Verity® Locale Configuration Guide V 5.0 for PeopleSoft®

November 15, 2003
Original Part Number DM0619

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

Verity Benelux BV
Coltbaan 31
3439 NG Nieuwegein
The Netherlands

Table of Contents

Preface

Chapter 1 Language Concepts

Chapter 2 Verity Locales

Chapter 3 Using Locales

Appendix A Locales, Character Sets and Languages

Appendix B Tokenization Delimiters

Appendix C Creating a Custom Thesaurus

Appendix D Glossary

Index

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.

Using This Manual

This section describes the organization of the *Verity Locale Configuration Guide* and lists the stylistic conventions used.

Version

The information in this manual is current as of K2 Enterprise version 5.0. The content of the manual was last modified November 15, 2003.

Organization of This Manual

This manual contains the following chapters and appendixes:

- **Chapter 1, "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 2, "Verity Locales."** Describes the language-handling characteristics of each Verity locale.
- **Chapter 3, "Using Locales."** Describes how install, configure, and use Verity locales.
- **Chapter 4, "Creating Language-Aware 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 word delimiters to control indexing.
- **Appendix C, "Creating a Custom Thesaurus."** Describes how to customize or create a thesaurus file, used to support synonym search for a given locale.

Stylistic Conventions

The following stylistic conventions are used in this manual.

Convention	Usage
`Courier type`	Used to format file names, paths and required user input. Examples: The `name.ext` file is installed in: `C:\Verity\Data\` In the **User Interface** text box, type `user1`.
`Courier oblique type`	Used for user-replaceable strings. For example: `user` *`username`*
`Courier bold`	Used to format command-line tool names. For example: The **`rck2`** command-line tool allows you to search collections and test the effects of your changes.
Palatino	Used in narrative text.
Palatino bold	Used in narrative text to format user interface elements. For example: Click **Cancel** to halt the operation.
italics	Used for book titles and new terms that are defined. A *newterm*, explanation of term.

Command-Line Tool Syntax

The following conventions are used in this manual to describe command-line tool syntax:

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 or 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 either `option1`, `option2`, or both options.

Convention	Usage
`required`	Absence of braces or brackets indicates required syntax in which there is no choice; you must enter the required syntax without modification, as in **`mkre`**.
`variable`	Italics specify variables to be replaced by actual values, as in `C:\MyData` for *`filename`*.
`...`	Ellipses indicate repetition of the same pattern, as in `-merge` *`filename1, filename2`* `[,` *`filename3`* `... ]` where the ellipses specify `,` *`filename4`*, and so on.

Punctuation characters, such as single and double quotes, commas, and periods indicate actual syntax; they are not part of the syntax definition.

1

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 and summarizes the language-related issues that it addresses. The chapter includes the following sections:

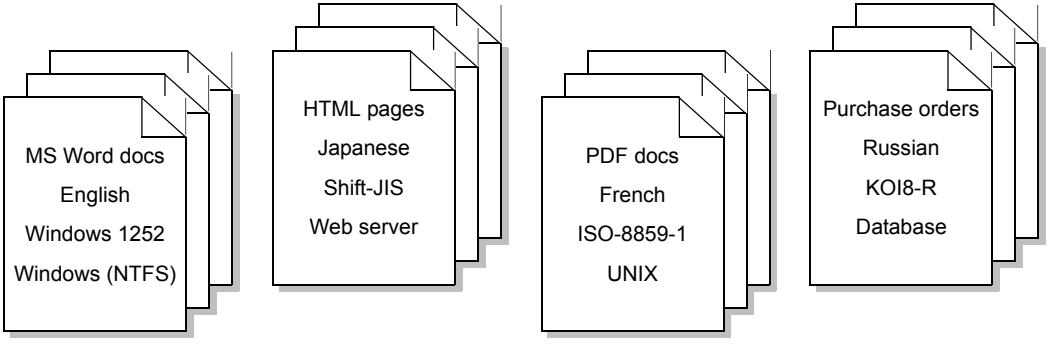
- Language and Encoding in Documents
- Language-Related Indexing Features
- Language-Related Search Features
- Limitations in Handling Source Documents

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

Figure 1-1: 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 implementable in several character sets and, conversely, a single character set can sometimes be used to store text in several languages.

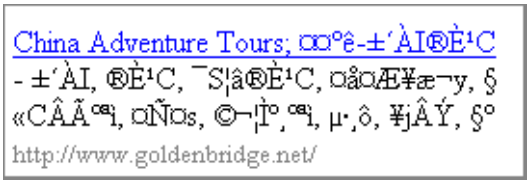
  Verity software can read document text stored in dozens of different character sets, and

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 57 languages, and it provides linguistically sophisticated support for 26 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.

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 the Western European locales, such as `englishx`.

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 current Verity Western European 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 “Customizing Tokenization Behavior” in Chapter 3 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.

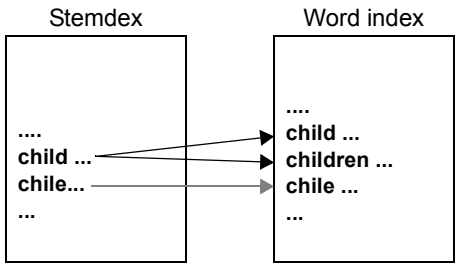
(Verity locales use *inflectional* stemming, meaning that only stems of the same part of speech as the word being stemmed are extracted. In the above example, all are verbs.)

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 Guide* and the *Verity K2 Enterprise Intelligent Classification Guide* for more information.

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:



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.

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), 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.

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).

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 locales, 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 in Japanese” in Chapter 3 for details.

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.

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**. When installed, most locales are pre-configured to specify accent-insensitive searches. The Verity administrator can in some cases reconfigure those locales to make searches accent-sensitive, if desired.

Symbol Search

Normally, punctuation, white space, and other non-alphanumeric characters are not searchable. In Western European locales, 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 Appendix C, "Creating a Custom Thesaurus."

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 `englishx` locale only.

Typo Search

For single-byte 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 kinds of 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.

  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 V*erity Collection Reference Guide*.

- `style.fxs`. This stop-word file 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.

  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 V*erity Collection Reference Guide*.

- `qp_inet.stp`. This stop-word file is used by the Verity Internet-style query parser. 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 Guide*.

- `vdk30.stp`. This 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.

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.st` 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 3 of this book. The QBE parser is described in the *Verity Query Language Guide*.

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, and handles language assignment in this way:
  - *Single-language locales.* The file is indexed according to the language rules of the current locale.
- **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.
- **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.
  - *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.

2

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
- Western European Locales
- Eastern European / Middle-Eastern Locales
- Asian Locales

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

verity_product\common*locale_name*

where

- *verity_product* is the path to the directory containing the component of Verity that has been installed (for example, usr/verity/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, germanx).

In a K2 Services installation, the library files are in the directory

*verity_product**os_platform*\bin

where *os_platform* is the name of the operating system-specific directory (for example, _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. Third-party library files required by the locale are also in the bin directory.

Locale Definition File

The file loc00.lng, in the directory *verity_product*\common*locale_name*, controls several aspects of locale behavior. The Verity administrator can edit that file to customize certain aspects of the locale's behavior. See "Configuring Verity Locales" in Chapter 3 for details on editing loc00.lng.

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 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 under "Verity Locales and Character Sets" in Appendix A of this book.

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 is specified in the Verity configuration file (`verity.cfg`). Its initial value is `englishx`.
3. If the default installation locale is not defined, Verity uses the *system default locale* as the session locale. The system default locale is also `englishx`.

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.

NOTE: If your installation requires it, you can reset the default installation locale from `englishx` to the older Verity locale (`english`). See "Redefining the Default Session Locale" in Chapter 3 for instructions.

Built-In Locales

The following Verity locales are installed automatically when K2 Services is installed:

Verity locale name	Language
`englishx`	English
`uni`	Multiple languages (UTF-8)
`english`	English (basic)

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

- Use of the `englishx` locale is covered by K2 Services (or VIC or VDK) license. No separate locale license is required.
- 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:

- Western European locales
- Eastern European and Middle-Eastern locales
- Asian locales

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

Western European Locales

The locale modules in this category include the built-in `englishx` locale plus other locales serving the languages native to Western Europe. These locales make use of language-processing technology from inXight Software, Inc. (version 2.2), in combination with Verity's own language capabilities.

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

Verity locale	Language	Verity locale	Language
`bokmalx`	Norwegian	`germanx`	German
`danishx`	Danish	`italianx`	Italian
`dutchx`	Dutch	`nynorskx`	Norwegian
`englishx`	English	`portugx`	Portuguese
`finnishx`	Finnish	`spanishx`	Spanish
`frenchx`	French	`swedishx`	Swedish

Verity Western European locales support the indexing and search features described in the following table.

Western European locale features

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.

Western European locale features (continued)

Feature	Support
Stemming	All locales support stem indexing and search.
Normalization	No normalization applied.
Compound words	Decomposition into subwords supported by `dutchx`, `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	Accent-insensitive search is supported for all locales and is the default.
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 `englishx`.
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.

Eastern European/Middle-Eastern Locales

The locale modules in this category serve the languages of Eastern Europe and Russia, Southeastern Europe, and the Middle East. These locales make use of character-set-based tables and Verity's language capabilities.

The following table lists the currently available Eastern European and Middle-Eastern locales and their internal character sets. For common names of the listed character sets, see Appendix A, "Locales, Character Sets and Languages."

Verity Locale	Language	Charset	Verity Locale	Language	Charset
arabic	Arabic	1256	hungarian	Hungarian	1250
bulgaria	Bulgarian	1251	russian	Russian	1251
czech	Czech	1250	russian2	Russian	koi8-r
greek	Greek	1253	polish	Polish	1250
hebrew	Hebrew	1255	turkish	Turkish	1254

Verity Eastern European and Middle-Eastern locales support the indexing and search features noted in the following table:

Eastern European/Middle-Eastern locale features

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 assigned the locale's internal character set.
Language identification	Verity's language-detection technology identifies the language of source documents to be indexed.
Sorting order	All locales use case-insensitive and accent-insensitive sorting behavior based on the locale internal character set. (Indexing rules are based on the current locale, not the document language.)
Tokenization	Performed by all locales. Tokenization delimiters are editable.
Stemming	Not supported.
Normalization	No normalization is performed.
Compound words	Decomposition into subwords not supported.

Eastern European/Middle-Eastern locale features (continued)

Feature	Support
Part-of-speech	Not supported.
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	Accent-insensitive search is supported for all locales and is the default.
Searchable symbols	Not directly supported. However, symbols can be re-defined as either punctuation or regular (alphabetic) characters.
Synonym search	All locales support use of thesaurus for synonym search.
Soundex search	Not supported.
Typo search	Supported by all locales.
Wildcard search	Supported by all locales.
Stop words	All locales support use of a stop-word list for use in feature extraction and by the free-text query parser. Verity provides a simple default stop-word file for all locales.
Date formatting	For date fields in a collection, all locales support dates with month and day names in the locale's language.

Asian Locales

The locale modules in this category serve the multiple-byte languages of East Asia: Chinese, Japanese, and Korean. These locales make use of language-processing capabilities from Basis Technologies Corp (version 3.6.2).

The following table lists the currently available Asian locales and their internal character sets. For common names of the listed character sets, see Appendix A, “Locales, Character Sets and Languages.”.

Verity Locale	Language	Charset
`japanb`	Japanese	sjis
`koreab`	Korean	ksc
`simpcb`	Chinese (simplified)	gb
`tradcb`	Chinese (traditional)	big5

Asian locales support the indexing and search features noted in the following table:

Asian locale features

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.

Asian locale features (continued)

Feature	Support
Normalization	`japanb`: Half-width Kana equivalent to full-width Kana. Katakana indexed as equivalent Hiragana. 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.

Asian locale features (continued)

Feature	Support
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.

3

Using Locales

This chapter describes how to use Verity locales to provide appropriate language-specific indexing and searching capabilities. It also gives suggestions for creating Verity data structures (such as collections) in locales other than the default (`englishx`).

This chapter includes the following sections:

- Configuring Verity Locales
- Notes on Creating Non-English Indexes

Configuring Verity 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.

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" in Chapter 2.) 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 `englishx`, which is the installed default.

> NOTE: You can use this technique to switch the default session locale between `english` and `englishx` 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, specified for K2 installations in the Verity configuration file (`verity.cfg`). Take these steps to change it:

1. Open the file `verity.cfg`, in the directory *`verity_product`*`/common`, where *`verity_product`* is the path to the directory containing the component of Verity that has been installed (for example, `usr/verity/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:

   ```
   locale=englishx
   ```

3. Change it to

   ```
   locale=english
   ```

4. Save and close the file.

Customizing Tokenization Behavior

For Western European and Asian locales, you can change certain aspects of tokenization behavior by making modifications to the simple-tokens behavior specified in the locale's `loc00.lng` file.

Customizing tokenization through `loc00.lng` is not available for other locales.

NOTE: After making the changes described in this section, you must re-index existing collections if you want the changes to apply to those collections.

Disabling and Enabling Simple Tokens

For Western European and 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.

For Chinese in the `uni` locale, enabling simple tokens also enables single-character tokenization, which means that each Chinese character becomes a separate token. (This is in addition to word-level tokenization, which remains.)

Simple-tokens behavior is not supported for Eastern European and Middle-Eastern locales.

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 *`verity_product`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which should look something like this:

   ```
   driver: "loc_xlt -simple_tokens..." "loc_xlt"
   ```

3. To disable simple tokens, remove the `-simple_tokens` option (plus any `-tokenized_as_alphabet` and `-searchable_non_alphabet` options that follow it), leaving something like this:

   ```
   driver: "loc_xlt" "loc_xlt"
   ```

4. Save and close the file.

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

NOTE: Specifying the default `-simple_tokens` option (without any following options) is equivalent to this specification:

```
-simple_tokens
-tokenized_as_alphabet -_&
-searchable_non_alphabet #$%©®¢£¥™
```

See the next two sections, "Refining the Set of Token Delimiters" and "Making Symbols Searchable," for explanations of the `-tokenized_as_alphabet` and `-searchable_non_alphabet` options.

Refining the Set of Token Delimiters

When it indexes a document, the Verity tokenizer breaks words at whitespace and punctuation characters (see "Tokenization and Word Delimiters" in Chapter 1). If simple-tokens behavior is enabled for a locale, you can modify the set of symbols that are considered punctuation for tokenization.

This section shows the process for Western European locales. For Asian locales, changing the set of delimiters is not supported.

NOTE: After making these changes, you must re-index existing collections if you want the changes to apply to those collections.

Western European Locales

Western 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 *`verity_product`*`\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 Appendix B, "Tokenization Delimiters."
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," later in this chapter, for an illustration of how these settings affect tokenization results.

Eastern European and Middle-Eastern Locales

Simple-tokens behavior is not available for these locales. To modify the set of token delimiters for one of these locales, you can directly edit the CTYPE table in the locale's `loc00.lng` file.

Asian Locales

For Asian locales, tokenization of native script is word-based and not customizable, but tokenization of ASCII text by default uses simple-tokens behavior. For example, the driver statement in the `japanb` locale looks like this:

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

You can disable simple-tokens behavior by deleting the option, but you cannot alter the set of delimiters used.

For the locales `simpcb` and `tradcb`, the same simple-tokens behavior applies, but you can also 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. 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.

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" in Chapter 1.)

This feature is fully supported only for Western European locales.

Western European Locales

To make symbols searchable, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`verity_product`*`\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 Appendix B, "Tokenization Delimiters."
5. Save and close the file.

See "Tokenization Example," later in this chapter, for an illustration of how these settings affect tokenization results.

Eastern European and Middle-Eastern Locales

Searchable-symbols behavior is not provided for these locales, because simple-tokens behavior is not available. However, you can redefine symbols as punctuation or as alphabetic characters for one of these locales by directly editing the CTYPE table in the locale's `loc00.lng` file.

Tokenization Example

The table in this section shows two examples of tokenization in a Western 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.

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.

Disabling and Enabling Stemming

In a stemmed search (see “Stemming” in Chapter 1), 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.

Stem indexing is enabled by default in Western European locales and Asian locales `japanb` and `koreab`. Stemming is not available in the Eastern European / Middle-Eastern locales, and stemming is not applicable to the Chinese locales.

For improved indexing speed, you might wish to disable stemming for a given locale. For Japanese and Korean in particular, disabling stemming can speed indexing—at the expense of supporting stemmed search, of course.

Asian Locales

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 *`verity_product`*`\common\`*`locale_name`*.
2. In the `locale` block, locate the `driver` statement, which for the locale should look something like this:

   ```
   driver:  "locbasis -simple_tokens"  "locbasis"
   ```

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

   ```
   driver:  "locbasis -no_stems -simple_tokens"  "locbasis"
   ```

4. Save and close the file.

Other Locales

For locales other than `uni`, `japanb`, and `koreab`, there is no locale-specific control on stemming. If you want to disable or enable stemming for a collection built in one of those locales, 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 *`verity_product`*`\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 Guide*.

Customizing Word Decomposition in Japanese

The Verity `japanb` locale allows 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. 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.

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).

Installing the User Dictionary

When your user dictionary is complete, install it this way:

1. Give it any name.
2. Store it in the locale's directory (*`verity_product`*`\common\japanb`).
3. Open the locale's `loc00.lng file` and add a `user_dictionary` option to the driver entry, like this:

```
driver:  "locbasis -simple_tokens & -user_dictionary 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_dictionary` option with a comma-separated list of dictionary filenames:

```
-user_dictionary 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 sample user dictionary (`sample_dict.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.

Changing Search Characteristics

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

NOTE: After making the changes described in this section, you must re-index existing collections if you want the changed behavior to apply to those collections.

Enabling Case-Sensitive Search

All locales have built-in support for case-sensitive searching. For multibyte 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 when you build a collection, 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 Guide*.

Enabling Auto-Case

As described in "Case-Insensitive Search" in Chapter 1, 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.

For all single-byte locales, auto-case is disabled by default. (Auto-case does not apply to multibyte locales.) If you want to enable auto-case, take these steps:

1. Open the locale's definition file `loc00.lng`, in the directory *`verity_product`*`\common\`*`locale_name`*.
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.

Disabling Accent-Insensitive Search

Accent-insensitive search (see "Accent-Insensitive Search" in Chapter 1) treats all accented variations of a single character as the same character. For all single-byte locales, accent-insensitive search is enabled by default at installation. (Accented text does not occur in multibyte locales.)

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" in Chapter 1) will contain only unaccented versions of their characters.
- Collections created with an earlier, accent-sensitive, version of the locale my need to be re-indexed to retain the same search behavior.

If for these or other reasons you wish to treat accented characters individually, you can disable accent-insensitivity.

NOTE: This procedure is available for Western European locales only.

For most Western European locales, accent-insensitivity is enabled by default. For the `englishx` locale, however, accent-insensitivity is disabled by default, to minimize the need to re-index existing collections created with the older `english` locale.

To disable accent insensitivity in a Western European locale, take the following steps.

1. In the locale's directory (*`verity_product`*`\common\`*`locale_name`*), locate the three files

   `loc00.lng`
   `xlt.ia`
   `xlt.is`

   and rename them—for example, to something like

   `loc00.lng.50`

```
xlt.ia.50
xlt.is.50
```

This action saves off copies of the current, accent-insensitive versions of your locale definition file and search-configuration files.

2. In the same directory, locate the three files

```
loc00.45
xlt.ia.45
xlt.is.45
```

and rename them to

```
loc00.lng
xlt.ia
xlt.is
```

This action replaces the accent-insensitive versions of your locale definition file and search-configuration files with the accent-sensitive versions.

3. If you had previously edited `loc00.lng` to customize behavior (for example, to enable auto-case), be sure to restore those edits in the replacement `loc00.lng`.

To enable accent-insensitivity in the `englishx` locale, or to reconfigure another locale back to accent-insensitive searching, reverse the process:

1. Rename and save your accent-sensitive files as `loc00.45`, `xlt.ia.45`, `xlt.is.45`.
2. Restore your accent-insensitive files to their valid names (`loc00.lng`, `xlt.ia`, `xlt.is`).

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 *`verity_product`*`\common\`*`locale_name`*.
2. In the `locale` block, locate 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 *`verity_product`*`\common\`*`locale_name`*.
2. In the `locale` block, locate 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

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" in Chapter 1).

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 Appendix C, "Creating a Custom Thesaurus."

Creating a Stop-Word File

As noted in "Stop Words" in Chapter 1, 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, only one of which is locale-specific.

The file `vdk30.stp`, in the directory *`verity_product`*`\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-insensitive. You do not need to list all case variants.
   - The order of the words is not important.
   - Enter only literal words. Regular expressions are not supported.

   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:

verity_product\common*locale_name*

where *verity_product* 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 V*erity Collection Reference Guide.*

Configuring Language Identification

By default, language identification occurs as a part of indexing in all locales. 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 Guide.*

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 meta tag

```
<meta http-equiv="Content-Type" content="text/html; charset=shift-jis">
```

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 *`verity_product`*`\common\langid`. That directory also contains the *language-data files*—the files containing the language-defining information—of all Verity-supported languages.

This is the content of the default version of `langlist.cfg`:

```
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
```

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" in Appendix A) and character set (see "Supported Source-Document Character Sets" in Appendix A) to which it applies. The languages enabled here are German, English, and French (in the Windows 1252 character set).

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

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.

You can disable language identification by either of these two methods:

- Delete the entire `langid` directory from the *`verity_product`*`\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 *`verity_product`*`\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.

Notes on Creating Non-English Indexes

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.

Locale and Character Set for Collections

When creating a non-English collection, note the following locale and character-set issues.

Using Verity Spider to Create a non-English Collection

You can create a collection using the Verity Spider, executed as a command-line tool (**`vspider`**). If you are creating a non-English collection, you need to specify the locale you want for the collection and possibly also the character set you want to view it with.

> 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.

With **vspider**, specify either the `-locale` or `-charmap` options:

Option	Description
`-locale` *`locale_name`*	Create the collection in the locale specified by *`locale_name`*. 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" in Appendix A. This option is not required if the collection is to be in the default locale; see "Default Locales and the Session Locale" in Chapter 2.
`-charmap` *`charset_name`*	Specify that **vspider** is to use the character set specified in *`charset_name`* to display collection information to the screen. *`charset_name`* must be the name of one of the supported character sets for the locale specified in *`locale_name`*. This option is not required if you want to display collection information in its locale's internal character set. "Verity Locales and Character Sets" in Appendix A lists the supported character sets for each Verity locale and indicates which one is the internal one.

Locale and Character Set for Command-Line Tools

When using the command-line tools **`vspider`** and **`rcvdk`** on a collection, you might need to include the `-locale` option, plus possibly the `-charmap` option, to access and properly display the content of the collection or index.

Option	Description
`-locale` *`locale_name`*	The name of the locale in which the collection or other index you are accessing was created. *`locale_name`* must be the name of a locale for which you are licensed, and must be one of the Verity locales listed in "Verity Locales and Character Sets" in Appendix A. This option is not required if the collection or index uses the default session locale; see "Default Locales and the Session Locale" in Chapter 2.
`-charmap` *`charset_name`*	Use the character set specified in *`charset_name`* to display the contents of the collection or other index. *`charset_name`* must be the name of one of the supported character sets for the collection's locale. This option is not required if you want to display the collection data using its locale's internal character set. "Verity Locales and Character Sets" in Appendix A lists the supported character sets for each Verity locale and indicates which one is the internal character set.

NOTE: *`locale_name`* refers to the locale you use to access the collection, and *`charset_name`* refers to the character set you use to display its content. Neither necessarily corresponds to the language or character set of any of the documents indexed by the collection.

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 Chapter 2, “Verity Locales.” and Chapter 3, “Using Locales.”

This appendix includes the following sections:

- Verity Locales and Character Sets
- Supported Source-Document Character Sets
- Supported Language Codes

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.

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 German as the locale for a collection you are creating with the **rcvdk** tool, use the option `-locale germanx`.

- **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 `germanx` locale, if you want the output to use the MS-DOS character set, use the option `-charmap 437`.

  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.

Verity locales and character sets

Verity locale	Language	Internal character set	Other supported character sets
`uni`	(All languages)	**utf8**	
`arabic`	Arabic	**1256**	8859-6
`bulgaria`	Bulgarian	**1251**	
`czech`	Czech	**1250**	8859-2
`danishx`	Danish	**1252**	437, 850, 8859
`dutchx`	Dutch	**1252**	437, 850, 8859

Verity locales and character sets (continued)

Verity locale	Language	Internal character set	Other supported character sets
englishx	English	**1252**	437, 850, 8859
finnishx	Finnish	**1252**	437, 850, 8859
frenchx	French	**1252**	437, 850, 8859
germanx	German	**1252**	437, 850, 8859
greek	Greek	**1253**	8859-7
hebrew	Hebrew	**1255**	8859-8
hungarian	Hungarian	**1250**	8859-2
italianx	Italian	**1252**	437, 850, 8859
japanb	Japanese	**sjis**	eucjp, iso2022_jp
koreab	Korean	**ksc**	
bokmalx	Norwegian	**1252**	437, 850, 8859
nynorskx	Norwegian	**1252**	437, 850, 8859
polish	Polish	**1250,**	8859-2
portugx	Portuguese	**1252**	437, 850, 8859
russian	Russian	**1251**	8859-5, koi8-r
russian2	Russian	**koi8-r**	1251, 8859-5
simpcb	Chinese (simplified)	**gb**	big5
spanishx	Spanish	**1252**	437, 850, 8859
swedishx	Swedish	**1252**	437, 850, 8859
tradcb	Chinese (traditional)	**big5**	gb
turkish	Turkish	**1254**	8859-3, 8859-9

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."

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)
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)

Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
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)
big5	cp950, Big5, Windows 950, IBM 950	Chinese (traditional)
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)

Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
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)
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)

Supported source-document character sets (continued)

Encoding Name	Typical aliases and alternate spellings	Comment
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)
iso-2022-cn	Iso2022cn, Iso-2022-cn	Chinese (7-bit) (GB2312 pus CNS 11643)
iso-2022-jp	Iso2022jp, Iso-2022-jp	Japanese (7-bit)
iso-2022-kr	Iso2022kr, Iso-2022-kr	Korean (7-bit)
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
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)

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 language-identification command-line tool.

- 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 Appendix C, “The Language ID Command Tool.”

Verity-supported language codes

Language	Code	OK for <lang*/id*> and uni*/id*?
Afrikaans	af	
Albanian	sq	
Arabic	ar	
Armenian	hy	
Basque	eu	
Belarusian	be	
Bengali	bn	
Bulgarian	bg	
Catalan	ca	
Cherokee	chr	
Chinese (simplified)	zh	Yes
Chinese (traditional)	zt	Yes
Croatian	hr	
Czech	cs	Yes
Danish	da	Yes
Devanagari (Hindi)	hi	
Dutch	nl	Yes
English	en	Yes
Ethiopic	gez	
Estonian	et	
Finnish	fi	Yes
French	fr	Yes
Georgian	ka	
German	de	Yes
Greek	el	Yes
Gujarati	gu	
Gurmukhi (Panjabi)	pa	
Hebrew	he	

Verity-supported language codes (continued)

Language	Code	OK for <lang*id*> and uni*id*?
Hungarian	hu	Yes
Icelandic	is	
Indonesian	id	
Italian	it	Yes
Japanese	ja	Yes
Kannada	kn	
Khmer	km	
Korean	ko	Yes
Lao	lo	
Latvian	lv	
Lithuanian	lt	
Macedonia	mk	
Malay	ms	
Malayalam	ml	
Mongolian	mm	
Myanmar	bms	
Norwegian (Bokmal)	nb	Yes
Norwegian (Nynorsk)	nn	Yes
Oriya	or	
Philippine (Tagalog, Hanunoo, Buhid, Tagbanwa)	phi	
Polish	pl	Yes
Portuguese	pt	Yes
Romanian	ro	Yes
Russian	ru	Yes
Serbian	sr	
Sinhala	si	
Slovak	sk	
Slovenian	sl	
Spanish	es	Yes

Verity-supported language codes (continued)

Language	Code	OK for <lang/*id*> and uni/*id*?
Swahili	sw	
Swedish	sv	Yes
Syriac	syr	
Tamil	ta	
Telugu	te	
Thai	th	
Thanna (Dhivehi)	div	
Tibetan	bo	
Turkish	tr	Yes
Ukranian	uk	
Vietnamese	vi	
(All others)	un	Yes

B

Tokenization Delimiters

This appendix lists the tokenization delimiters applied by default to Western European locales when simple-tokens behavior is enabled. Individual symbols in this table can be removed from the list of delimiters and/or made searchable. See “Refining the Set of Token Delimiters” and “Making Symbols Searchable” in Chapter 3.

Available tokenization delimiters

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
2E	U+002E	FULL STOP
2F	U+002F	SOLIDUS

Available tokenization delimiters (continued)

Character Code		Description
1252	Unicode	
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
92	U+2019	RIGHT SINGLE QUOTATION MARK
93	U+201C	LEFT DOUBLE QUOTATION MARK
94	U+201D	RIGHT DOUBLE QUOTATION MARK

Available tokenization delimiters (continued)

Character Code		Description
1252	Unicode	
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
B5	U+00B5	MICRO SIGN
B6	U+00B6	PILCROW SIGN
B7	U+00B7	MIDDLE DOT
B8	U+00B8	CEDILLA

Available tokenization delimiters (continued)

Character Code		Description
1252	Unicode	
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

Creating a Custom Thesaurus

Synonym search is a type of search that locates occurrences of either the search term or any of its synonyms. For example, a synonym search for **brave** might return documents that contain **brave** or **courageous** or **fearless**. A search application specifies a synonym search by adding the VQL operator <THESAURUS> to the user's search term.

Synonym search requires the use of a *thesaurus* file, which lists groups of synonyms. Verity K2 includes a default English thesaurus that may be adequate for most purposes in the `englishx` locale. To construct a thesaurus for use in other locales, or to create a custom English thesaurus, follow the instructions in this appendix.

The <THESAURUS> operator is described in the *Verity Query Language Guide.*

This appendix includes the following sections:

- Creating a Thesaurus Control File
- Compiling a Thesaurus with mksyd
- Integrating the Thesaurus with Verity

Creating a Thesaurus Control File

A Verity thesaurus is a compiled file with a `.syd` extension. To create or modify a thesaurus, you need to first create or edit a text file called a *thesaurus control file*, which has a `.ctl` extension. You then compile the control file into a locale-specific thesaurus file with the **`mksyd`** command-line tool.

When creating a thesaurus, you can

- Create a complete control file using a text editor.
- Edit an existing control file to add or remove synonyms.
- Purchase a commercial thesaurus, then turn it into a thesaurus control file by adding the statements described here.

NOTE: You can re-create a control file from a thesaurus; see "Creating a Control File from an Existing Thesaurus," later in this appendix.

Control-File Structure

A thesaurus control file contains synonym lists. Each list is defined by the `list` keyword. The list contains synonyms and, optionally, *keys*. Keys are words that must appear in the search term for the synonym list to be used. In other words, if a search term consists of one of the non-key words in a synonym list, the term itself is searched for, but none of its synonyms is. A list with specified key terms is an *asymmetric* list.

If a given list has no keys, every synonym in the list is considered a key, and the list is *circular*.

The following is an example of a small thesaurus control file.

```
$control:1
synonyms:
{
list: "abort,miscarry,terminate,halt,end,fail"
list: "cease,stop,desist,terminate,end,discontinue"
list: "karma <or> fate <or> destiny"
  /keys = "karma"
}
$$
```

The first two lists are circular; the third is asymmetric. A synonym search for any term in the first list, for example, will locate that term plus any of its synonyms. Likewise, a synonym search for **karma** will find all occurrences of **karma**, **fate**, or **destiny**. However, a synonym search for **fate** will find only occurrences of **fate**.

If a key word (explicit or implicit) appears in more than one list, all lists for which it is a key are included in the synonym search. For example, note that the words **terminate** and **end** are keys in two lists in this example. In this case, a thesaurus query for either **terminate** or **end** results in an expanded query containing both lists:

```
 "(cease,stop,desist,terminate,end,discontinue) <or>
(abort,miscarry,terminate,halt,end,fail)"
```

A list can be more than a simple comma-separated set of terms. Note that the third list in this example includes the query expression `"karma <or> fate <or> destiny"`.
You can use query expressions in a thesaurus control file to apply sophisticated search logic to synonyms or to override default the default query expansion of synonym lists. See "The qparser Keyword," later in this appendix, for more information.

The control Directive.

The `$control:1` directive must be the first non-comment line in the control file.

The synonyms Keyword

The `synonyms` keyword is required in a thesaurus control file. It must appear directly after the `$control:1` directive.

The list Keyword

The `list:` keyword specifies the synonyms in a list, either in query form or in a list of words or phrases separated by commas. The optional modifier `/keys` specifies the keys list, which must be a list of words separated by commas. If `/keys` is absent, all synonyms in the list become keys. The optional modifier `/op-default` defines the fallback operator to use if there is no match for a thesaurus query.

The maximum length for a single list is 32,000 characters.

NOTE: If you separate your list into multiple lines (inserting new lines), you must include a backslash (\) at the end of each line so that the lines are treated as one list.

The following is a sample `list` statement:

```
list:"happy, joyous, joyful, glad, blithe, merry,\
cheerful, contented, blissful, delighted, satisfied,\
pleased, favored, lucky, fortunate, propitious,\
appropriate, felicitous, befitting"
```

The qparser Keyword

The synonym lists in a thesaurus control file are parsed and expanded as queries when the thesaurus is created.

The default expansion applied during thesaurus creation is different from the default expansion applied to user queries by applications that use the simple query parser. For example, the simple query parser expands a list of words separated by commas (the default combination operator) by applying the <ACCRUE> operator to the list. In default thesaurus query expansion, however, the comma-separated list is expanded by applying the <ANY> operator to it.

The following table lists the default values for expansion operators during thesaurus creation.

Type of Expansion	Operator in thesaurus creation	Comment
leaf operator	<STEM>	Synonyms are stemmed for searching
combination operator	<ANY>	Synonym searches are not ranked
phrase operator	<PHRASE>	Phrases are searched as phrases

To make sure that the same expansion operators are used during thesaurus expansion as are used during search, you can use the qparser keyword in your control file to specify a query parser. For example:

```
qparser: simple
```

Creating a Control File from an Existing Thesaurus

The **mksyd** command-line tool is primarily used to compile a thesaurus from a control file (see "Compiling a Thesaurus with mksyd," later in this appendix), but you can also use it to de-compile (export) a thesaurus, turning it back into a control file.

The easiest way to create a custom thesaurus in a locale for which you already have a thesaurus is to export the thesaurus to a text file, modify it, and then recompile it as a .syd file.

Existing thesaurus files are stored in the directory *verity_product*/common/*locale_name*, where *verity_product* is the directory containing the component of Verity that has been installed (for example, usr/verity/K2 for K2 Services), and *locale_name* is the name of the thesaurus's locale (for example, frenchx).

To use **mksyd** to create a control file from an existing thesaurus file, execute this command from within the directory that holds the existing thesaurus file:

```
mksyd -locale locale_name -charmap charset -dump -syd vdk30.syd -f ctrl_file.ctl
```

where

- *locale_name* is the name of the locale whose thesaurus you are de-compiling.

  (This option is not required if the thesaurus is in the default locale.)

- *charset* is the character set you want the control file to use. It must be one of the character sets supported by *locale_name*, as listed in "Verity Locales and Character Sets" in Appendix A.

  (This option is not required if the control-file's character set is to be the default character set for *locale_name*.)

- vdk30.syd is the name of the thesaurus file that you want to de-compile.

- *ctrl_file*.ctl is the name you want to give to the control file (note the extension .ctl).

The resulting file is in control-file format:

```
$control: 1
synonyms:
{
  list: "word1, synonym1-1, synonym1-2, synonym1-3"
  list: "word2, synonym2-1, synonym2-2, synonym2-3"
  list: "word3, synonym3-1, synonym3-2, synonym3-3"
...
}
$
```

You can then edit the control file as needed and re-compile it as explained next.

Compiling a Thesaurus with mksyd

After you have created a thesaurus control file, you can use the **mksyd** command-line tool to compile it into a thesaurus. The control file must have the file-name extension `.ctl`.

Execute the following command from within the directory that holds the thesaurus control file:

```
mksyd -locale locale_name -charmap charset -f control_file.ctl -syd vdk30.syd
```

where

- *locale_name* is the locale of the thesaurus, which must be the locale of any collections that the thesaurus is to be used with.

  (This option is not required if the thesaurus is in the default locale.)

- *charset* is the character set of the control file. *charset* must be a character set supported by the locale *locale_name*.

  The *charset* option is optional; leave it off if the control file's character set is the internal character set of the thesaurus's locale. For a list of the supported character sets and internal character set for each locale, see "Verity Locales and Character Sets" in Appendix A.

- *control_file* is the name (minus file extension) of the control file to compile.

- `vdk30.syd` is the name of the thesaurus file that you want to create.

Integrating the Thesaurus with Verity

Once you have created a new thesaurus, it should be placed in the appropriate directory for use.

Naming and Installing the Thesaurus

First, the thesaurus file must have the appropriate filename. Regardless of its locale, every thesaurus file must be named `vdk30.syd`.

> NOTE: Only one active thesaurus file is allowed per locale. Only one `vdk30.syd` file can be present in a *`locale_name`* directory. If you are creating a thesaurus for a locale (like `englishx`) that has a default thesaurus provided by Verity, move the default thesaurus from the *`locale_name`* directory, or else rename it, before adding your new thesaurus. It is recommended that you do not permanently remove the default thesaurus.

To integrate your custom thesaurus into your search application, move the compiled thesaurus file to the locale's directory:

`verity_product``/common/`*`locale_name`*

where *`verity_product`* is the installation directory (such as `/usr/verity/k2`) of your Verity component, and *`locale_name`* is the name of the directory (such as `dutchx`) containing the locale for which you are creating the thesaurus.

WARNING! All application processes, including user searches, must be terminated before you remove or change the contents of the `common` directory or any of its subdirectories. The new thesaurus will be available when the application is started or restarted.

Using a Knowledge Base Map to Point to a Thesaurus File

You can also use a knowledge-base map to point to a `.syd` File. This is a sample map file:

```
$control
kbases:
{
kb: "Thesaurus"
/kb-path = "vdk30.syd"
}
```

In K2, point to this map file either through the client in a local context, or through the server configuration file in a remote context.

Note that no thesaurus operator is necessary in queries using a knowledge-base map. The query works like a topic, so any word in the thesaurus that you enter will automatically map to its synonym list.

D

Glossary

Term	Definition
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**.
asymmetric list	In a thesaurus control file, a list of synonyms in which only some entries are keys. Compare **symmetric list**.
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*.

Term	Definition
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.
circular list	In a thesaurus control file, a list of synonyms in which all entries are keys. Compare **asymmetric list**.
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 `japanb` locale, 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**.

Term	Definition
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**.
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).

Term	Definition
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.
key	In a list of synonyms in a thesaurus control file, a word that must appear in the search query for any of its synonyms to be found.
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 language-identification command-line tool.
language-identification command-line tool	A Verity tool that opens a document and returns a language assignment.
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.
mksyd	A command-line tool used to build a thesaurus from a thesaurus control file.
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.

Term	Definition
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.
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.

Term	Definition
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.
style.zon	A filter style file that controls functioning of the zone 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	The locale `englishx`. 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.
thesaurus control file	A text file containing lists of synonyms. You create a thesaurus control file, then you use the **`mksyd`** command-line tool to compile it into a thesaurus.
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.

Term	Definition
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.
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 recognition, and thesaurus use. See also **single-language 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

E

F

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X