcb`, and `tradcb`, you can create a custom file, called a user dictionary, into which you can place words that you want decomposed in a non-standard manner. For example, you might want to create a user dictionary to hold proper names, industry-specific terms, or words of foreign origin. Or, you might want to prevent trademarked terms or company names from being decomposed into subwords at all.

Creating the User Dictionary

Your user dictionary must be a text file in the following format:

- File encoding must be UTF-8.

- Comment lines must begin with a pound sign (#).
- Each dictionary entry must be on a separate line. Each line must end with a carriage return.
- Blank lines are permitted.

On each line, you specify how the term is to be decomposed by following it with a tab (U+0009) followed by a *decomposition pattern*. The decomposition pattern consists of a string of digits, each one representing the number of characters (up to a maximum of 9) in the respective component. For example, the entry

言語処理　22

specifies that the term should be decomposed into two two-character components:

言語

処理

Note that the sum of the digits in the pattern must match the total number of characters in the term. For example,

言語処理　23

is invalid because the term has 4 characters while the pattern is for a 5-character string.

You can also use the dictionary to prevent decomposition of a term that is normally decomposed during indexing. To do so, follow the term's entry in the dictionary with a decomposition pattern that is either **0** (zero) or a single digit equal to the full length of the entry. For example:

未定義　3

貴乃花　0

情報提供　4

(The nonzero-digit alternative works only for terms with nine or fewer characters).

Note Placing a term with a specific decomposition pattern into the user dictionary does not guarantee that searches on its components will always return the term itself. At search time, the components might be tokenized differently, based on context.

For example, if the entry is `ABCD 22` (where `A`,`B`, `C`, and `D` are individual characters), a search for `AB` will not return `ABCD` if `AB` is tokenized at search time as `A` and `B` (separate characters) instead of `AB`.

Installing the User Dictionary (Multilanguage Locale)

When your user dictionary is complete, install it into the multilanguage locale this way:

1. Give it any name—for example, `user_dict_1.utf8`.
2. Store it in the directory *`productDir`*`/common/basdata/`*`language`*`/dicts`, where language is `ja`, `ko`, `zh`, or `zt`.
3. Open the language options file for this language. For Japanese, for example, open the file `jla-options.xml`, in the directory *`productDir`*`/common/basdata/ja`.
4. To make your dictionary available, add a `<DictionaryPath>` subelement to the `<DictionaryPaths>` element of the language options file, like this:

```
...
<DictionaryPaths>
   ...
   <DictionaryPath><env name="root"/>/../../ja/dicts/user_dict_1.utf8<DictionaryPath>
   ...

<DictionaryPaths>
```

   If you have created multiple user dictionaries, make them available by creating a separate `<DictionaryPath>` entry for each one.

5. Specify that your dictionary is to be used by setting this value in the file:

   ```
   FavorUserDictionary="true"
   ```

6. Save and close the language options file.

Installing the User Dictionary (Asian Locales)

When your user dictionary is complete, install it into an Asian locale this way:

1. Give it any name.
2. Store it in the locale's directory—for example, *`productDir`*`\common\japanb`.
3. Open the locale's `loc00.lng file` and add a `user_dictionaries` option to the driver entry, like this:

```
driver:  "locbasis -simple_tokens & -user_dictionaries dictName" "loc"
```

   where *`dictName`* is the filename of the dictionary.

   If you have created multiple user dictionaries, add them to the locale by following the `-user_dictionaries` option with a comma-separated list of dictionary filenames:

```
-user_dictionaries dictName1,dictName2,dictName3,...
```

There must be no spaces in the filenames or between them. You can add up to 128 user dictionaries, as long as the entire `driver:` statement is not over 2048 characters long.

4. Save and close the file.

Note Verity provides a small sample user dictionary (`user_dict_1.utf8`) with the `japanb` locale.

Using Multiple User Dictionaries

If you have a large number of terms whose decomposition you need to customize, you can create multiple user dictionaries and install them as just described. You might want to divide the entries so that each dictionary holds an alphabetically sorted range, or an industry-specific set of terms, or a certain set of proper names.

Custom Dictionaries for Non-Asian Languages

For non-Asian languages in the multilanguage locale, and for the single-language locales, you can use the `_mdic` command-line tool to add additional words to language dictionaries. During indexing, the locales use these language dictionaries to control stemming, word decomposition, and noun extraction (part-of-speech identification).

How to Modify a Language Dictionary

The dictionary-customization process consists of creating a list of words, then merging that list with an existing language dictionary to create a new dictionary.

You run the `_mdic` tool to generate a language-specific compiled dictionary file of type `.mdic` (such as `german.mdic`). The language dictionary produced by `_mdic` is actually a merger of an existing compiled dictionary of type `.bmdic` (such as `german.bmdic`) and a user-defined dictionary source file of type `.txt` (such as `newwords.txt`).

Take these steps to modify your dictionary:

1. Locate the standard language dictionary (*languageDict*.bmdic) to which you want to add more terms. Save a copy of it before proceeding.

   In K2, the standard language dictionaries for a given language in the uni (multilanguage) locale are in the directory *installDir*/k2/common/terdata/*language*, where *language* is an abbreviated language name (such as en for English). for example, the standard English-language dictionaries provided with K2 are the following:

emd08_mstem.bmdic	Multistemming (1252 charset)
emd08_mstem.utf8.bmdic	Multistemming (utf8 charset)
emd08_mstem_norm.bmdic	Multistemming + normalization (1252 charset)
emd08_mstem_norm.utf8.bmdic	Multistemming + normalization (utf8 charset)
emd08_stem.bmdic	Simple stemming (1252 charset)
emd08_stem.utf8.bmdic	Simple stemming (utf8 charset)

   (The utf8 dictionaries are for use with the uni locale; the 1252 dictionaries are for use with the englishv locale. Different levels are stemming are provided to allow for trading stemming accuracy for indexing performance.)

2. If _mdic has been used previously to create a custom dictionary, locate that dictionary and save off a copy of it before you run _mdic. (You will be replacing it.)

   Custom dictionaries (*customDict*.mdic) are stored along with the standard dictionaries, in *installDir*/k2/common/terdata/*language*. For example, the custom English-language dictionaries provided with K2 are the following:

emd08_mstem.fast.mdic	Multistemming (1252 charset)
emd08_mstem.fast.utf8.mdic	Multistemming (utf8 charset)
emd08_mstem_norm.fast.mdic	Multistemming + normalization (1252 charset)
emd08_mstem_norm.fast.utf8.mdic	Multistemming + normalization (utf8 charset)
emd08_stem.fast.mdic	Simple stemming (1252 charset)
emd08_stem.fast.utf8.mdic	Simple stemming (utf8 charset)

   (If you are creating a custom dictionary for English and intend to replace one of these, be sure to save a copy first.)

3. Create a dictionary source file (*words*.txt) as described in "Creating a Dictionary Source File" on page 154. The file is a simple list of text lines, each of which defines a term to add to the dictionary.

4. If appropriate for your dictionary's language, create a predefined compounds file (*compounds*.txt) with explicit decomposition guidelines. See "Creating a Predefined Compounds File" on page 156.
5. Use the -minus option on _mdic if you want your source-file entries (including part-of-speech tags or other meta-information) to replace the equivalent entry in the language dictionary.
6. Execute the _mdic command-line tool to compile the source file, the predefined compounds file, and a specified standard language dictionary into a custom dictionary.

   For syntax details, see "_mdic Command-Line Syntax" (next). Here is an example command:

   ```
   _mdic -compile_user emd08_mstem.bmdic multistem_source.txt
   -tmpdir /usr/home/tmp -compress >emd08_mstem.fast.mdic
   ```

7. Place the output file *customDict*.mdic into the directory *installDir*/k2/common/terdata/*language*, replacing any previous dictionary of the same name.
8. To make the locale (uni or Advanced European) use the custom dictionary, do this:
   a. Open the locale's uni.cfg file, in *installDir*/k2/common/*locale*, where *locale* is the locale name (for example, uni, englishv, or frenchv).
   b. Locate the language mapping list and find the entry for your dictionary's language. For English in the uni locale, the entry looks like this:

```
#English

  language: en "en-us" "en-can"
{
    driver: "unitera -lang en -stkzo english.utf8.stkzo
                     -mdic emd08_mstem_norm.fast.utf8.mdic
                     -concepts english_small.concepts" "unitera"

}
```

   c. If the driver: line already contains the -mdic option, make sure that the option specifies the custom dictionary you just created. If not, add the option and specify your dictionary.

_mdic Command-Line Syntax

You run the `_mdic` tool from the UNIX command line. The tool is located in the directory *`platformDir`*`/bin`, where *`platformDir`* is the platform-specific directory within the K2 installation (for example, `usr/verity/k2_6/k2/_ssol26`).

The command syntax for `_mdic` is this:

```
_mdic -compile_user languageDict.bmdic words.txt [-tmpdir
tmpDirName] [-compress] [-minus] [-predef_compounds compounds.txt]
>customDict.mdic
```

where

- `-compile_user` is the flag that specifies that dictionary compilation is to occur.
- *`languageDict`*`.bmdic` specifies the name of a default, language-standard binary dictionary file (such as `german.bmdic`). In K2, binary language dictionaries have the file extension `.bmdic` and are stored in the directory *`installDir`*`/k2/common/terdata/`*`language`*.

  The `uni` locale includes pre-defined language dictionaries for English only, in the directory *`installDir`*`/k2/common/terdata/en`.
- *`words`*`.txt` is the name of a user-defined dictionary source file, containing terms to be added to the dictionary. See "Creating a Dictionary Source File" on page 154 for a description of its format and usage.
- `-tmpdir` *`tempDirName`* is an optional flag to specify the name of a temporary directory for writing files while processing.
- `-compress` is an optional flag to specify that the resulting language dictionary should be compressed. If `-compress` is specified, the resulting memory usage will be smaller and the speed for processing new words will be faster; however, the loading time will be longer.
- `-minus` is an optional flag to specify that the entries in *`words`*`.txt` should replace (override) the equivalent entries in the language dictionary.

  Both stemming and additional information—that is, both the *`ROOT`* and *`INFO`* portions of the entry, as described in "Creating a Dictionary Source File" on page 154—are overridden by using this option.
- `-predef_compounds` *`compounds`*`.txt` specifies the name of a user-defined file that contains predefined compound words to be added to the language dictionary. See "Creating a Predefined Compounds File" on page 156 for a description of its format and usage.

- *customDict*.mdic specifies the name of the output compiled dictionary file, such as german.mdic. The output dictionary contains terms from both the language-standard (.bmdc) binary dictionary and the source (.txt) file inputs.

  If you specify no output file, results are written to standard output. If you specify a filename only, the file is written to the current directory. If you specify a path, it is interpreted as relative to the current directory.

  For use, the output dictionary needs to be stored in *installDir*/k2/common/terdata/*language*, replacing the previous dictionary of the same name.

This is an example _mdic command:

```
_mdic -compile_user german.bmdic newwords.txt -tmpdir /usr/home/
tmp -compress -predef_compounds gcompounds.txt >german.mdic
```

Creating a Dictionary Source File

The _mdic tool operates on a source file that contains the terms you wish to add to a pre-existing user dictionary. Create the source file as a plain-text file named *words*.txt, where *words* can be any valid filename string.

You add each term to the file as a single text line. Lines should be separated by a single newline character. You can add simple words, compound words, and predefined compounds.

To add words to the dictionary, create a source file consisting of text lines with the following format

KEY,*ROOT*:*INFO*

where

- *KEY* is the inflected word to be analyzed.
- *ROOT* is the root form (or stem) for the word specified by *KEY*. A comma (,) must be present between *KEY* and *ROOT*.
- *INFO* is an arbitrary string representing additional information for the analysis, such as part-of-speech tags. A colon (:) must be present between *ROOT* and *INFO*.

Note The file does not need to be sorted. All lines not conforming to the correct syntax will be ignored.

For example, three lines of the file might look like this:

```
booked,book:w
books,book:w
compilations,compilation:w
```

The first two lines specify that `book` is the root form of the words `booked` and `books`, while the third line specifies that `compilation` is the root form of `compilations`. For all three words, the information section uses `w`, which means that the intended use is simple stemming only (associating a single stem with any inflected word).

For cases in which part-of-speech information is also desired, you can add part-of-speech tags to the *`INFO`* string. For a list of the part-of-speech tags, refer to the Morphological Dictionary for your language.

For instance, lines in a sample file might look as follows:

```
booked,book:Vpp
books,book:Npl
compilations,compilation:Npl
```

Note that multiple lines can have the same *`KEY`*, to account for multiple word meanings. In the following file:

```
booked,book:Vpp
booked,book:Vpt
books,book:Npl
books,book:V3s
compilations,compilation:Npl
```

the word `books` can have both the verb form and the plural noun form, and the word `booked` can be either the past tense or the past participial form of the verb.

The case used in the source file should be standard usage. In the previous examples, all common English words are in all lower case. By contrast, in a language like German, nouns would start with upper-case letters. For example:

```
Gesellschaften,Gesellschaft:w
trinkest,trinken:w
```

Creating a Predefined Compounds File

For Germanic languages, it may be useful to break up compound words into their constituent parts, even if they are in the standard language dictionary and could be stemmed as one word. This is useful if you want to allow users to search for both `Krebs` (cancer) and `Nieren` (kidneys), in addition to `Nierenkrebs` (all of which are in a standard German language dictionary).

The compounds for which you provide explicit decomposition information are called predefined compounds. A large number of predefined compounds are included in standard language dictionaries for Germanic languages. However, you may want to add your own predefined compounds to your dictionary. In that case, create a compounds source file (*`compounds`*`.txt`) containing lines like this:

```
hauptbahnhof,haupt:bahnhof
```

Each entry should be on its own line. On each line, write the compound form, followed by a comma, followed by the broken up compound. The compound's parts should be separated by colons. Compounds of several parts are acceptable, like this:

```
ausfahrhubbegrenzungseinheit,ausfahr:hub:begrenzungs:einheit
```

Add to the source file any compounds that are already in the standard language dictionary as single entries that otherwise would not be decomposed.

Note There is no need to add predefined compounds to this file if their parts are already in the standard dictionary or in your *`words`*`.txt` file; decomposition will happen automatically. One exception, however, is that if a compound has one or more parts that are fewer than four characters long, you must add them to both *`words`*`.txt` and to *`compounds`*`.txt`.

D

The Language ID Command Tool

Verity collections created with the multilanguage locale can include documents in more than one language. During indexing, the Verity language-identification filter (see "Configuring Language Identification" on page 106) detects the language of each incoming document to ensure that its content is indexed according to the rules of that language.

The language-identification command-line tool (`langid`) is a Verity tool that is built on that language-detection capability. It detects the language and encoding of a document that you provide. You can use it to identify the language and character set of a document before you index it into a collection, or you can perform other language-dependent processes based on its output.

This appendix includes the following sections:

Using the Language ID Tool

You can run the language-identification command-line tool against any plain-text, PDF, HTML, XML, or WYSIWYG document supported by the indexer, including Word, WordPerfect, Excel, PowerPoint, or RTF. The tool generates an output file that identifies the character set and the predominant language or languages in the document.

To help in evaluating multilanguage documents or documents whose language cannot be determined with complete certainty, the tool applies a score to each identified language and sorts the output that score. You can use the score to identify the predominant language or the most probable language for the document.

The language ID tool by default uses only the first 2KB of a document to detect language and encoding.

To run the Language ID tool, use the following command:

```
langid [-n num_langs] [@]input_file [-bif bif_file][output_file]
[-buffer_size detection_size] [-config config_file]
[-data_path language_data_path]
```

where

- `-n num_langs` specifies the maximum number of languages to be considered. The value must be a positive integer. (Default = 1.)

  The `-n` option is optional. It tells how many pairs of language and score values to identify in the output file for a document. For example, if the `langlist.cfg` file (see "Editing the Language Configuration File" on page 162) identifies codes for 12 languages, but you only want to see the two highest scores in the output file, include `-n 2` in the command line.

  Specifying `-n` does not mean you always get results for *`num_langs`* languages. The tool only returns language identifications that score greater than zero. That may be fewer than the number specified by `-n`.

- *`input_file`* is the file whose language (or languages) is to be identified.

  The `@` prefix is optional. It indicates that *`input_file`* a list file, meaning that it contains a list of documents whose languages are to be identified.

- `-bif` *`bif_file`* specifies the name of a bulk insert file (BIF) that you want the results written to.

  The `-bif` option is optional. It instructs the language ID tool to construct a BIF describing the results of the language analysis. Use this option if you want to use `mkvdk` to index a collection directly from the results of running this tool. It saves you the effort of manually constructing a BIF.

  Using the `-bif` option when the language ID tool analyzes a list of documents in different languages results in one BIF being generated for each identified language. For example, if the file list includes both French and English documents, and if your `-bif` option looks like

  ```
  -bif test.bif
  ```

the following two BIFs are generated:

```
test.bif.en
test.bif.fr.
```

The BIF format produced by this option contains the following elements for each document:

Value	Description
`VDKVGWKEY:`	The Verity gateway key for the input file.
`VLANG:`	The language detected in the file. Only the language with the highest score appears in these results. The value of this field is the ISO-standard language code, as listed in "Supported Language Codes" on page 135.
`CHARSET:`	The encoding detected in the file. For non-text documents, it shows `Unknown`.
`<<EOD>>`	End of data for the document. (If the file specified in [@]<input file> is a list of documents, this value separates the output data for each document.)

For more information on bulk insert files and the BIF format, see the *Verity Command-Line Indexing Reference.*

- *`output_file`* is the name of the file that the language ID tool is to write its results to.

  The *`output_file`* parameter is optional. If you do not use it, the command-line tool writes its results to standard output. In either case, the results are in the following format for each analyzed document:

Value	Description
`DOC_NAME`	The file name.
`LANGUAGE`	The name for the language detected in the file.
`CHARSET`	The encoding name detected in the file. For non-text document, it will show Unknown.

Value	Description
SCORE	A numeric value-between 0.0 and 1.0 that is relative to the other languages detected in the document. A value greater than 0.0 indicates that the language ID tool has detected the specified language in the document. The language with the highest score is listed first.
. . .	(Additional entries for LANGUAGE, CHARSET, and SCORE, if more than one language was detected for this document)
<<EOD>>	End of data for the document. (If the file specified in [@]<input file> is a list of documents, this value separates the output data for each document.)

- `-buffer_size` *`detection_size`* specifes how much of the text of a document to analyze in making a language determination.

  The `-buffer_size` option is optional. If you leave it off, the default value of 2KB applies.

- `-config` *`config_file`* specifes the location of the language configuration file to use. For an explanation of this file, see "Editing the Language Configuration File" on page 162.

  The `-config` option is optional. If you leave it off, the default value (`langlist.cfg`, located at *`productDir`*`/common/langid`) applies.

- -data_path *`language_data_path`* specifies the location of the language data files.

  The `-data_path` option is optional. If you leave it off, the default path *`productDir`*`/common/`*`langid`* applies.

Example Output Files

The following output shows the results for a text document called `francais.txt`, when the `-n` option is not used. The output shows results for the highest-scoring language only (French). This document is suitable for indexing into a French collection.

```
DOC_NAME:/data/french/francais.txt
LANGUAGE: French
CHARSET: 1252
```

```
SCORE: 0.95
<<EOD>>
```

The following output shows results for the same document, but with `-n` set to 2. The scores reveal that French is the predominant language, but some words in the document are Portuguese.

```
DOC_NAME:/data/french/francais.txt
LANGUAGE: French
CHARSET:  1252
SCORE: 0.95
LANGUAGE: Portuguese
CHARSET: 1252
SCORE: 0.03
<<EOD>>
```

The following output shows the results for a document containing nearly equal amounts of two languages (French and German). This document could be indexed into either a French or German collection.

```
DOC_NAME:/data/swiss/francais_deutsch.txt
LANGUAGE: French
CHARSET: 1252e indexed
SCORE: 0.95
LANGUAGE: German
CHARSET: 1252
SCORE: 0.90
<<EOD>>
```

The following output shows the results when the input file is a list of documents. In this case, `<<EOD>>` separates the information for each document.

```
DOC_NAME:/data/french/francais.txt
LANGUAGE: French
CHARSET: 1252
SCORE: 0.95
LANGUAGE: Portueguese
CHARSET: 1252
SCORE: 0.05
<<EOD>>
DOC_NAME:/data/german/deutsch.doc
LANGUAGE: German
CHARSET: Unknown
SCORE: 0.85
<<EOD>>
```

Tuning the Command-Line Tool

The language ID tool employs several strategies to detect the language of a document:

- For documents that contain clear language information, such as a character set name in a metatag of an HTML document, the command-line tool extracts it. However, at times this encoding information may not be the same as the actual document encoding.
- If an HTML document has more than one such metatag regarding the character set or language, only the first one is used.
- For documents that lack clear information, the command-line tool relies on the language data file and language configuration file to detect the language/encoding. The command-line tool supports the detection of up to 90 languages. Since the mechanism is configuration-data based, you can update the configuration data to tune the command-line tool to give better results within a more narrow domain.

Editing the Language Configuration File

The language configuration file `langlist.cfg` specifies how many languages a document is to be compared to for the purpose of identifying its language.

The file is located in the directory *`productDir`*`/common/`*`locale_name`*, where *`productDir`* is the pathname of the directory (such as `C:\Verity\K2`) containing the component of Verity you have installed, and *`locale_name`* is the name of the locale, such as *`productDir`*`/common/czech`.

`langlist.cfg` lists the names of *language-data files*, which contain information used for language identification. There is one language data file listed for each language/character-set combination to be analyzed.

Language data files are located in the directory *`productDir`*`/common/langid`. A language data file has a name of the form *`langID-encoding`*`.lm`, where *`langID`* is a two-character language ID and *`encoding`* is a character-set designation. For example, `ar-1256.lm`. is the language data file for Arabic documents using the Widows 1256 character set. Language IDs and character-set designations are listed in "Locales, Character Sets and Languages" on page 127.

During the language-identification process, the language ID engine processes a portion of the text of an incoming document with each language data file in turn, generating a score for that document for each language.

The `langid` directory contains language data files that represent all language/encoding pairs supported for language identification. `langlist.cfg` specifies the subset of those files to use for language identification. The following table lists the content of the default version of `langlist.cfg`.

```
ar-1256.lm          nb-1252.lm           de-utf8.lm
ar-iso-8859-6.lm    nn-1252.lm           es-utf8.lm
cs-1250.lm          pl-1250.lm           fi-utf8.lm
cs-iso-8859-2.lm    pl-iso-8859-2.lm     fr-utf8.lm
da-1252.lm          pt-1252.lm           it-utf8.lm
de-1252.lm          ru-1251.lm           ja-utf8.lm
el-1253.lm          ru-855.lm            ko-utf8.lm
en-1252.lm          ru-866.lm            nl-utf8.lm
es-1252.lm          ru-iso-8859-5.lm     nb-utf8.lm
fi-1252.lm          ru-koi8-r.lm         nn-utf8.lm
fr-1252.lm          sv-1252.lm           pl-utf8.lm
it-1252.lm          zt-big5.lm           pt-utf8.lm
ja-eucjp.lm         zh-gb.lm             sv-utf8.lm
ja-sjis.lm          ar-utf8.lm           zh-utf8.lm
ko-ksc.lm           cs-utf8.lm
nl-1252.lm          da-utf8.lm
```

Note When a document contains clear language information in its metadata, the `langlist.cfg` file is not consulted and it doesn't matter what languages it specifies. For example, if Japanese (`ja-sjis.lm`) is not listed in `langlist.cfg` but the document being analyzed (`japanese.html`) contains the tag

```
<meta http-equiv="Content-Type" content="text/html;
charset=shift-jis">
```

the language ID tool will still report the following results:

```
DOC_NAME: japanese.html
LANGUAGE: ja
CHARSET: sjis
SCORE: 1.0
```

You can add or remove language data files from this list as needed. For detailed instructions on editing `langlist.cfg`, see "Adjusting the Set of Languages to Identify" on page 107.

IMPORTANT This configuration file is used for indexing and collection creation as well as by the language ID tool. If you make changed to is for running the language ID tool, those changes will also affect any subsequent indexing through vspider or K2 Spider.

Glossary

accent-insensitive search	A type of search that includes all accented variations of a letter in the search term. In accent-insensitive search, the search term *si* would find all instances of both *si* or *sí*, for example. Conversely, in accent-sensitive search, the search term *si* would find only instances of the unaccented *si.*
Adobe PDF filter	A document filter that processes PDF files for indexing. Compare **KeyView Filter**.
auto-case	A Verity search feature which, when enabled, conducts case-insensitive search when the search term is single-case (such as *cat* or *CAT*), and case-sensitive search when the search term is mixed-case (such as *Cat* or *caT*). With auto-case, the word *cAt* would be found by searching for *cat* or *CAT*, but not by searching for *Cat* or *caT*.
auto-detection	A Verity capability in which a document is analyzed to determine its character set and/or its language. Verity's auto-detection can accurately determine both the character set and the native language of many documents.
browse	A command-line tool that displays the contents (field names and values) of a collection's document table.

case-insensitive search	A type of search in which the case of the letters in the search term does not matter. In case-insensitive search, the search term *Cat* would find all instances of *cat* or *CAT* or *Cat*, for example. Conversely, in case-sensitive search, the search term *Cat* would find only instances of *Cat*.
character set	A numeric encoding of the characters of a language. Text in a given language can be stored and manipulated using one or more character sets. Examples include ASCII, Shift-JIS, and UTF-8.
collection	A set of files and folders that stores information needed to search and classify documents in a **repository**. A collection stores the locations of all the indexed documents, the locations of all the indexed words in those documents, and metadata about the documents. It does not store the documents themselves.
compound word	A word created by the concatenation of several independent words. **Decomposition** in indexing breaks up a compound word into subwords and creates index entries for each one.
current locale	The Verity locale within whose context an indexing or text-manipulation process is occurring. Every VDK session has a current locale.
decomposition	The process of breaking a **compound word** into its constituent subwords for indexing. Searches for a subword will then return all occurrences of the compound word.
decomposition pattern	In a user dictionary for the `koreab`, `japanb`, `simpcb`, or `tradcb` locales, a numeric pattern that specifies how a compound word is to be broken into subwords.
default installation locale	The locale specified in the configuration file `verity.cfg`. If defined, it is the default session locale.
default session language	The language used as the default for queries during a VDK session. Applies only when the session locale is the multilanguage (`uni`) locale.
default session locale	The locale assigned to a VDK session if no locale is specified when the session is opened.

delimiter	A character used by the tokenizer to split document text into searchable units. For many locales, white space and punctuation are the most common delimiters.
didump	A command-line tool that generates a list of the words (tokens) in a collection's word index.
document cluster	An automatically generated grouping of similar documents, based on **document features**.
document feature	A noun or noun phrase that characterizes a topic or concept in a document. Document features are identified automatically during the process of **feature extraction**.
document filter	A driver-level plug-in software module that can read documents in one or more specific formats (such as PDF, XML, Microsoft Word). Document filters receive documents from gateways, extract text data and field information from them, and pass that information along for indexing and storage in a collection.
document key	A unique identifier assigned to each document indexed in a collection. In the document table of a collection, it is in the field `VdkVgwKey`.
document summary	A concise description of the contents of a document. An automatically generated document summary can be based on **document features** or the document's initial text.
document table	A table in a collection that specifies the location of each indexed document. The document table also contains all metadata (fields) associated with each document.
dynamic highlighting	A method of highlighting the search term in a document summary or in a retrieved document. In dynamic highlighting, the application actually searches through the results or the document to locate and highlight the term. Dynamic highlighting is slower but more accurate than **static highlighting.**
feature	See **document feature**.

feature extraction	The process of identifying the important subjects and concepts in a document by analyzing its nouns and noun phrases. Feature extraction underlies the creation of **document clusters** and **document summaries**.
foreign word	In the `uni` locale, a word in any language other than the overall language of its document.
full-width character	In Japanese, a Katakana or romaji character that occupies the same amount of horizontal space as a Kanji character. In Japanese character sets, a full-width character has a different character code from its half-width equivalent.
gateway	A driver-level plug-in software module that can retrieve documents from a specific type of platform or through a specific protocol. For example, Verity gateways exist for UNIX and Windows file systems, Web servers, and ODBC-accessible databases. During indexing, gateways pass retrieved files to document filters for processing.
half-width character	In Japanese, a Katakana or romaji character that occupies half the horizontal space of a Kanji character. In Japanese character sets, a half-width character has a different character code from its full-width equivalent.
inflectional stemming	A style of stemming in which the words of a stem are all of the same part of speech (such as noun or verb).
internal character set	The character set used internally by a locale. All collection data written in that locale, and all BIFs and style files used by that locale, must be in the locale's internal character set.
Internet-style query parser	A free-text query parser that lets users conduct familiar Web-style searches.
KeyView filter	A document filter, based on Verity KeyView technology, that is used during indexing to process many types of files.
language-data file	A file containing language-defining information. Used by the language-identification filter and the language-identification command-line tool.

language ID	A two-character (ISO 639) code that specifies an individual language. Examples are `en` for English and `zh` for simplified Chinese. Verity uses language ID for specifying languages for the multilanguage locale and the language-identification command-line tool.
language-identification command-line tool	A Verity tool that opens a document and returns a language assignment.
language-identification filter	A document filter (`flt_lang`) used by the multilanguage locale to assign a language to a document before indexing.
locale	1. A geographic or political region whose residents share the same language and customs. 2. **Verity locale**.
locale definition file	A file (`loc00.lng`) in each locale's directory that controls the language-handling characteristics of the locale.
locale directory	The directory that holds the files belonging to a particular locale. The name of a locale's directory is the name of the locale.
multilanguage locale	A Verity locale (`uni`) that supports multiple languages simultaneously. Compare **single-language locale**.
normalization	An indexing feature in which a single version of a character is used when alternate versions exist (such as half-width and full-width kana in Japanese), and a single spelling is used for a word that has alternate spellings (such as *color* and *colour* in English). Users searching a normalized collection for a word find all words with either the common spelling or any of the alternate spellings.
noun phrase	A group of words (for example, *due process* or *court of law*) that functions as a noun. Part-of-speech identification during indexing can lead to the automatic extraction of noun phrases, which can be used in the automatic creation of document features, summaries, and clusters.
okurigana	In Japanese, pronunciation marks added to Kanji words.

partition	A subdivision of a collection. Partitioning collections improves scalability and searching performance.
part-of-speech identification	During indexing, the assignment of the appropriate part of speech (noun, verb, adjective, and so on) to each token in the word index.
qp_inet.stp	A locale-specific stop-word file used by the Verity Internet-style query parser. It contains words that the query parser will strip from query terms before conducting a search. See also **vdk30.stp**.
session character set	The character set used for input to and output from VDK during a VDK session. Must be a character set supported by the session locale.
session locale	The locale used for all operations during a VDK session.
simple tokens	A behavior, available for some locales, in which nearly all symbols (in addition to white space and punctuation) are defined as delimiters. In simple-tokens behavior, words are broken down into smaller searchable units, thus increasing the potential for search hits.
single-language locale	A Verity locale that supports only one language. Most locales are single-language. Compare **multilanguage locale**.
sorting order	The order in which a locale sorts the characters of its language. Verity locales sort characters in a manner that facilitates accent-insensitive and case-insensitive search and display.
Soundex search	A type of search in which occurrences of the search term plus any words with similar pronunciation are returned. Verity supports Soundex search for the English language only.
static highlighting	A method of highlighting the search term in a document summary or in a retrieved document. In static highlighting, the application uses offsets in the collection's word index to calculate the positions of terms to highlight. Static highlighting is faster but less accurate than **dynamic highlighting.**

stem	See **word stem**.
stemmed search	A type of search that locates all words that share the same word stem. For example, a stemmed search for the term *house* would find all occurrences of *house*, but also all occurrences of *houses*, *housed*, and *housing*.
stemming	The process of extracting a word's root portion, or word stem, during indexing. For example, *house* is the word stem for *houses*, *housed*, and *housing*. Indexing of word stems makes stemmed search possible.
`stopword.`*`ID`*	Specifies stop words for a particular supported language in the multilanguage locale. The *`ID`* variable is the language ID.
stop word	A search term that should be ignored. Verity supports several types of stop-word lists, some used at indexing time and others used at search time.
style.dft	A collection style file that controls the contents of the virtual document created during indexing.
style.fxs	A collection style file that contains feature-extraction stop words—words that should not appear in document summaries and clusters. See also **`vdk30.stp`**.
style.lex	A collection style file that can control how tokenization occurs during indexing. Use of `style.lex` is discouraged; tokenization control is now available through the locale definition file associated with each locale.
style.prm	A collection style file containing parameters that control the generation of specialized indexes.
style.stp	A collection style file that contains indexing stop words—words that should not be included in the collection's word index.
style.ufl	A collection style file that defines custom fields to be included in the collection's document table and optionally specifies the generation of indexes for those fields.

style.uni	A collection style file that controls the functioning of the universal filter.
subword	A constituent element of a compound word.
summary	See **document summary**.
synonym search	A type of search that returns all occurrences of the search term and also any of its synonyms, as defined in a **thesaurus**.
system default locale	It is the default session locale if the default installation locale is not defined.
thesaurus	A dictionary of synonyms. Each Verity locale support use of a thesaurus for searching. In a synonym search, all occurrences of the search term and any of its synonyms are returned.
token	A searchable unit in a document. Tokens are typically the individual words in a document, but they can also be word stems, subwords, or any string fragments that occur between delimiter characters.
tokenization	The process by which the tokenizer converts a document's text into searchable units (such as words and word stems). The tokens are then stored in a collection's word index.
typo search	A type of search that corrects for minor misspellings in the search terms. In a typo search, occurrences of the search term and any words close to it in spelling are returned.
Unicode	A standard for 16-bit character sets. Unicode provides character encoding for all major modern languages. There are various implementations of portions of the Unicode standard. The implementation used by the Verity multilanguage locale is UTF-8.
VDK	1. Verity Developer's Kit, the API that enables OEM developers to build Verity functionality into their products. 2. The Verity search engine and other core Verity technology.

vdk30.stp	A locale-specific file that contains feature-extraction stop words—words that should not appear in document summaries and clusters. See also **`style.fxs`**.
Verity locale	A driver-level plug-in software module that allows Verity applications to operate on documents in a wide variety of languages. Locales provide language-specific tokenization, stemming, part-of-speech identification, and thesaurus use. See also **single-language locale**, **multilanguage locale**.
virtual document	A pure text version of a document, constructed by a document filter. The virtual document is converted by the tokenizer into tokens to be stored in the word index.
wildcard search	A type of search in which the search term contains special symbols that represent multiple characters. For example, a wildcard search with the term *abc** returns occurrences of all words that start with *abc*.
word index	In a **collection**, a list of all words that appear in the documents, plus the location of every instance of the word.
word stem	The root portion of a word. For example, *house* is the word stem for *houses*, *housed*, and *housing*. Indexing of word stems makes stemmed search possible.
XML filter	A document filter that processes XML documents.
zone	A named, searchable region of a document. Examples are HTML tags such as `H1` or `BODY`, an the values of email and Usenet message fields such as `TO` or `SUBJECT`.
zone filter	A document filter that processes documents—such as HTML, Usenet news, and email documents—that contain zones. See also **XML filter**.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z