Verity K2
Getting Started Guide

Version 6.0

August 19, 2005
Part Number DM0677

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

Verity Benelux BV
Coltbaan 31
3439 NG Nieuwegein
The Netherlands

Copyright 2005 Verity, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of the copyright owner, Verity, Inc., 894 Ross Drive, Sunnyvale, California 94089. The copyrighted software that accompanies this manual is licensed to the End User for use only in strict accordance with the End User License Agreement, which the Licensee should read carefully before commencing use of the software.

Verity®, Ultraseek®, TOPIC®, KeyView®, and Knowledge Organizer® are registered trademarks of Verity, Inc. in the United States and other countries. The Verity logo, Verity Portal One™, and Verity® Profiler™ are trademarks of Verity, Inc.

Portions of this product Copyright 2003, Sun Microsystems, Inc. All rights reserved. Use is subject to license terms. Sun, Sun Microsystems, the Sun logo, Solaris, Java, the Java Coffee Cup logo, J2SE, and all trademarks and logos based on Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Xerces XML Parser Copyright 1999-2000 The Apache Software Foundation. All rights reserved.

Microsoft is a registered trademark, and MS-DOS, Windows, Windows 95, Windows NT, and other Microsoft products referenced herein are trademarks of Microsoft Corporation.

IBM is a registered trademark of International Business Machines Corporation.

WordNet 1.7 Copyright © 2001 by Princeton University. All rights reserved

Includes Adobe® PDF. Adobe is a trademark of Adobe Systems Incorporated.

Portions of this product use Teragram Software.

Includes IBM's XML Parser for C++ Edition.

Includes software developed by the Apache Software Foundation (http://www.apache.org/).

This product may incorporate intellectual property owned by Microsoft Corporation. The terms and conditions upon which Microsoft is licensing such intellectual property may be found at

http://msdn.microsoft.com/library/en-us/odcXMLRef/html/odcXMLRefLegalNotice.asp?frame=true

All other trademarks are the property of their respective owners.

Notice to Government End Users

If this product is acquired under the terms of a **DoD contract:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of 252.227-7013. **Civilian agency contract:** Use, reproduction or disclosure is subject to 52.227-19 (a) through (d) and restrictions set forth in the accompanying end user agreement. Unpublished-rights reserved under the copyright laws of the United States. Verity, Inc., 894 Ross Drive Sunnyvale, California 94089.

Contents

Figures, Tables, and Listings

Preface

This book is for readers who want to learn about Verity K2 technology. It explains Verity's approach to searching, classifying, and personalizing large bodies of enterprise information, for the purpose of maximizing return on intellectual capital investment.

This preface contains the following sections:

Using This Book

Welcome to the *Verity K2 Getting Started Guide*. This document introduces K2 services and K2 applications. With Verity K2, enterprise employees can discover, organize, and connect with the information critical to their jobs.

This book covers all areas of K2. It should be the starting point for understanding how to use, manage, and develop K2 applications.

Version

The information in this book is current as of K2 version 6.0. The content was last modified August 19, 2005. Corrections or updates to this information may be available through the Verity Customer Support site; see "Verity Technical Support" on page 16.

Organization of This Book

This book includes the following chapters and appendixes:

- **Chapter 1, "Introducing Verity K2."** Describes the capabilities of K2 and outlines some of the many benefits that this technology offers to your organization.
- **Chapter 2, "Setting Up a Verity K2 Installation."** Explains the steps to design and set up an installation of K2 in your enterprise.
- **Chapter 3, "Indexing and Classifying Information."** Describes how to implement the powerful information-access and analysis capabilities in a K2 installation.
- **Chapter 4, "Delivering Information to Users."** Describes how to implement the powerful search and information display capabilities of K2 in your application.
- **Chapter 5, "Developing Your Application."** Describes how to create and use a K2 application to search documents, browse classified information, and connect to communities of experts.
- **Appendix A, "Contacting Verity Technical Support."** Describes how to get help with your questions about Verity products and features.
- **Glossary**. Defines Verity terms and related industry terminology.

Stylistic Conventions

The following stylistic conventions are used in this book.

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

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

Convention	Usage
`[ optional ]`	Brackets describe optional syntax, as in `[ -create ]` to specify a non-required option.
|	Bars indicate "either | or" choices, as in `[ option1 ] | [ option2 ]` In this example, you must choose between `option1` and `option2`.
`{ required }`	Braces describe required syntax in which you have a choice and that at least one choice is required, as in `{ [ option1 ] [ option2 ] }` In this example, you must choose `option1`, `option2`, or both options.

Convention	Usage
`required`	Absence of braces or brackets indicates required syntax in which there is no choice; you must enter the required syntax element.
`variable`	Italics specify variables to be replaced by actual values, as in `-merge` *`filename1`*
`...`	Ellipses indicate repetition of the same pattern, as in `-merge` *`filename1`*`,` *`filename2`* `[,` *`filename3`* `... ]` where the ellipses specify `,` *`filename4`*, and so on.

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

Verity K2 Documentation

Verity K2 offers an extensive set of interrelated technologies, tools, and programming interfaces. To support these far-reaching capabilities, Verity provides the documents listed in Table PR-1, in two broad categories: administrator and knowledge worker documents and developer documents.

Table PR-1 K2 documentation

For administrators and knowledge workers	For developers
Getting started: ■ Verity K2 Getting Started Guide **Installing and configuring K2:** ■ Verity K2 Installation and Setup Guide ■ Verity K2 Migration Guide ■ Verity Locale Configuration Guide	**Getting Started:** ■ Verity Developer Getting Started Guide **Administration:** ■ Verity K2 Administration Prog. Guide ■ (VAdministration JavaDoc)

Table PR-1 K2 documentation (continued)

For administrators and knowledge workers	For developers
Indexing and managing collections: ■ Verity K2 Dashboard Admin. Guide ■ Verity Collection Reference ■ Verity Command-Line Indexing Ref. *Gateway Guides:* ■ Verity K2 Lotus Notes Gateway Guide ■ Verity K2 ODBC Gateway Guide ■ Verity K2 Documentum Gateway Guide ■ Verity K2 Exchange Gateway Guide **Classifying and personalizing information:** ■ Verity Intelligent Classification Guide ■ Verity Business Console Guide ■ Verity K2 Recommendation Eng. Guide ■ Verity K2 Parametric Developer Guide ■ Verity K2 Profiler Programming Guide ■ Verity Query Lang. and Topic Guide	**Search applications:** ■ Verity K2 Client Programming Guide ■ (VSearch JavaDoc) ■ Verity K2 Viewing Service Prog. Guide ■ (VView JavaDoc) ■ Verity Federator Programming Guide ■ Verity Query Language and Topic Guide ■ Verity Web Services Guide **Indexing and classification apps (K2 Layer):** ■ Verity K2 Collection-Indexing Prog. Guide ■ (VIndex JavaDoc) ■ Verity K2 Parametric Developer Guide ■ (VParametric JavaDoc) ■ Verity Organization Developer's Kit Guide ■ Verity K2 Profiler Programming Guide **Indexing and classification apps (VDK Layer):** ■ Verity Organization Developer's Kit Guide ■ Verity Developer's Kit Prog. Reference ■ Verity Extractor Programming Guide ■ Verity Profiler Programming Guide ■ (Profiler JavaDoc) **DDA Layer Programming:** ■ Verity Gateway Developer's Kit Reference **OEM Deployment:** ■ Verity OEM Deployment Guide

Verity Technical Support

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

Telephone: (403) 294-1107

Fax: (403) 750-4100

Email: tech-support@verity.com

Web: http://www.verity.com

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

https://customers.verity.com

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

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

1

Introducing Verity K2

Information is the key to business success, but you must be able to find the right information before you can act on it. Verity's K2 infrastructure enables enterprise employees to discover, organize, and connect with the information critical to their job success.

Verity K2 is designed for building knowledge-management applications and business portals that provide customized access to corporate information and leverage investments in intellectual capital. K2 delivers these capabilities with powerful features such as full-text search, parametric selection, intelligent classification, adaptive ranking, and recommendations.

This chapter contains the following sections:

Intellectual Capital Management With Verity K2

Enterprises of all sizes throughout the world are facing the need to better manage their information. The intellectual capital of an organization—its stored information and data, along with the knowledge and experience of its employees—is now recognized as a critically important resource that has traditionally been difficult to locate and exploit.

Studies have shown that knowledge workers today spend approximately 35% of their productive time searching for information online, yet nearly 40% of corporate users cannot readily find the information they need to do their jobs. Employees who waste precious time searching for information can cost a large corporation hundreds of millions of dollars annually.

Intellectual capital management addresses these problems. It combines human knowledge and experience with the information and data in an enterprise for the purpose of exploiting greater value. It allows people in your organization to more effectively locate information that they know exists but aren't sure where, and also uncover unknown information assets. With that improved access they can respond quickly and accurately to questions, and they can better evaluate content to make better decisions.

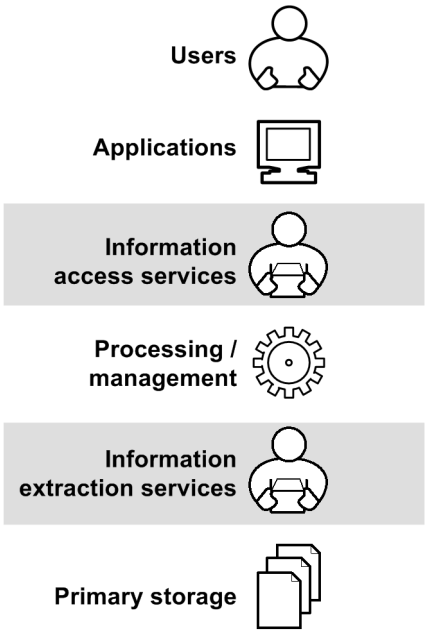
Verity K2 contains technologies that help your users succeed in these tasks. Together, the technologies are called *intelligent content services*. The services include search, classification, recommendation, profiling, entity extraction, and other activities that can maximize the usefulness of your organization's information.

Verity is the recognized leader in intellectual capital management software. The open design of Verity K2 and its support for universal standards ensures straightforward integration with your existing applications. Its built-in multilanguage capabilities give international access to information. Its scalable architecture gives your business unlimited growth potential and exceptional fault tolerance while maintaining full security. This means users anywhere in your organization, anywhere in the world, can make the most of your information assets.

The Intelligent Content Services Architecture

K2 is an implementation of Verity *Intelligent Content Services*, an integrated set of technologies that locate, extract, and analyze information to aid decision making. When K2 is installed in a production environment and applications have been developed to use its capabilities, the services fit into an overall architecture like that shown in Figure 1-1.

Figure 1-1 Components of Intelligent Content Services

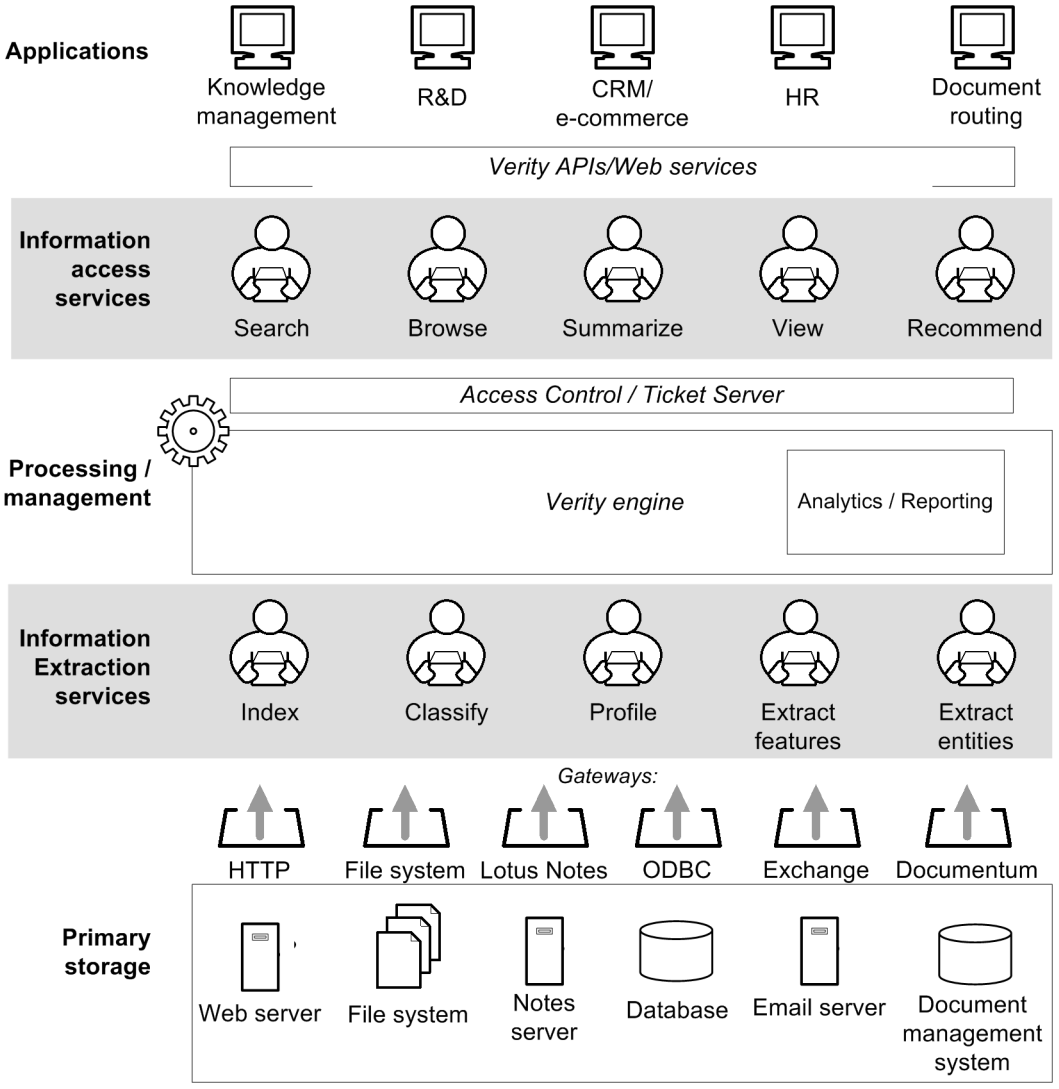


A user accesses an application that makes use of one or more Verity *information-access services*. These services allow the application to assist the user in searching for or browsing through information, thus helping the user to learn facts, form opinions, or make decisions or purchases. A search interface is one example of an application UI built on information-access services.

The information presented to the user is processed through the Verity engine, a set of core libraries that manage not only the information access services but also the *information extraction services*, which locate raw information in its primary storage form, then process it into a variety of indexed, organized, and classified states that the information-access services can use. A K2 collection is one possible result of applying information extraction to primary data.

Figure 1-2 expands the picture of the Intelligent Content Services architecture to show some specific examples of supported application types, information-access services, information-extraction services, and primary storage types. This figure is not exhaustive; K2 includes services not shown here.

Figure 1-2 Intelligent Content Services architecture



The rest of this section explains the K2 architecture briefly, from the bottom up.

Multiple Primary Storage Formats

The lowest level of the Intelligent Content Services architecture consists of unstructured, semi-structured, and structured data belonging to a given organization or enterprise.

The typical enterprise has documents residing in a variety of content sources and databases. K2 provides access to documents in the following types of information *repositories*:

- File systems (Windows and UNIX)
- Web sites
- Lotus Notes servers
- Email servers (Microsoft Exchange)
- Databases (ODBC)
- Document-management systems (Documentum)

Documents can exist in a myriad of file formats ranging from Microsoft Office documents to Lotus Smartsuite to Adobe PDF. K2 utilizes state-of-the-art technology to read documents in over 200 such formats, extracting the structured and unstructured content from the documents. Documents may exist in many languages with different *character sets*—sets of numeric codes based on the characters of a language—and have attachments in multiple MIME-types and languages. K2 can support close to 100 languages and popular character sets including the Unicode standard.

K2 makes documents from this wide variety of sources available for searching by processing them into Verity *collections*, index structures that support extremely rapid and flexible search capabilities over very large numbers of documents.

Secure Access Through Gateways

Verity *gateways* are software modules that allow K2 to access information in a variety of different primary storage formats. Each gateway is tied to a particular type of repository (such as HTTP, Lotus Notes or Microsoft Exchange); the gateway can access documents and metadata in the repository's native formats.

A user's role in the organization dictates which documents the user is allowed to access. For example, an employee outside of the Human Resources department should presumably get a "No Results Found" message on the query "layoffs next week" even when a matching document exists that is accessible to Human Resources department employees. K2 supports multiple levels of secure search implementations ranging from

the stringent role-based security in the above example, to weaker forms in which the existence (but not the contents) of all matching documents is revealed to the user regardless of his access rights.

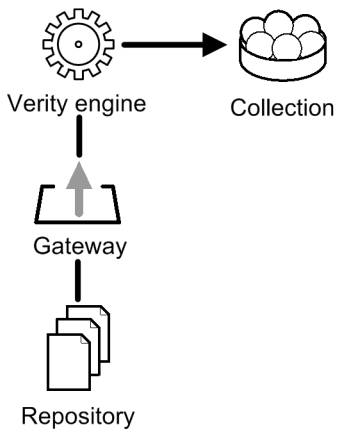
Information-Extraction Services

The core capabilities of Verity K2 focus on accessing documents in heterogeneous repositories to extract information and build data structures needed by information-access applications.

Administrators and knowledge workers (or indexing applications that use the verity APIs) can use these information-extraction services to create collections and other types of indexes that support search and information access.

Figure 1-2 on page 20 shows some of the available information-extraction services. More information on each is available elsewhere in this book.

- **Indexing**. Before users can search or classify enterprise information, it generally must be indexed. The K2 system uses gateways and the Verity engine to gather information into a universal index called a *collection*.



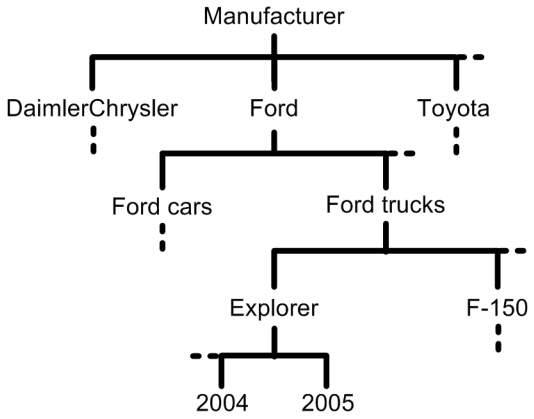
  The main purpose of a collection is to support sophisticated, multi-featured text search. A collection stores the locations of all indexed documents and a list of essentially all words contained within the text of those documents. (A collection does not contain the actual documents themselves.)

- **Classification**. Beyond search, information access can also involve classification, in which documents are organized into one or more *taxonomies*, which are browsable,

searchable, hierarchies of categories. Verity K2 includes several information-extraction services related to creating and populating taxonomies.

- *Parametric indexes*. Parametric indexes are structures built on top of collections or other sets of documents. These indexes support both *parametric search*, the ability to search heterogeneous collections of documents containing both structured metadata (parameters) and free text, and *taxonomy browse*, the ability to navigate through a taxonomy of category links to arrive at desired documents.

  For example, documents that describe cars might have structured attributes such as Color, Price, Make, Model, Location, and Year, plus free-text descriptions of the cars. Also, a taxonomy applied to the descriptions might organize the documents by manufacturer, make, model, and year.



  The parametric-selection portion of a parametric search would involve the user selecting the desired attributes from the structured data. The free-text portion would involve searching for terms in the descriptions. Taxonomy browse would involve navigating through the hierarchy of links to find the desired model and year. (See, for example, Figure 4-1 on page 107.)

- *Relational taxonomies*. Verity parametric indexes can further include a high-level classification concept called *relational taxonomies.* With relational taxonomies, more than one taxonomy is applied to a parametric index. Users can simultaneously navigate the multiple taxonomies, drilling down and jumping from one to the other, navigating to the information they seek in the manner most intuitive to them. (See, for example, Figure 4-3 on page 109.)

- *Thematic Mapping* is a process that automatically extracts key concepts from a set of documents, constructs a taxonomy from them, and assigns the documents to the taxonomy to create a parametric index. (See "Building the Taxonomy" on page 85.)

- **Profiling**. Using the K2 Profiler, an application can automatically classify incoming documents, assigning them to one or more categories based on criteria such as subject areas of interest to specific users.

  The categories used for profiling are implemented in structures called *profile nets*, which are stored queries that knowledge workers can create manually or with the help of command-line tools. (See "Creating Profile Nets" on page 89.)

- **Feature Extraction**. When displaying search results, a K2 application can provide a summary of each document and it can cluster related documents together on the page.

  Clustering, thematic mapping, and some kinds of summarization rely on an underlying process called *feature extraction*, in which the most important key words and concepts in a document are automatically extracted and saved during collection indexing.

  See "Extracting Document Features" on page 80 for more details.

- **Entity Extraction**. K2 includes a component called the Verity Extractor, which is an engine that applications can use to extract *entities*—words or blocks of text that have specific meaning (for example, names, telephone numbers, URLs, addresses, product IDs)—from a document or set of documents.

  Applications can use the results of extraction to populate collection fields or taxonomy categories, to forward the entity information to other analytical programs, or to validate or route documents based on entity rules. See "Extracting Entities" on page 81.

- **Entity Profiling (for Recommendation)**. The Verity Recommendation Engine is a K2 component that brings sophisticated, high-level socialization and personalization capabilities to applications. Recommendation applications can connect users to social networks, not only providing results for search queries but also adaptively ranking results, locating experts, recommending alternative documents, and connecting the user with communities of people with similar interests.

  The K2 Recommendation Engine stores the information it needs in entity profiles (recommendation indexes) that record the activities and preferences of users and the actions taken on various documents and other entities. The information changes over time as the system evolves. See "Providing Recommendations" on page 91.

The Intelligent Content Engine

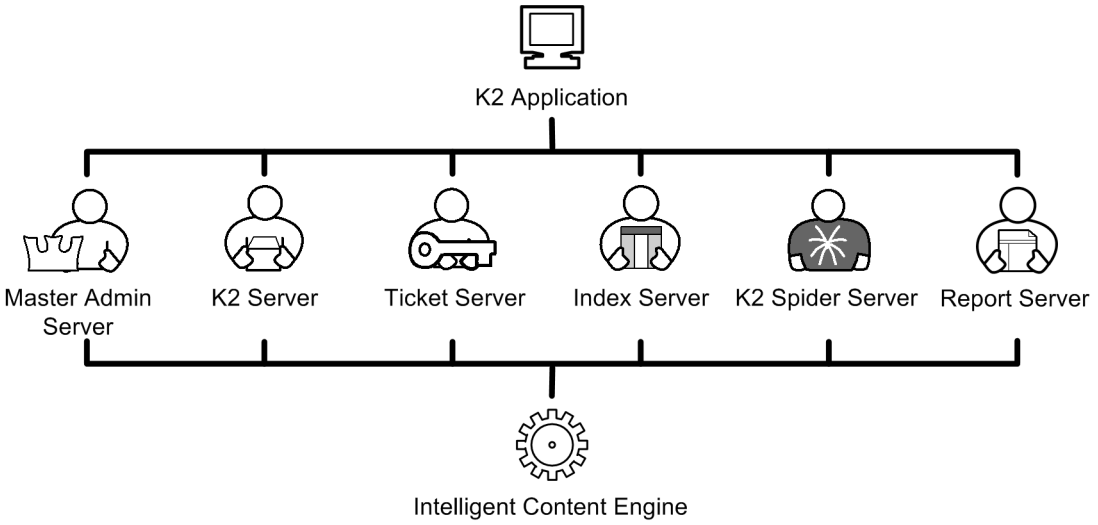
The heart of K2 is the Intelligent Content Engine. The engine consists of several modules that extract information, create structures, and process data to support K2-enabled applications.

Some components of the Intelligent Content Engine include the following:

- The Verity engine, or VDK, which supports collection creation and search.
- The Extractor Engine, which supports entity extraction.
- The Recommendation Engine, which supports recommendation indexes.
- Application-level services for displaying information to users, such as document summaries and spelling suggestions. See "Presenting Search Results" on page 103
- The Viewing service, which displays highlighted content for documents (in any of hundreds of native formats) retrieved from a search results page. See "Retrieving and Displaying Documents" on page 105.

The Intelligent Content Engine includes extensive low-level programming interfaces (APIs) and configuration settings that allow applications to control its behavior closely. A large variety of command-line tools also provide access to the engine for creating and manipulating the information structures it manages.

The Intelligent Content Engine can also be accessed through high-level K2 APIs, exposed on the various distributed services in a K2 system.



These services include Administration Servers (for managing the system and its indexes), K2 Servers and K2 Brokers (for accessing collections and other indexes), Ticket Servers (for secure access), Index Servers (for direct indexing of documents), K2 Spider Servers (for using a spider to crawl and index repositories), and Report Servers (for generating user-activity reports). See "Choose Which K2 Services to Use" on page 42 for more information.

Flexible Access Control

Verity K2 supports secure access to the information in collections and other indexes and to the documents in repositories. K2 allows for multiple levels of security, multiple authentication methods, and single sign-on. See "Setting Up Security" on page 51 for details.

Information-Access Services

At the application level, Verity K2 offers a variety of services that applications can use to give users functional and convenient access to the information that has been extracted, indexed, and classified by the information-extraction services described earlier.

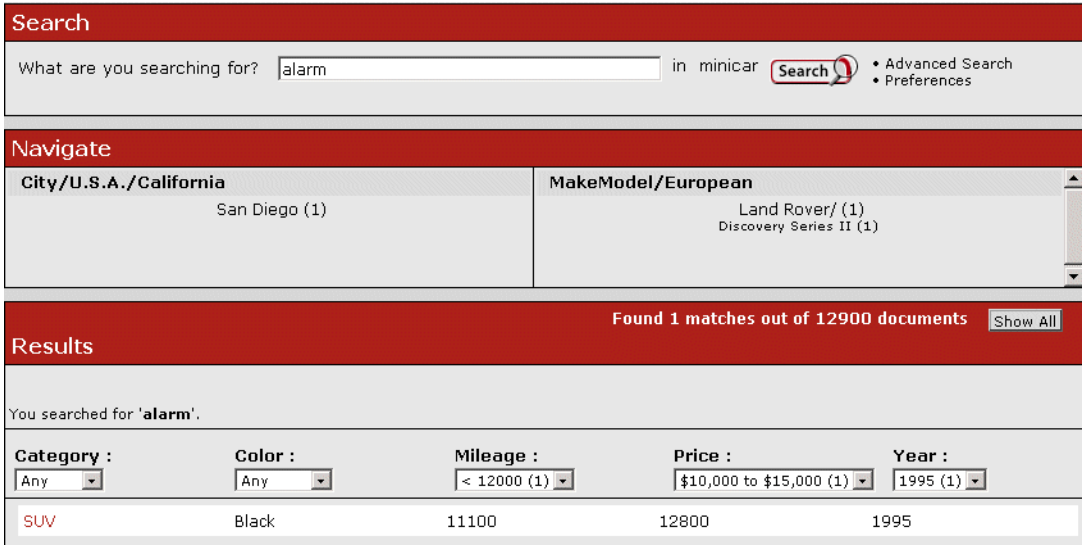
Figure 1-2 on page 20 shows some of the available information-access services. More information on each is available elsewhere in this book.

- **Search**. Sophisticated search is at the core of the information services that K2 applications provide for users. The search capabilities include
  - Full-text search, supporting single or multiple words or phrases, case- or accent-sensitive or insensitive searches, synonyms, wildcards, proximity searches, and logical combinations of them.
  - Fuzzy searches, using automatic re-spelling or stemming of search terms.
  - Language-specific search, in any of a large number of languages.
  - Topic search, in which a single search term can expand into query expression of any size or complexity.
- **Federated Search**. Federated search (licensed separately from K2 as Verity Federator) gives out-of-the-box access to many information sources, including proprietary sources licensed by Verity on behalf of its customers. Federator sends a single query to multiple sources such as internal Verity indexes plus external sources like Web sites and proprietary subscription sources such as news feeds and business information services. Federator then merges results from all of them into a unified presentation for the user.

  Federator is a powerful, standalone application and API that combines federated search with the Verity Ultraseek technology to extend the reach of an enterprise's search capabilities. See your Verity representative for more information.
- **Parametric Search and Taxonomy Browse**. An application that works with information that has been processed into classification structures (parametric indexes and taxonomies) can use sophisticated information-access services such as the following:

- *Parametric selection*, in which the user can find documents by selecting values for various parameters instead of by searching.
- Text search combined with parametric selection, restricting the results to documents that match both the currently selected parameters and the search terms.
- *Taxonomy browse*, in which users navigate through hierarchical categories of information to reach a desired set of results. The results at each stage consist of documents that match the selected taxonomy category, plus the currently selected parameters, plus any search terms that are applied.
- *Relational taxonomies*, in which multiple taxonomies are applied to the data. The user can start browsing one taxonomy, then switch to another, and even back again, until arriving at a desired category or document. Results can be restricted to documents that simultaneously match the selected category in each of the represented taxonomies, plus the currently selected parameters, plus any search terms that are applied.

For example, a user might search for a used car by selecting values for categories such as color, mileage, price, and year, then navigate both geographic and manufacturer taxonomies, and finally search for a specific desired feature (such as a car alarm).



- **Universal Document Viewing**. The Verity K2 viewing service provides applications with a powerful document viewing and highlighting service. When the user clicks a link on the search results page to view a document (in any of hundreds of supported formats), the viewing service displays the document content and highlights occurrences of the search terms throughout the document.

- **Spelling Suggestion**. S*pelling suggestion* can be used to suggest corrections to mistyped words in a user's query. If a search returns no or few results, the application can display a message on the search results page, listing a suggested alternate query. For example, if the user searches for "helo wonderful worlld" and that phrase returns no hits, your application could respond with

  Are you searching for "**hello** wonderful **world**"?

  See "Providing Spelling Suggestion" on page 105 for more information.

- **Document Summarization**. By presenting a short, automatically generated summary for each document in a results list, a K2 application can help users quickly assess the relevance of the returned documents before retrieving the documents themselves.

  Verity K2 supports several kinds of document summaries, including passage-based summaries, which consist of text excerpts in which the search term appears, optionally highlighted. For more information about summarization, see "Returning Document Summaries" on page 104.

- **Document Clustering**. When presenting search results to the user, an application can cluster, or group together, documents covering similar topics or concepts.

  Clustering relies on the extraction of document features at collection-indexing time (see "Extracting Document Features" on page 80).

- **Recommendation**. An application that uses the K2 Recommendation Engine can suggest or recommend documents, expert users, or other entities that are specifically relevant to the current user's context. To do so, the Recommendation Engine uses profiles (recommendation indexes) that are continually updated with users's actions and preferences, meaning that the recommendations can evolve over time.

  Features available with the Recommendation Engine include

  - Adaptive ranking of search results
  - Context-based personalization in user profiles
  - Concept-based retrieval (independent of specific keywords)
  - Location of experts and communities
  - Session-based profiles (updated dynamically within a single session)

  See "Providing Recommendations" on page 91 for more information.

Multilanguage APIs and Web Services

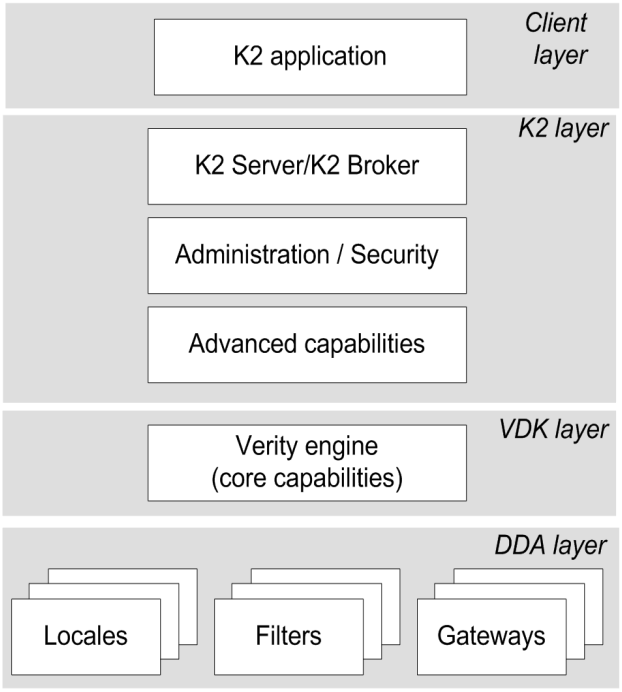
Creating an application that makes use of the information-extraction and information-access services provided by K2 requires development effort. Depending on the scale of your needs, it can be as simple as customizing the look and feel of the client component sample programs included with K2 (see "Using the Component Framework" on page 116), or it can be a large, multi-departmental development project involving many software developers.

Verity K2 supports application development in Java/JSP, C++, and C#. APIs are available for connecting to features at all levels of the software architecture. See "Developing Your Application" on page 115 for more information.

K2 Software Architecture

K2 is an integrated set of software technologies that allows enterprise applications to bring the capabilities of Verity's intelligent content services to users. The software is modular and layered; it includes application-programming interfaces (APIs) at several levels for maximum programming flexibility. Figure 1-3 is a high-level overview showing how the major pieces fit together.

Figure 1-3 Layers of K2 architecture



Client Layer

The topmost layer in this architecture is the client application layer, consisting of a K2 application or portal created by a Verity customer. It typically includes a Web-based user interface that displays search, classification, and personalization capabilities to the end user. The client can be a business portal, Web application, or a third-party application from vendors such as Oracle, BEA or IBM.

K2 Layer

The client application communicates directly with the K2 layer, connecting to either a K2 Server or a K2 Broker (which in turn connects to one or more K2 Servers). The K2 Server/Broker architecture provides distributed, scalable, secure, and fault-tolerant access to Verity's core and advanced capabilities. See "Deployment Architecture" on page 32 for more information on K2 Server and K2 Broker.

The K2 Server makes use of advanced Verity features and capabilities within the K2 layer. These features include:

- Administration, security, and distributed indexing services, as described in "Setting Up a Verity K2 Installation" on page 37.
- Content services such as knowledge tree creation, parametric-index creation, and the use of the Recommendation Engine and Logistic Regression Classifier, as described in "Classifying Information" on page 84 and "Providing Recommendations" on page 91.

VDK Layer

Features in the K2 layer communicate in turn with the VDK layer. (VDK is an abbreviation for "Verity Developer's Kit" or "Verity Development Kernel.") The VDK layer holds the core searching and indexing capabilities of the Verity architecture. These capabilities include:

- Content services such as indexing and collection creation, simple and advanced text search, parametric selection, and document clustering, as described in "Building Collections" on page 71 and "Classifying Information" on page 84.
- The document viewing service, as described in "Retrieving and Displaying Documents" on page 105.

DDA Layer

Features in the K2 layer communicate in turn with the Dynamic Data Access (DDA) layer, the lowermost layer in the architecture. This layer contains driver-level modules that access and process external data. There are three kinds of DDA modules:

- Gateways, which provide read-only access to information repositories (see "Information Repositories" on page 38).
- Document filters, which process documents of different formats read in through the gateways (see "Verity Document Filters" on page 40).
- Locales, which perform character conversion and apply language-specific indexing techniques to document content (see "Locales" on page 55).

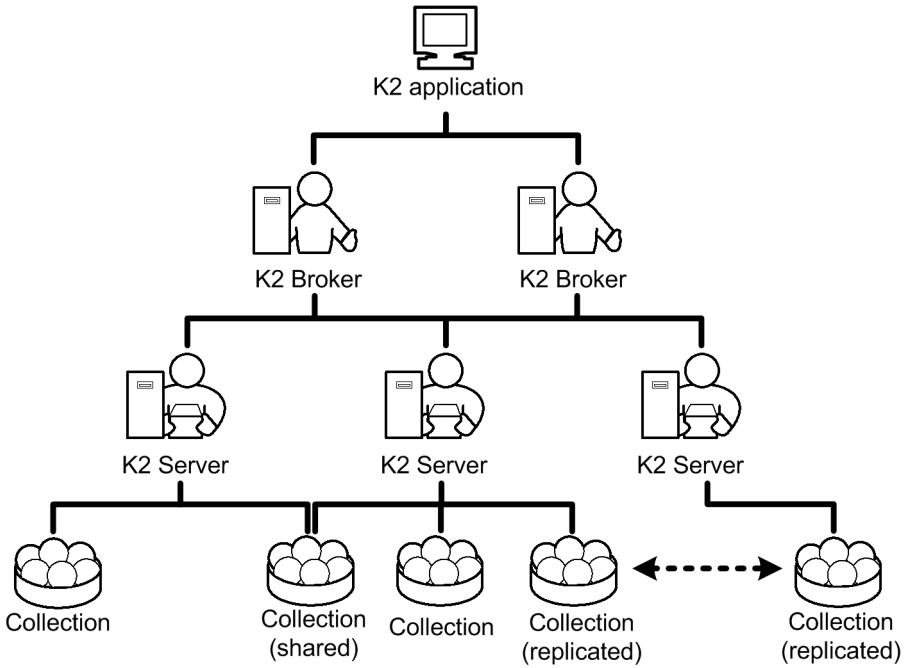
Taken together, these main software layers provide the support for Verity intelligent content services, as shown conceptually in Figure 1-4.

Deployment Architecture

K2 includes a brokered, distributed server architecture that offers fault tolerance, load balancing, and growth potential for your client application. Brokering your users' searches can increase both the amount of information that can be included in a search and the number of simultaneous users that can be supported—with little degradation in performance.

As shown earlier in Figure 1-3 on page 30, the upper layers of the K2 architecture consist of the K2 client, the K2 Broker, and the K2 Server. In a distributed environment, they might interact as shown in Figure 1-4.

Figure 1-4 K2 Distributed Architecture



This example shows a single client potentially accessing many different collections.

- The *K2 application* is a Verity client. It is integrated with K2 through Verity APIs. It provides the search, content organization, and social network functionality to users.

The client can connect directly to a K2 Server, but in a distributed environment the client connects to a K2 Broker.

- The *K2 Broker* is a service that is an intermediary. It receives requests from a K2 client and distributes them to available K2 Servers. Multiple brokers can execute on one machine, or they can be installed on separate machines.

  Each K2 Broker connects to one or more K2 Servers. It distributes the requests it receives among the active K2 Servers that are attached to it. A broker can communicate with all its K2 Servers simultaneously, whether on the same machine or on different machines.

  In cases where the application searches multiple collections, each K2 Server performs its appropriate portion of the task and returns the results to the broker, which is responsible for consolidating the results and returning them to the application.

- The *K2 Server* is a service that receives requests and performs searches of collections, knowledge trees, and parametric indexes. It also provides recommendations. A K2 Server can accept requests directly from a client, but in a distributed environment those requests usually come through a K2 Broker.

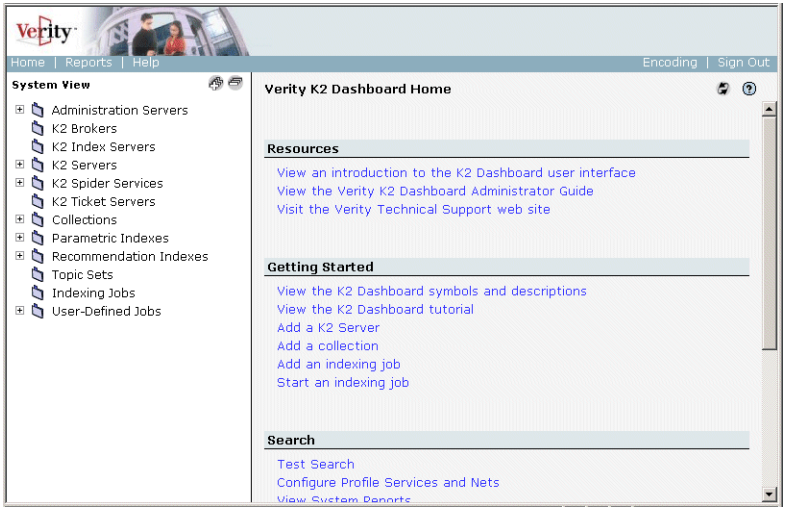
  Each K2 Server connects to one or more Verity collections, which are the index structures that the K2 client searches.

For more information on K2 Server/Broker configurations, see "Designing a K2 Server/ K2 Broker Deployment" on page 42.

Administration Console

Most administration of a K2 installation is performed through a browser-based application called the *K2 Dashboard*. The K2 Dashboard manages K2 services and functionality. Its user interface consists of Web pages (see Figure 1-5). Each page manages a specific function within K2.

Figure 1-5 K2 Dashboard Home page



In the System View pane, The K2 Dashboard gives a visual representation of your distributed K2 system and allows you to configure the system. For example, you can use the Dashboard to control the settings for the following:

- *K2 Services.* Executable processes such as K2 Broker, K2 Server, and K2 Ticket Server.
- *Indexes*. Verity collections, parametric indexes, and recommendation indexes.
- *Jobs*. Collection-indexing tasks and user-defined tasks.

For specific administrative tasks, including batch processing, you can also use one of several Verity command-line tools.

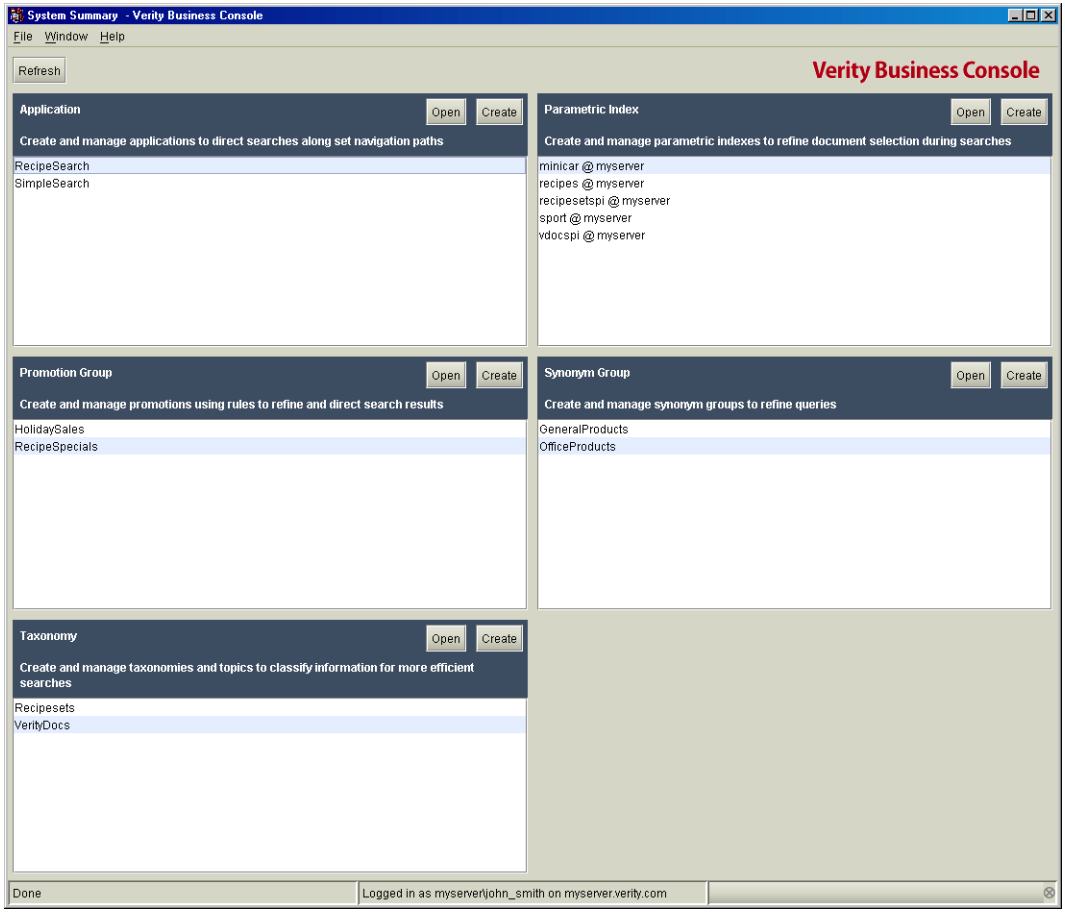
For more information on the K2 Dashboard and the command-line administration tools, see "Administering a K2 Installation" on page 58.

Business Console

Users at a K2 installation can take advantage of the Verity Business Console to create and work with the information extraction and access services that K2 provides. The Business Console is an application that provides a graphical interface for manipulating

taxonomies, parametric indexes, synonyms, and promotions. The Business Console also allows software developers to easily create and configure application components when creating K2 applications with the Verity Component Framework.

Figure 1-6 Business Console Summary page



For more information on the Business Console, see "Managing Information With the Business Console" on page 66.

2

Setting Up a Verity K2 Installation

To set up and use Verity K2, you will need to understand the various components and how they relate to each other. Various people in your organization can help execute these tasks:

- *Administrators* design, install, configure, and maintain the K2 installation. They may also create collections and taxonomies.
- *Knowledge workers* are librarians and domain experts that make decisions about what information sources to make available to users of a K2 installation. They design, create, and maintain collections and taxonomies.
- *Developers* create search applications and implement user interfaces that leverage Verity search, classification, and social-network technologies.
- *End users* use applications to search, browse, and retrieve information.

This chapter contains information primarily of interest to administrators. It contains the following sections:

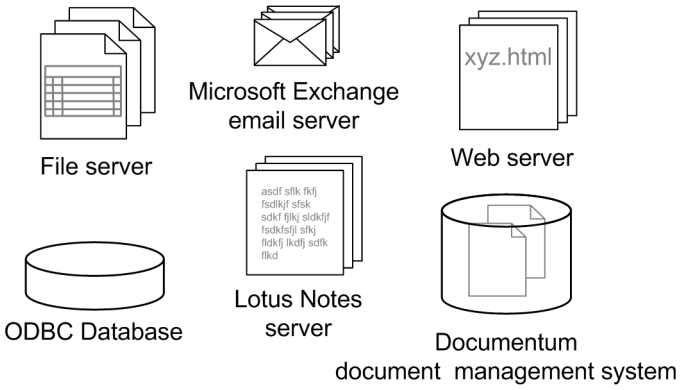
Identifying Information Sources

The first step in setting up a K2 installation is to identify the different kinds of internal and external information that your business produces or accesses. You will need to know how the information you want to analyze is stored.

Information Repositories

An enterprise might store millions of files on dozens of file servers. This same enterprise might also store many thousands of Web pages on multiple Web servers. In each case, files are stored in their native formats on a single type of platform or storage medium, or accessed through a specific connection protocol. The file server represents one type of information *repository,* while the Web server represents another.

Figure 2-1 Examples of information repositories



K2 includes indexing technology that gives you simultaneous access to many types of repositories. You can classify, search for, retrieve, and compare information in locations as diverse as a record in an Oracle database and an XML page on an IIS Web server.

Indexed repository data is the basis for many of Verity's information-management features, including text search, parametric selection, topic sets, and knowledge trees.

Verity Gateways

Verity applications access repositories through *gateways*, driver modules that provide interfaces to specific repository types. Gateways unify your business information by making it all available for indexing and retrieval, regardless of where and how it is stored.

The gateways supplied by Verity with the K2 product include the following:

- **Verity ODBC Gateway**. Provides access to ODBC-compliant databases. The gateway can combine data from any number of databases and tables, such as help desk, sales tracking, or marketing information, so users can view it alongside other enterprise information resources.
- **Verity Lotus Notes Gateway**. Provides connectivity to Lotus Notes repositories. The gateway allows remote or local access, supports encrypted secure Internet passwords, and permits users to search and retrieve information in all views, as well as attachments, OLE objects and encrypted fields.
- **Verity Documentum Gateway**. Provides connectivity to the Documentum eContent Server. The gateway allows users to search and retrieve Documentum content, metadata, and repository-managed properties. It supports hierarchies and relationships such as simple and virtual documents and handles annotations.
- **Verity Exchange Gateway**. Provides secure access to documents in Microsoft Exchange public folders through an Exchange MAPI client. The gateway allows authorized users to search and retrieve information in e-mail attachments and public folders.
- **Verity HTTP Gateway**. Provides simultaneous accesses to information on multiple Internet and intranet Web sites. The gateway allows exploration of all CGI-compliant Web servers. It supports proxy and firewall authentication, HTTPS/SSL and various login methods.
- **Verity File System Gateway**. Provides access to information on UNIX and Microsoft NTFS file systems. It supports local access, as well as remote mounted, mapped, or UNC access.

Note Additional types of gateways may be available through Verity Professional Services.

If your business stores information in repositories other than those described here, Verity offers the *Gateway Development Kit (GDK)*, which includes a set of APIs for writing customized gateways for unique repositories. These APIs allow you to build new gateways or modify existing gateways to accommodate your specific features. See the *Verity Gateway Developer's Kit Programming Reference* for more information.

Verity Document Filters

Accessing a file in a repository is only the first step toward indexing its information. Repositories store documents in hundreds of native file formats. Therefore, Verity also supplies *document filters*, driver modules that can detect, open, and extract the text from files in hundreds of the most popular file types, including

- Word processing files, such as Microsoft Word, Lotus Word Pro and Corel WordPerfect
- Spreadsheet documents, such as Lotus 1-2-3, Corel QuattroPro and Microsoft Excel
- Presentation files, such as Corel Presentations, Microsoft PowerPoint, and Lotus Freelance
- Adobe Acrobat PDF
- HTML and XML
- Entity-extraction filter

Document filters not only extract the textual content of documents for indexing, they can also extract field information, such as the title or author of a text document. (Field information from a file is indexed separately from its text content.) The entity extraction filter goes even farther; it extracts entities (such as names or addresses) from a document's regular body text and saves them in collection fields.

Data Sources

Depending on the nature of your organization's information and the kinds of tasks you want to perform, you may find that different sources are most appropriate for different kinds of information structures.

Sources for Collections

Information from repositories that passes through gateways and document filters is indexed into Verity collections, where it is made available for search by K2 applications. The best sources for searchable data are any of the repositories that hold your enterprise's unstructured and semi-structured information. For example, all word-processing documents, spreadsheets, memos, emails, discussion threads, and presentation documents related to a given project can be indexed together into a single collection related to the project.

More structured data that is searchable on its own (such as database information) might also be beneficial to index into a collection, so that it will be searchable from within a K2 application, along with all the other document types.

Sources for Parametric Indexes

Parametric indexes support parametric selection, in which the user can select certain parameters—for example, color or manufacturer or price range in the case of an automobile—to narrow the scope of a search.

In parametric selection, each parameter that a user can choose must be related to a collection field or XML element. Therefore, the kinds of data sources most useful for parametric selection are those that are semi-structured—that is, documents in which a significant amount of information can be turned into fields:

- Database records
- HMTL catalog pages
- XML files
- Spreadsheet documents

XML files are useful also because you can build parametric indexes on them directly, without first indexing them into a collection.

Sources for Entity Extraction

Entity extraction is the process of recognizing and capturing small-scale text structures—names, addresses, dates, numeric or monetary values, and the like—from unstructured text. Given that, the best sources to which to apply Verity Extractor might be documents that are entity-rich. Documents such as telephone directories, employee lists, activity logs, ledgers, account statements, and transaction records might be especially promising candidates for entity extraction.

On the other hand, the purpose might not be to extract large numbers of entities, but to instead locate only entities of a specific kind within a large body of documents. In that case, the entity extraction can be applied to any kind of readable unstructured or structured document.

Sources for Recommendations

If a K2 application is set up to recommend documents, experts, or other entities to a user, those recommendations are based on historical searching behavior and feedback provided by that user and other similar users. The recommendations are dynamic; they come from the interaction between users and documents.

Therefore, the kinds of information sources you might want to use for recommendations include the following:

- All your indexed enterprise data that is available for searching.

- Published data authored by your users. If your organization uses a document management system, you can use Recommendation Engine APIs to import users' documents and other information from it.
- Emails, from which the Recommendation Engine can extract author and content information for updating user profiles.
- Employee information.

Designing a K2 Server/K2 Broker Deployment

The architecture on which K2 is built uses a flexible component-based design. This design allows you to construct distributed systems at any scale, with brokering, serving, security, administration, indexing, searching, and viewing services arranged in the optimal configuration for your environment, even if you have a mix of platforms.

Choose Which K2 Services to Use

K2 services are implemented as multithreaded server processes. Depending on your organization's needs, you may want to use from a few to all of the available K2 services. At installation, you can choose which services are to be installed on a given host machine. The available service include the following:

- **K2 Servers**. These servers are the main service providers of K2. They allow applications to access document repositories to create and update collections and other indexes, to retrieve documents for viewing, and to support profiling and entity extraction. They give applications access to collections and indexes to support searching, parametric selection, recommendations, and highlighting of search terms in viewed documents.
- **K2 Brokers.** These servers support scaling of a K2 installation to deployment of large numbers of K2 Servers and many collections and indexes. Applications can access a small set of K2 brokers, which can in turn access many K2 Servers, distributed across many hosts and around the world.
- **Ticket Servers**. If your installation needs to support secure access to collections and documents, you can install a K2 Ticket Server to handle security in a K2 domain.

- **K2 Spider Servers**. If collection indexing is to be performed by using K2 Spider to crawl and index the directories of your repositories, you need to install one or more K2 Spider Servers.
- **Index Servers**. If any collection indexing is to be performed directly—that is, by providing K2 with the explicit locations of documents to index—you need to install a K2 index Server.
- **Report Servers**. If you plan to make use of the K2 reporting service, which analyzes user search activity in a k2 domain, you need to install a K2 report Server.
- **Administration Servers**. Every host machine in an installation needs an Administration Server, which is installed automatically whenever any K2 component is installed. Administration servers manage communication among the K2 components in an installation.
- **Master Administration Server**. Each K2 domain has a single Master Administration Server, which controls the administration of the domain. The K2 Dashboard administration tool is installed on the host that holds the Master Administration Server.

Each of these kinds of services is described in more detail in the *Verity K2 Dashboard Administrator Guide*.

Flexible Configuration

The way you configure brokers and servers depends on your business needs. Brokering enables you to scale the system as your user base and the amount of information being searched grows. As more users submit simultaneous queries, you can add brokers to balance the load. Likewise, as the number of documents to search grows, you can add K2 Servers to accommodate the increased workload. You add brokers and servers only as you require them, rather than paying for excess capacity before you need it.

Multiple Brokers and Servers

Consider an enterprise that stores millions of documents on different file systems in North America and Asia. These documents are searched by thousands of users simultaneously. The enterprise can use one K2 Broker to consolidate these user queries and deliver them to a K2 Server, or it can arrange multiple brokers to speed up the process. You can also configure each broker to distribute search requests to multiple K2 Servers.

By allowing processes to be distributed across K2 Servers, you eliminate the risk of any single point of failure taking your entire K2 installation offline. If a K2 Server experiences a hardware failure, queries are automatically brokered to the remaining K2 Servers. This results in uninterrupted service 24 hours a day, seven days a week.

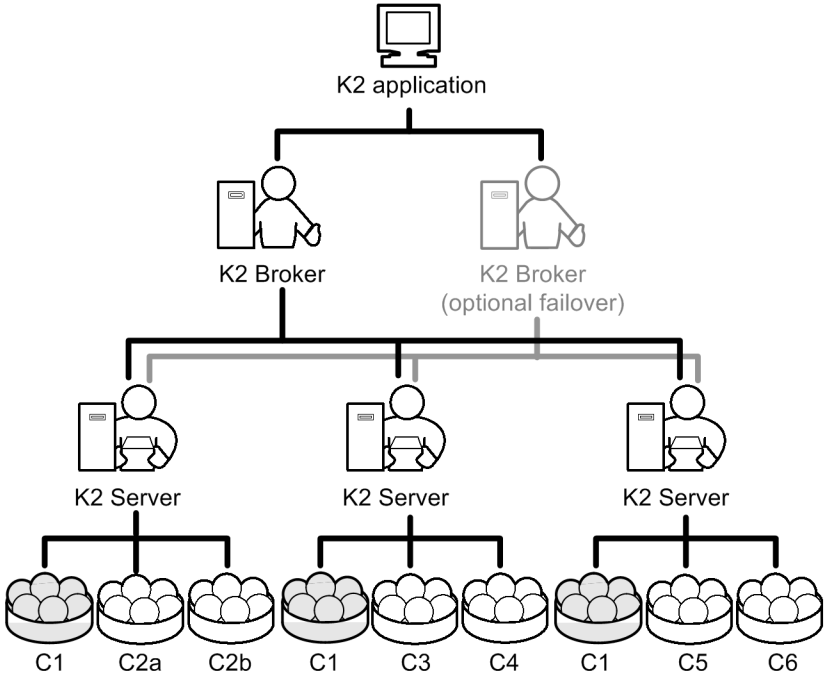
Parallel Querying

The K2 infrastructure supports *parallel querying*, in which a single search query can be distributed simultaneously to multiple collections.

In the case of a single K2 Server attached to multiple collections, the K2 Server simultaneously processes the request against each collection. The K2 Server then merges the results for display to the user.

In the case of a larger installation, separate K2 Servers may be involved. For example, consider a company that must provide fast and accurate search results across many collections in its large corporate intranet. Its documents reside in six separate collections: C1 through C6 (Figure 2-2). Assume that an application needs to search collections C3, C4, and C5. In that case, the K2 Broker distributes the search to the two appropriate K2 Servers. One server distributes the search to collections C3 and C4; the other server searches collection C5. All three searches are conducted simultaneously. Final merging of the results is performed by the K2 Broker.

Figure 2-2 Parallel querying, mirroring, and failover in a large deployment



Mirroring

You can configure a K2 installation to use *mirroring*, in which a collection is duplicated and attached to separate K2 Servers. A K2 Broker can then distribute searches and document retrievals among the mirrored collections in a load-balanced fashion.

As Figure 2-2 shows, the company in this example has decided to mirror its most heavily used collection, C1, on all three K2 Servers. (To enable load-balanced search of mirrored collections, the collection duplicates must be on different K2 Servers and must have identical names.)

This mirrored configuration allows simultaneous access to C1 by many users, distributing any performance impact. The K2 Broker handles incoming requests to search collection C1 by routing the requests among the appropriate K2 Servers.

Additionally, if one of the collections or K2 Servers goes off-line, the broker is still able to access the remaining mirrored collections, without requiring any action by the end user or administrator.

Another advantage of the distributed architecture shown in Figure 2-2 is that K2 Brokers can be duplicated for failover purposes. If the primary K2 Broker goes offline, the application can switch to the alternate and still access all the same information.

Note You can configure K2 to automatically create mirrored collections when indexing. You can also create multiple non-mirrored collections in a single indexing pass (as illustrated by collections C2a and C2b in Figure 2-2), distributing documents between the collections based on document content or other criteria.

Multi-Domain Search Groups

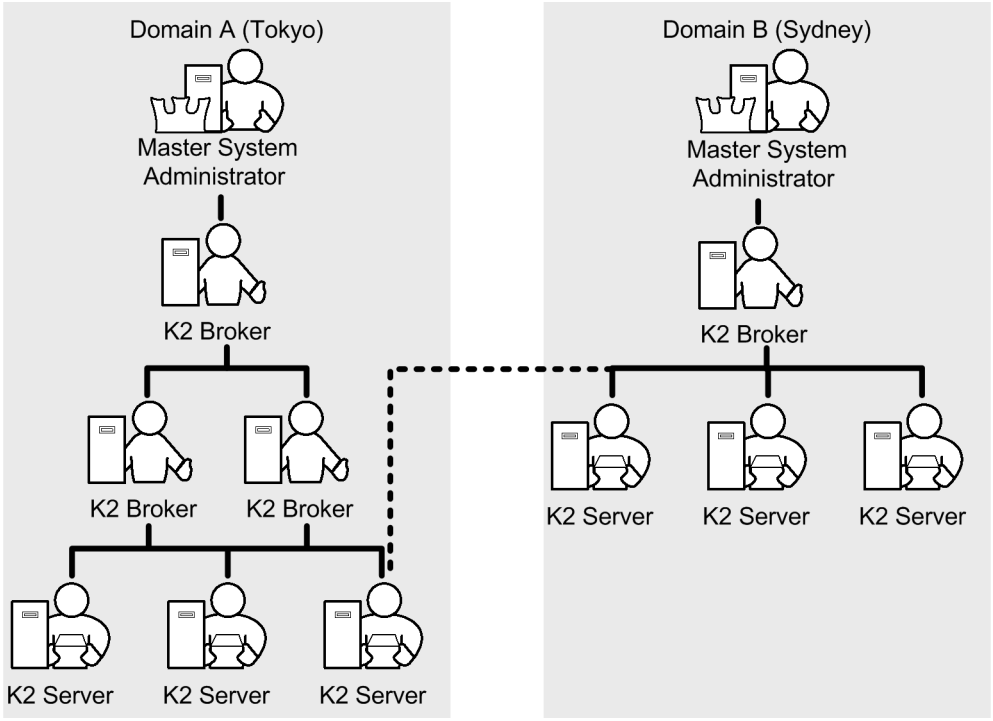
A K2 domain consists of one Master Administration Server and all the K2 services (K2 Ticket Servers, K2 Brokers, K2 Servers, and so on) that are configured by that Master Administration Server. All K2 services are associated with their own administration server. An individual K2 domain typically serves an individual geographic location or an individual department within an organization.

A K2 *search group* consists of one top-level K2 Broker plus all the other K2 Brokers and K2 Servers attached below it. A search request handled by the top level K2 Broker can be passed to any of the other K2 Brokers and K2 Servers in the search group.

It is possible to set up a *distributed search group*, in which some servers or brokers in the group are from a different domain. Figure 2-3 illustrates the concept with two domains: one in Tokyo and one in Sydney. The Sydney search group includes all of its local K2 Brokers and K2 Servers, plus one K2 Server in the Tokyo domain.

Use of K2 search groups that span multiple K2 domains gives administrators flexibility in deploying K2 and in handling search requests.

Figure 2-3 A distributed search group that spans two domains



Installing Verity K2

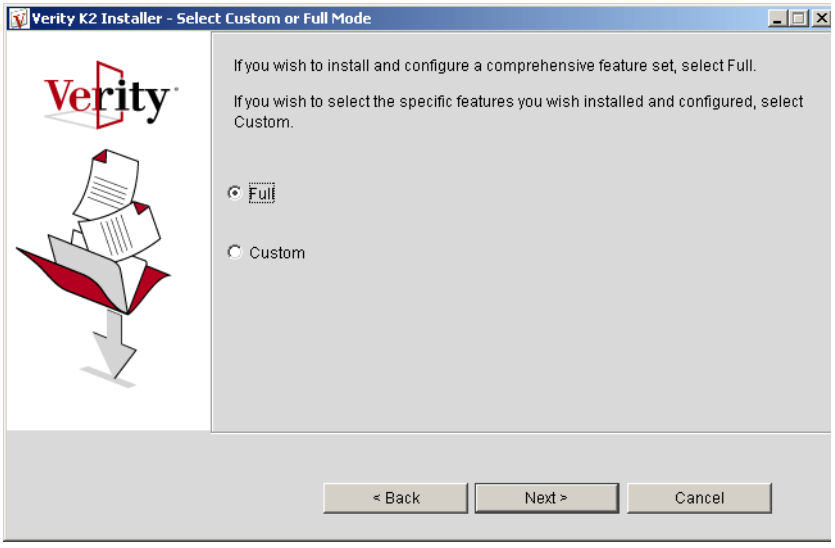
Once you have planned the kinds of information to index and know what configuration you want your K2 installation to have, you are ready to install the K2 software itself. Verity K2 makes it easy for you to install a complete K2 system, or individual components of it, on one or multiple host machines.

For specific instructions on how to install K2 in your enterprise, see the *Verity K2 Installation and Setup Guide*.

Multiplatform Installer

On all supported platforms, you use the same executable installation program to install the components of K2. After launching the installer and entering basic information, including license information, you select the type of installation to perform and are then offered the choice of performing either a **Full** or **Custom** installation, as shown in Figure 2-4.

Figure 2-4 Select Custom or Full Mode dialog from the K2 installer



- A full installation installs all components of K2 (see "Installed K2 Components" on page 49) onto the local host. It creates a single instance each of K2 Broker, K2 Server, and K2 Ticket Server. It also installs a Master Administration Server.

  A full installation is appropriate for development purposes or prototyping.

- A custom installation installs only the components you choose onto the local host.

  A custom installation is required for multimachine installations, such as a large-scale production deployment. In such an installation, you might install K2 Servers and Brokers on many machines. However, you install only one Master Administration Server, and you must install it on the first machine you run the installer on.

Installed K2 Components

You can run the K2 installer to install some or all of the following K2 components on the local host machine.

K2 Services:

- **K2 Broker**. A K2 service that receives client search requests and distributes them to available K2 Servers.
- **K2 Server**. A K2 service that receives requests for searching, viewing, profiling and recommendations, and performs searches of collections, parametric indexes, and recommendation indexes.
- **K2 Ticket Server**. A K2 service that stores an array of users who have been authenticated for access to K2. K2 Ticket Server authenticates users and supports search of secure indexes.
- **K2 Spider Server**. Gathers document information and builds collections, distributing the indexing load across multiple processes and/or machines.
- **K2 Index Server**. A K2 service that supports direct indexing, the process of building a collection through direct submission of documents, rather than use of a spider such as K2 Spider.

K2 Administration:

- **Master Administration Server**. Serves as a central hub for K2 system configuration information. A K2 system must have one and only one Master Administration Server. For each host on which you run the K2 installer, you choose whether to install a Master Administration Server or an Administration Server (next).
- **Administration Server**. A repository for configuration information. In a K2 system, there is one Administration Server for every host.
- **K2 Dashboard.** A Web-based application for administering distributed K2 services from a single host.
- **StyleSet Editor**. Graphical editor for customizing collection style files for different gateways.
- **Business Console.** A Java application that helps knowledge workers and developers to create and manage classification structures and search applications.
- **Application server**. A server that can serve Web pages and Java Server Pages (JSPs). Each K2 installation uses its own embedded application server.

K2 Development:

- **K2 Java Client**. JAVA programming interfaces to support K2 client application development.
- **Client C APIs**. C programming interfaces to support K2 client application development.
- **.NET libraries.** Libraries necessary for .NET K2 client application development in C#, J#, or Visual Basic.
- **Organization Developer's Kit.** Java and C APIs that support development of K2 applications that manipulate taxonomies.

Indexes and Jobs:

- **Sample collections, parametric indexes, recommendation indexes.** These example indexes are installed for learning and testing purposes.
- **User-defined jobs.** Several user-defined jobs (see "Administration Through the K2 Dashboard" on page 60) are installed to extend the capabilities of the K2 Dashboard.

Documentation:

- **K2 documentation**. PDF and HTML versions of all K2 product documentation.
- **Online help.** HTML-based help for K2 Dashboard and Verity Business Console.
- **K2 component application samples**. Java/JSP sample applications constructed from the K2 application component framework. The component samples demonstrate the capabilities of the various K2 Services.

Other Installation Programs

The following components are licensed and installed separately from the K2 installer.

- **Verity Intelligent Classifier**. A tool with a graphical user interface that helps you to construct taxonomies of topics appropriate for an enterprise, and populate them with documents from the enterprise. Described in the *Verity Intelligent Classification Guide.*
- **Verity Locales**. Driver modules that provide language-specific processing and indexing for international documents. Described in the *Verity Locale Configuration Guide.*
- **Verity Developer's Kit**. C APIs, libraries, and tools that support lower-level, non-K2 application development. Described in the *Verity Developer's Kit Programming Reference.*

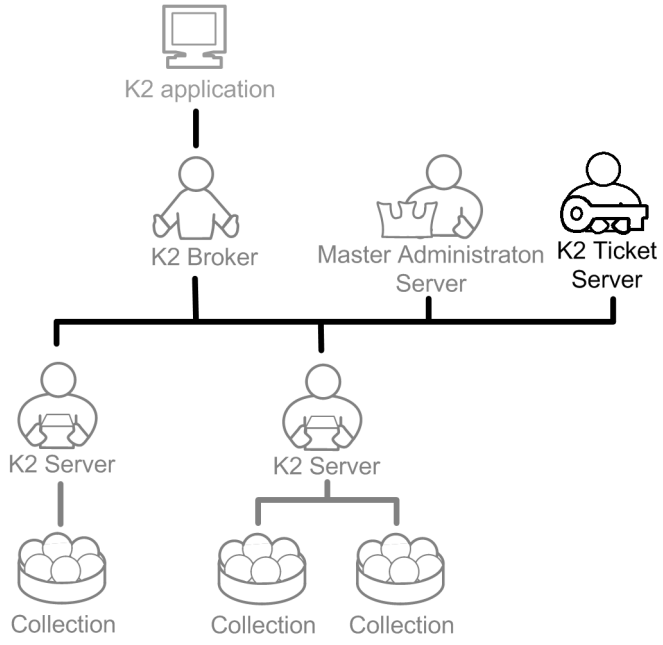
Setting Up Security

Security in K2 is multi-leveled and integrated with native security systems in your enterprise.

K2 Ticket Server

At each K2 installation, there are optionally one or more instances of the K2 Ticket Server (Figure 2-5). The K2 Ticket Server is a component that authenticates users to the K2 System and controls search access to secure indexes. The K2 Ticket Server uses security modules that can communicate to third-party information stores to validate authentication credentials. The authentication stores that are supported include LDAP Servers, Windows NT Domain, and UNIX.

Figure 2-5 K2 Ticket Server



When a user authenticates to a security model by providing valid login information, the user receives a *ticket*, or temporary access pass, from the Verity K2 Ticket Server. The K2 Ticket Server stores information in memory for users who have been authenticated to LDAP, Windows NT domain, or UNIX. Once the user ends the session by logging off, the ticket expires and the user's credentials are deleted from memory. Optionally, the K2 Ticket Server can save credentials to an encrypted store to retain them from session to session.

The K2 Ticket Server monitors search and viewing requests on a TCP/IP port. As each request is made, the K2 Ticket Server gives users access to only those collections for which they have the correct tickets. This integrates your native security model into the K2 system.

The K2 Ticket Server can handle authentications from multiple native security models. For example, if a user provides K2 with credentials to authenticate to the NT Domain security model, the user is able to search documents protected by that domain (as long as the user has permission to read them, based on group membership). However, collections can include information from two or more separate repositories, each with its own native security model. If the collection the user is searching also contains documents from a Microsoft Exchange repository, K2 also prompts the user to authenticate to Microsoft Exchange.

The K2 Ticket Server includes a persistent store, in which it keeps a list of administrative users for the system, plus security information for collections and repositories.

Gateway Security

Gateways provide secure information access to let users retrieve and view documents in their native formats from repositories located anywhere. Each gateway uses the native security model of the repository it represents. To access a repository through a gateway, users must first provide credentials to the native security model—for example, if users want to access information from a Documentum repository, they must provide user ID and password information for Documentum. Only then can they search or retrieve documents from the Documentum repository.

Collection-Level Security

Collection-level security limits which collections a user can search. To search a collection, a user must be a member of a particular user group. Network administrators create these user groups regularly when they set up native LDAP, Windows NT domain, or UNIX

security models. After creating a collection, the K2 administrator simply assigns it to one or more user groups. This assignment allows K2 to determine exactly which users have permission to search the collection.

For example, if your enterprise indexes all your human resources information into a single HR collection, it might be important to restrict collection access to a user group that consists only of senior human resources management. If an outside employee searches for enterprise-wide salary information, K2 excludes the employee from the HR collection because the employee is not part of that group. No salary information appears in the employee's search results.

In addition to providing flexible restrictions to sensitive material, collection-level security accelerates search performance by limiting the number of collections that a query is run against. This can be helpful if your enterprise has indexed its information into a large number of collections.

Obtaining Access to Secure Collections

To obtain access to a secure collection, the user must authenticate to K2. To authenticate, the user provides K2 with the login information of the enterprise's native security model. K2 then uses this information to verify the user's group membership. For example, if your enterprise uses a Windows NT Domain security model, the user provides K2 with a valid NT user name, password and domain name. K2 stores this information, authenticates the user with NT, and obtains the user's NT group information. Only then will K2 grant the user access to the collections for which the user is authorized.

Anonymous Access

Enterprises can allow anonymous access to collections and documents. This access method is useful for companies that expose information to public users. Such publicly available documents are stored in a non-secure collection and are configured in LDAP, NT or UNIX to be anonymously available. If a user signs on to K2 without first logging into the native security model, K2 gives access to these public documents only.

Document-Level Security

K2 uses native authentication and the K2 Ticket Server to achieve collection-level security. However, you may also want to limit certain documents within collections to specific users. In this case, you implement *document-level security* to control whether a document appears in a results list, and whether a user can retrieve it.

Document-level security uses Verity Gateways to determine a user's access rights for individual documents. Each gateway respects and enforces the document repository's existing security model. Since gateways support ODBC-compliant databases, Lotus Notes, Documentum, Microsoft Exchange, Web servers, and file systems, K2 can examine access rights for multiple models and use them to provide document-level security.

No Results Filtering

K2 offers different methods for document-level security. The first is *no results filtering*, in which you configure K2 to display all documents in a results list or category, regardless of user access rights. If a user doesn't have access rights to view a document, he can see its results list information, such as its title and summary, but he cannot retrieve it. This method is useful when you want users to be aware of documents, whether or not they can view the details within them.

Results-List Filtering

The second method of document-level security is called *results-list filtering*, in which K2 checks each document for access rights before it displays the results list to the user. Filtered results lists and categories only show documents that a user can retrieve.

Results list filtering is useful when you do not want particular users to be aware of certain documents within a secure collection. Results-list filtering can be important in some situations, because a query result for some documents might provide as much information as the entire contents of the document itself.

Access-Control Lists

For gateways that support them, K2 uses access-control lists (ACLs) to regulate security at the document level. To enhance performance, an ACL for each document is cached within a collection. When users submit queries, K2 uses the cached information to determine whether the user can access a document, instead of examining the access rights of each document in its remote repository. This approach dramatically increases the speed with which K2 returns results and significantly decreases the load on each repository.

K2 provides the flexibility to use cached ACLs or to check repository access rights when it generates a results list. However, when a user selects a document from a results list for viewing, K2 always checks the repository for access rights. It does not use cached ACLs to determine whether a user can open a document for viewing. Therefore, even if a document's access rights change immediately after a collection is indexed, K2 will apply the most current security measures before it displays the document itself to the user.

Single Sign-On

By integrating with third-party security solutions such as Netegrity SiteMinder, K2 can support *single sign-on*, in which a user logs in just once to access all repositories for searching, viewing, and indexing. Both basic authentication (submission of user ID and password) and HTML forms-based authentication are supported.

In a SiteMinder single sign-on deployment, K2 observes the underlying security privileges defined by SiteMinder to control whether a document appears in a results list, and whether a user can retrieve it. SiteMinder passes authenticated user information to the K2 Ticket Server, which accepts the users and grants tickets without requiring additional authentication.

Integration with collection-level security is not supported.

For gateway access to documents (for viewing or indexing), K2 can be configured for *pre-authentication*, in which the gateway accepts all users as being already authenticated, and thus not requiring separate authentication by the gateway.

Integrating K2 with SiteMinder is a complex process, best implemented with the help of Verity Professional Services. The process is described in the Verity K2 Technical Note *Integrating Verity K2 With Netegrity SiteMinder.*

Internationalizing a K2 Installation

K2 includes built-in language support. Administrators can create indexed collections of documents in any Verity-supported language, and users of a language-aware K2 application can apply language-specific searches against those collections. Furthermore, a single collection can hold indexed, searchable documents in many different languages.

To fully support non-English languages and the multitude of character-encodings used for storing international documents, K2 provides locales, automatic language identification, and automatic character-set detection and conversion.

Locales

Verity *locales* are code modules and data tables that allow documents to be indexed and searched in a language-specific manner. Locales are at the core of Verity's support for internationalization. By installing and configuring locales, an administrator can give a language-aware client application the ability to work in languages other than English.

Multilanguage Locale

Verity's principal locale is the *multilanguage locale*, which provides specific linguistic support for the following languages:

European	Eastern European/ Middle Eastern	Asian
Danish	Arabic	Chinese (simplified)
Dutch	Czech	Chinese (traditional)
English	Greek	Japanese
Finnish	Polish	Korean
French	Russian	
German		
Italian		
Norwegian (Bokmal)		
Portuguese		
Spanish		
Swedish		

The multilanguage locale also provide basic search and viewing support for these languages:

Afrikaans	French	Latvian	SeTswana
Albanian	Frisian	Lithuanian	Sinhala
Arabic	Gaelic	Luxembourgish	SiSwati
Armenian	Galician	Macedonia	Slovak
Basque	Georgian	Malay	Slovenian
Belarusian	German	Malayalam	Somalian
Bengali	Greek	Maltese	Spanish
Breton	Greenlandic	Mongolian	Swahili
Bulgarian	Gujarati	Myanmar	Swedish
Catalan	Panjabi (Gurmukhi)	Nepali	Syriac
Cherokee	Hausa	Norweigan (Bokmal)	Tamil
Chinese (simplified)	Hebrew	Norweigan (Nynorsk)	Telugu

Chinese (traditional)	Hindi	Lao	Thai
Croatian	Hungarian	Oriya	Tibetan
Czech	Icelandic	Pashto	TsiVenda
Danish	Indonesian	Philippine**	Turkish
Dhivehi (Thanna)	IsiNdebele	Polish	Ukranian
Dutch	IsiXhosa	Portuguese	Urdu
English	IsiZulu	Rhaeto-Romance	Vietnamese
Estonian	Italian	Romanian	Welsh
Esperanto	Japanese	Russian	XiTsonga
Ethiopic	Kannada	Sami	Yiddish
Faroese	Khmer	Serbian	
Farsi	Korean	Sesotho	
Finnish	Kurdish	SesothoSaLeboa	

**Includes Tagalog, Hanunoo, Buhid, Tagbanwa

The multilanguage locale uses the UTF-8 character set, which is based on Unicode 3.2. Use of UTF-8 allows individual collections built on the multilanguage locale to contain documents in more than one language.

Single-Language Locales

Verity also provides locales that each support a single language only. The locales are available in two sets: Asian locales and European locales. Single-language locales are available for the following languages:

Single-language Asian locales	Single-language European locales	
Japanese	Norwegian (Bokmal)	German
Korean	Danish	Italian
Chinese (simplified)	Dutch	Norwegian
Chinese (traditional)	English	Portuguese
	Finnish	Spanish
	French	Swedish

As delivered, K2 includes full support for English plus limited support for other languages (through the multilanguage locale). To obtain additional language capability, you can license and install additional locales and languages.

Language Identification

Because the multilanguage locale supports many languages, it generally must be able to identify the language of each document to be indexed, so that the proper language-specific processing is applied.

The Verity *language-identification filter* is used to detect the language of incoming documents before they are indexed. The filter makes use of the document's encoding and language features to make the identification.

Character-Set Detection and Conversion

When reading in a document for indexing, the Verity engine commonly has no prior knowledge of what character encoding the document uses. In those cases, the Verity auto-detection process analyzes the incoming document data to determine its character set.

After character-set detection, the engine converts the document's text, if necessary, to the character set required by the current locale. K2 then automatically converts between the character set of that indexed information and the character set used by a Verity application accessing it.

Verity's character-set conversion capability can convert between all common character sets in dozens of languages.

For more information on internationalization and localization of Verity applications, see the *Verity Locale Configuration Guide*.

Administering a K2 Installation

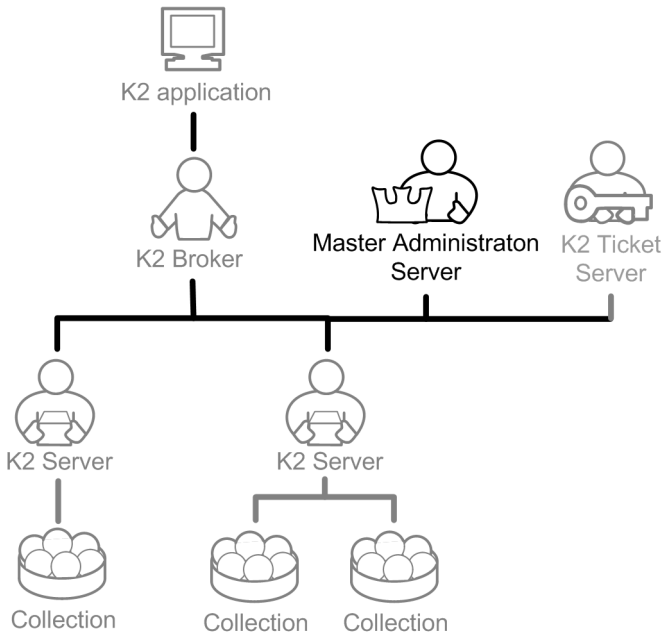
For most administrative purposes, you use the browser-based K2 Dashboard to manage K2 services and functionality at a K2 installation. The command-line tool `rcadmin` is also available for performing many administrative tasks.

Administration-Server Architecture

On each host on which it installs a component of K2, the K2 installer also installs a K2 Administration Server. The Administration Server takes care of interactions between the components of K2 on the local host, and between the local host and other machines.

Furthermore, for each installation as a whole there is one instance, on one host only, of the K2 Master Administration Server (see Figure 2-6). The Master Administration Server communicates with all the other Administration Servers. It is through the Master Administration Server that you administer your K2 installation.

Figure 2-6 Master Administration Server



At installation time, the K2 Ticket Server and the Administration Servers on each host machine are configured to connect to the Master Administration Server.

Administration Through the K2 Dashboard

Note This discussion is a brief summary only; for complete information on using the K2 Dashboard, see the *Verity K2 Dashboard Administrator Guide*.

The K2 Dashboard is an application with a browser-based user interface enabling you to control configuration settings for the components in your K2 system. If your K2 system consists of many K2 Servers, K2 Brokers, and indexes in different physical locations, you can use the K2 Dashboard to configure settings for the entire system from one computer.

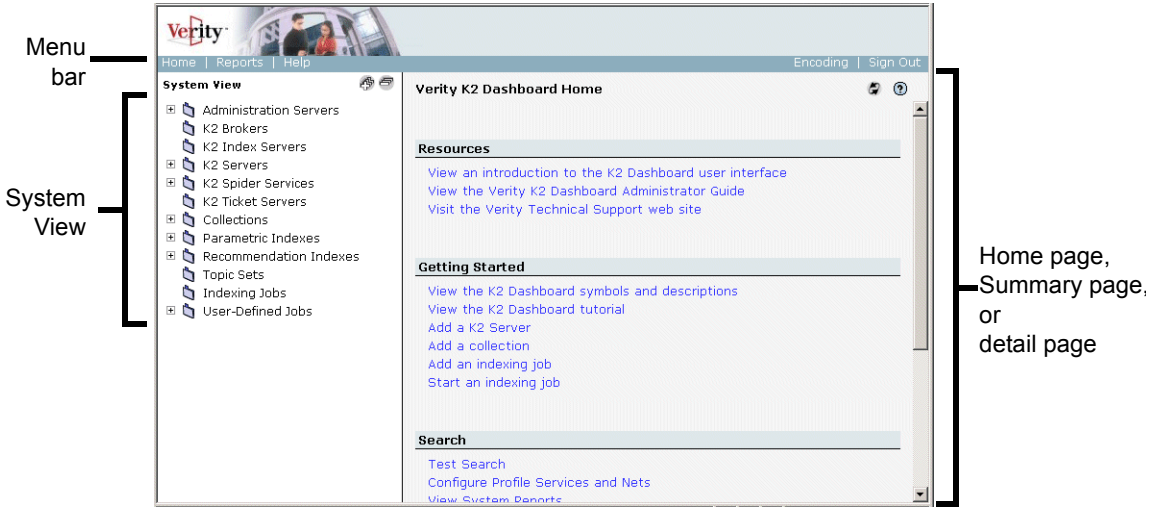
These are some of the K2 Dashboard configuration tasks you can apply to a distributed environment:

- Add a new service (K2 Broker, K2 Server, K2 Ticket Server, or other service).
- Manage and monitor the status of a K2 service.
- Create and manage collections and their style sets.
- Attach a collection or other index to a K2 Server.
- Assign security to an index.
- Generate user activity reports.
- Set logging options and view log files.
- Create a *user defined job*, a scripted command to run any command-line tool directly from the K2 Dashboard.
- Use user-defined jobs to optimize collections or stage indexes.
- Add and remove administrative users.
- Manage license keys.

Using the K2 Dashboard

To use the K2 Dashboard, you launch it and log in. After login, the K2 Dashboard home page appears (Figure 2-7).

Figure 2-7 K2 Dashboard home page



Note these areas of the K2 Dashboard screen:

- **Menu bar.** Across the top area of the window. Includes these links:
  - **Home.** Displays the K2 Dashboard home page (the page shown in Figure 2-7).
  - **Preferences.**
  - **Help.** Displays the *Verity K2 Dashboard Administrator Guide*. Context-sensitive Help is available from other Help buttons.
  - **Logout.** Ends the current K2 Dashboard session.
- **System View.** Displays all K2 system components, grouped by type. You select a K2 component in the **System View** to view or change its settings through the related summary or detail page (or action page, accessed through the summary or detail page).
- **Summary pages.** Each displays a summary of the components represented by the currently selected folder in the **System View**. For example, if you click the **K2 Server**

folder, the summary page lists all K2 Servers in the system and gives various statistics and configuration actions for each one.

Summary pages offer useful overviews that allow you to monitor the performance of system services and indexes.

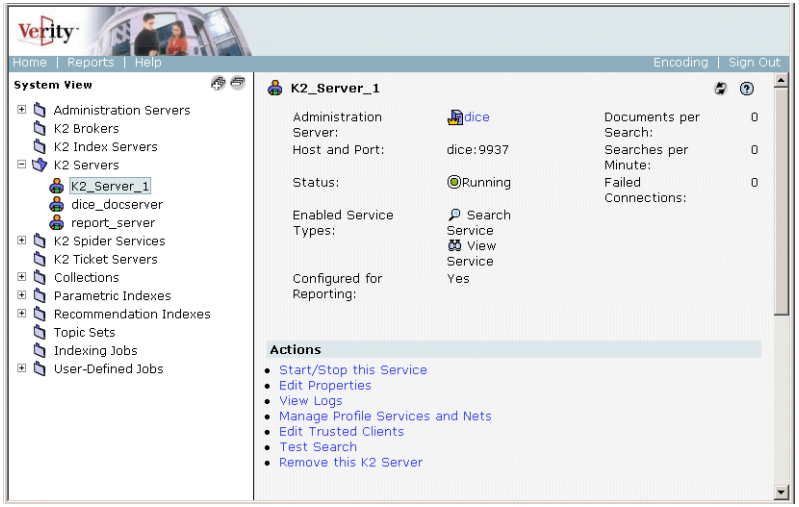
- **Detail pages.** Each displays information about an individual component that has been selected either from a summary page or from within a folder in the System View. For example, if you open the **K2 Server** folder in the **System View** and then click the icon **K2_Server_1** beneath it, the detail page for **K2_Server_1** (Figure 2-8) appears.

  A detail page shows an individual component's relationship to other services and indexes in the K2 domain. It also provides access to all configuration actions available for that component.

- **Actions.** You access most K2 Dashboard functions by selecting *actions*. For example, separate actions allow you to start a service, create an index, or run a collection indexing job.

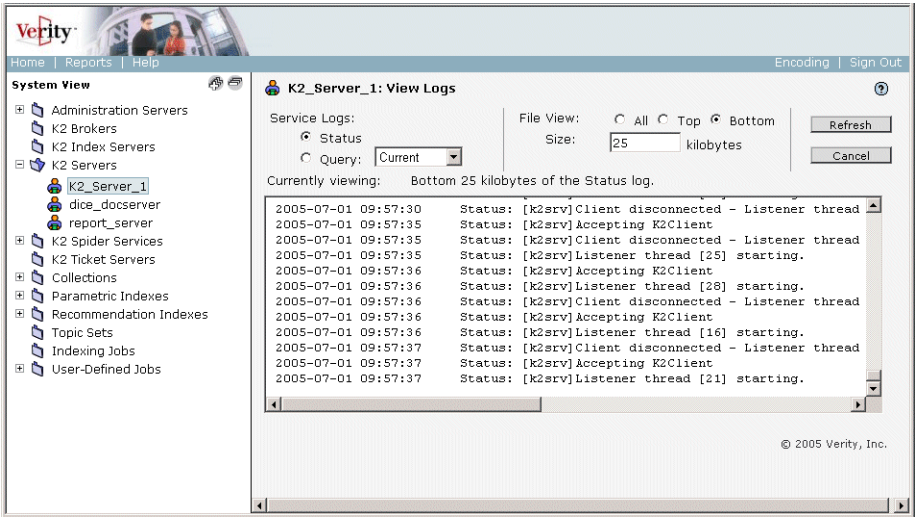
  Action links appropriate to each component are displayed in the upper area of the component's detail page (see Figure 2-8), and as links associated with the component's related services and indexes. Action links are also available on the K2 Dashboard home page, as shown in Figure 2-7 on page 61.

Figure 2-8 Action links on the K2_Server_1 detail page



When you click an action link, the K2 Dashboard displays information related to that action. For example, to view the service logs for **K2_Server_1** in Figure 2-8, click the **View Logs** action link. The K2_Server_1 detail page is replaced by the View Logs page.

Figure 2-9 View Logs page



You can then perform additional actions using the controls on this page.

- **Notifications**. K2 Dashboard notifications are displayed on the home page and on summary, detail, and action pages. They show system messages and prompt you to perform required actions, such as restarting a K2 Server.

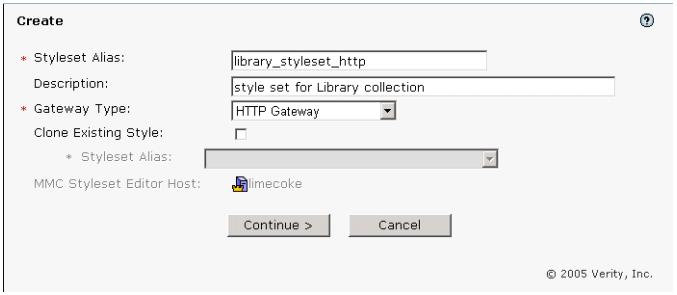
Using the StyleSet Editor

A *style set* is the complete set of *style files*, the Verity configuration files that control many aspects of indexing behavior and index structure. Before you create a collection, you need to select or create the style set that it will use.

Style files generally have the scope of an individual collection. When creating a collection, you typically create (copy and modify) a set of style files customized for that collection.

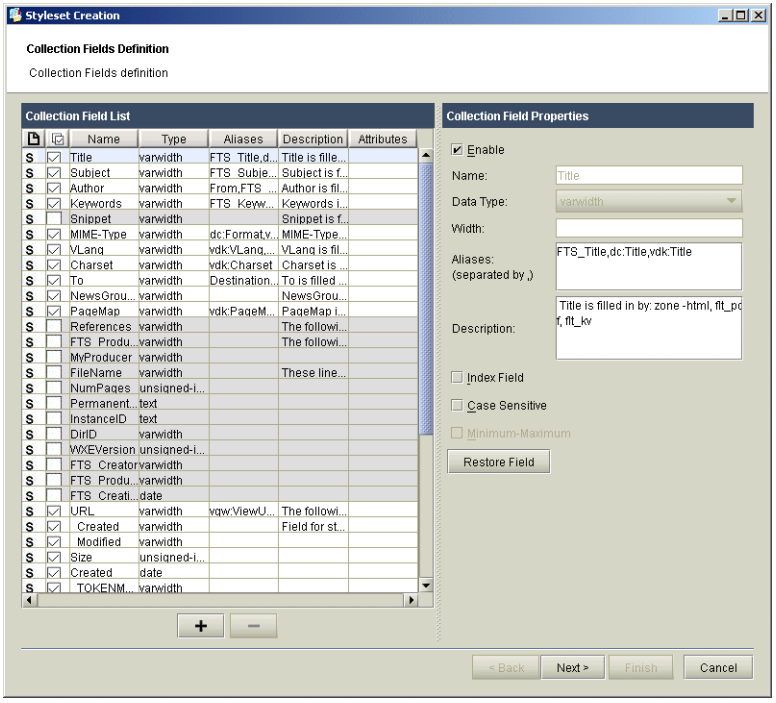
Verity K2 includes the StyleSet Editor application, which offers a graphical approach to creating and modifying style files. You can launch the StyleSet Editor from the K2 Dashboard. Figure 2-10 shows the Create Styleset page.

Figure 2-10 Create Styleset page (in K2 Dashboard)



Clicking **Continue** brings up the StyleSet Editor's Collection Fields Definition page (Figure 2-11).

Figure 2-11 Collection Fields Definition page (in StyleSet Editor)



The StyleSet Editor's user interface is gateway-specific; the fields, tabs, and navigation features that you see depend on which gateway your style set will be used with.

Once you create a style set through the StyleSet Editor, it is automatically registered and stored with the Master Administration Server, and then it is available to assign to a collection. For more information, see the chapter on identifying style sets in the *Verity K2 Dashboard Administrator Guide*.

Administration With Other Tools

The Verity command-line tool `rcadmin` and a variety of command-line indexing tools and configuration files provide a non-graphical alternative to administering a K2 installation.

Using rcadmin

The rcadmin command-line tool enables you to view and update configuration settings for the components (K2 Servers, K2 Brokers, collections, and other indexes) in your K2 system. It also provides a scriptable, command-line alternative to the K2 Dashboard. In some cases, you can use rcadmin to configure settings that are not available through the K2 Dashboard.

You can create scripts that contain rcadmin commands to automate many administrative tasks. For example, you can create a script to start or stop a server, add a new collection, or take an index offline.

For more information on rcadmin, see the *Verity K2 rcadmin Guide*.

Using Indexing Tools

Verity provides a large number of command-line tools for use by administrators. You use many of these tools to create the indexes and structures described in "Indexing and Classifying Information" on page 71. These are some of the tools:

- `mkvdk` creates a collection.
- `vspider` creates a collection by indexing documents in a given directory structure.
- `k2spider` works with K2 Server to create a collection by indexing documents in a distributed environment.
- `mkpi` creates a parametric index.
- `mktopics` creates a topic set.
- `mkprf` creates a profile net.
- `browse`, `didump`, `rcvdk`, `rck2`, `testqp` allow you to inspect collections and other indexes.

Manually Editing Style Files

One of the administrative tasks involved with creating collections and other indexes is the customization of style files. The K2 Dashboard provides access to the StyleSet editor, a tool for graphically editing style files. (See "Using the StyleSet Editor" on page 63.)

Style files are text files with names of the form `style.`*xxx*, where *xxx* is a three-letter extension related to the file's function. For example, the style file that controls the universal document filter (see "Verity Document Filters" on page 40) is named `style.uni`. You can edit certain style files manually, using a text editor. Style files are documented in detail in the *Verity Collection Reference*.

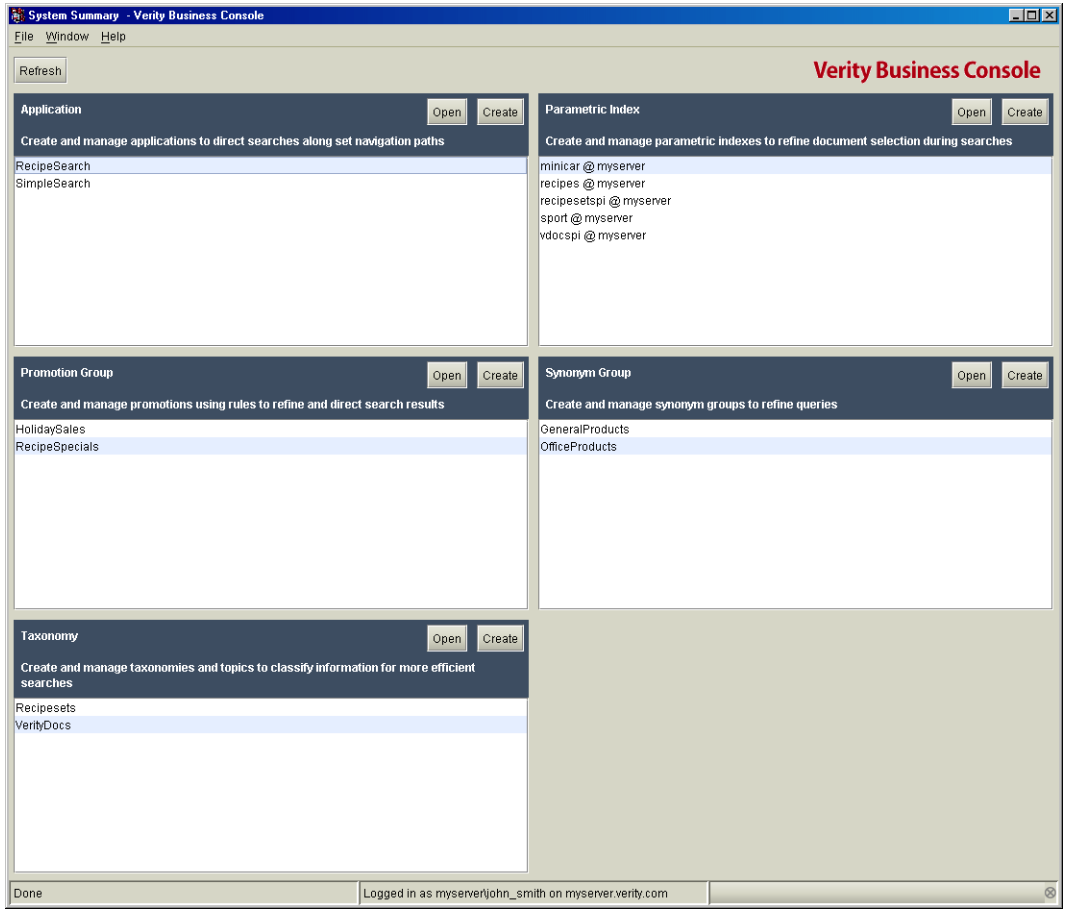
Managing Information With the Business Console

The Verity Business Console is a server-based Java application with a WebStart Java client that allows knowledge workers and other business users to manipulate information-classification structures and indexes. It also facilitates fast development of search and classification applications constructed with the Verity Component Framework.

Modules

The Business Console interface consists of several separate modules with different purposes. You can access the modules through individual panes in the Business Console Summary window, as shown in Figure 2-12.

Figure 2-12 Business Console Summary window



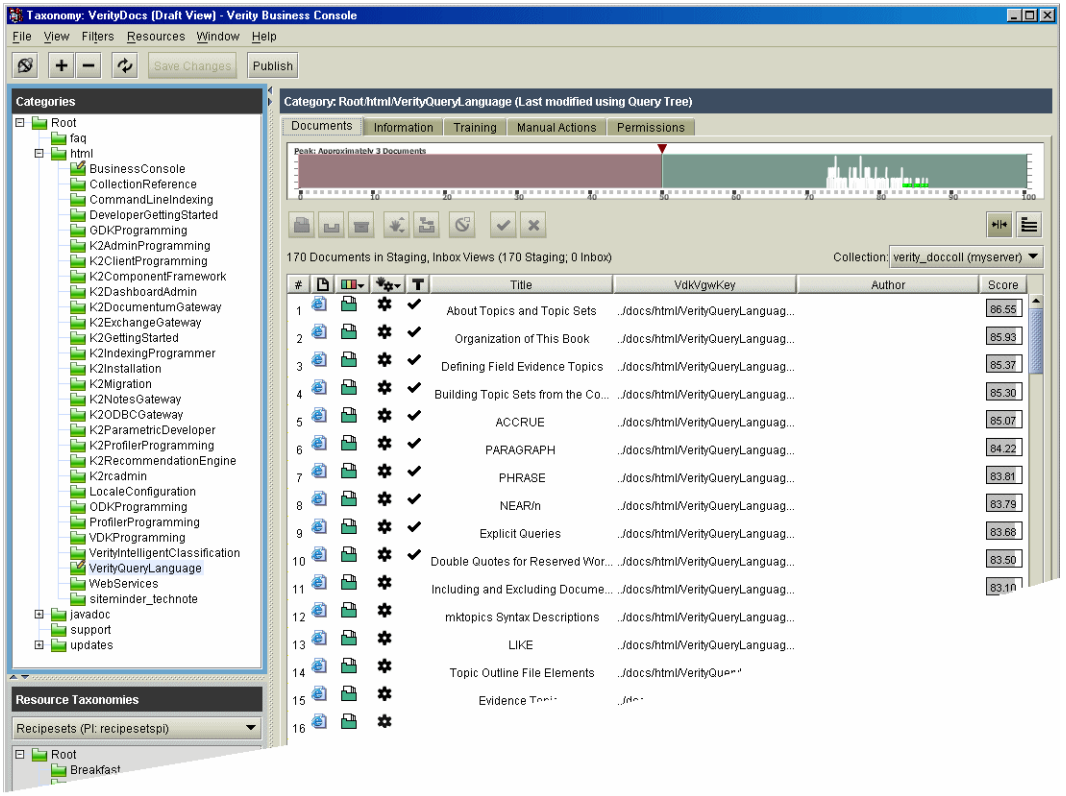
These are the available modules:

- **Taxonomy**. Use this module to manipulate taxonomies (see "About Taxonomies" on page 84), topics, topic sets, and parametric indexes. You can use the taxonomy module to classify information into categories for more efficient searching
- **Parametric index**. Use this module to create and maintain parametric indexes (see "Parametric Indexes" on page 78). Parametric indexes can refine the selection of documents in a search.
- **Applications**. Use this module to create application components with the Verity Component Framework and configure them to work together as a search or classification application. See "Using the Component Framework" on page 116.

- **Promotion group**. Use this module to define sets of links that should appear in your application's search results whenever the user enters specific queries. You can create a promotion, such as an advertisement or datasheet, configure the promotion in the Component Framework, and then display the promotions on the results page based on the rules you have set.
- **Synonym group**. Use this module to create and maintain keyword files that your application can use to refine queries by expanding the search to include also synonyms of the user's search term.

Once you have selected a module, the Business Console interface for that module appears (Figure 2-13). Each module's interface includes multiple panes for creating and manipulating the module's data.

Figure 2-13 Taxonomy module interface



User Types and user Roles

Several types of people are Business Console users:

- **Administrators** might use Business Console to create parametric indexes. They typically are IT specialists.
- **Business Analysts** might use Business Console to create promotions and synonym groups. They typically are domain experts in terminology and business issues.
- **Developers** use Business Console to create applications that make use of JSP and XML technologies.
- **Librarians** (knowledge workers) use Business Console to create and manage taxonomies.

Users in individual Business Console modules can occupy one or more roles. For example, *editors* in the taxonomy, promotion group, and synonym group modules can modify structures and can edit information for nodes. *Publishers* in the same modules can also publish nodes.

Using the Business Console

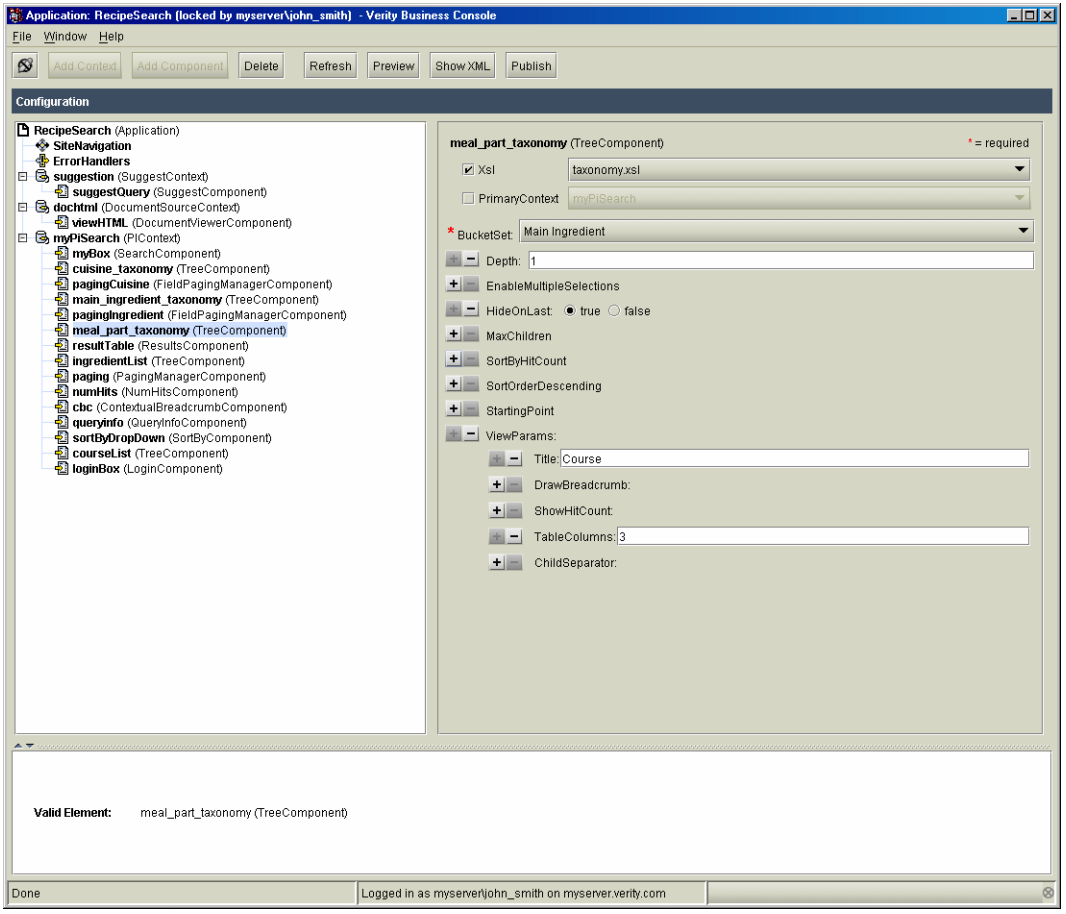
To use the Business Console, you open a browser and navigate to the Business Console URL. Once there, you click **Launch Business Console Client** and the Summary window (see Figure 2-12 on page 67) appears.

Note Some installations are not licensed for the taxonomy module, and some users may not have permission to access all licensed modules; therefore, your Summary window might not look exactly like Figure 2-12.

The user interface that the Business Console presents to you, and the tasks that you perform, are different for each module.

- In the taxonomy module, for example (see Figure 2-13 on page 68), the window is split into three main areas—one to display the categories of the taxonomy being worked on, one to display source information you can add to the current taxonomy, and one for manipulating information in the taxonomy itself.
- As a different example, when using the application module to configure an XML component from the Verity Component Framework (Figure 2-14), the window has two panes—one for displaying the component hierarchy and one for setting component attributes.

Figure 2-14 Configuring a component with the application module



3

Indexing and Classifying Information

At most Verity K2 installations, administrators or knowledge workers take responsibility for generating the data tables and search-support structures that K2 applications use to deliver information to users. Decisions about the business and access rules underpinning the tables and structures are made by knowledge workers (domain experts and librarians).

This chapter, therefore, contains information primarily of interest to administrators and knowledge workers. It contains the following sections:

Building Collections

Before users can search or classify enterprise information, it generally must be indexed. The K2 system uses various gateways to access and gather information into a universal index, called a collection. The K2 Spider or the Index Server can manage the indexing process that builds your collections. You control the indexing tool through the K2 Dashboard or through command-line tools such as `rcadmin`.

A collection stores the locations of all indexed documents and a list of essentially all words contained within the text of the documents. Collections are in general much smaller than the total size of the documents they represent. When a user chooses to view a document returned as a search result, K2 uses a gateway to retrieve the document from its repository.

A single K2 system can have multiple collections. For example, a business could establish a collection containing all human resource documents, such as resumes and personnel records. The same business could set up another collection that contains all customer service information, or a third that contains departmental budget forecasts and reports. It can then configure K2 to allow some users to search all of these collections, and to restrict other users from a particular collection. For example, only those employees in the human resources department who need access to personnel records would be able to run searches against the human resources collection.

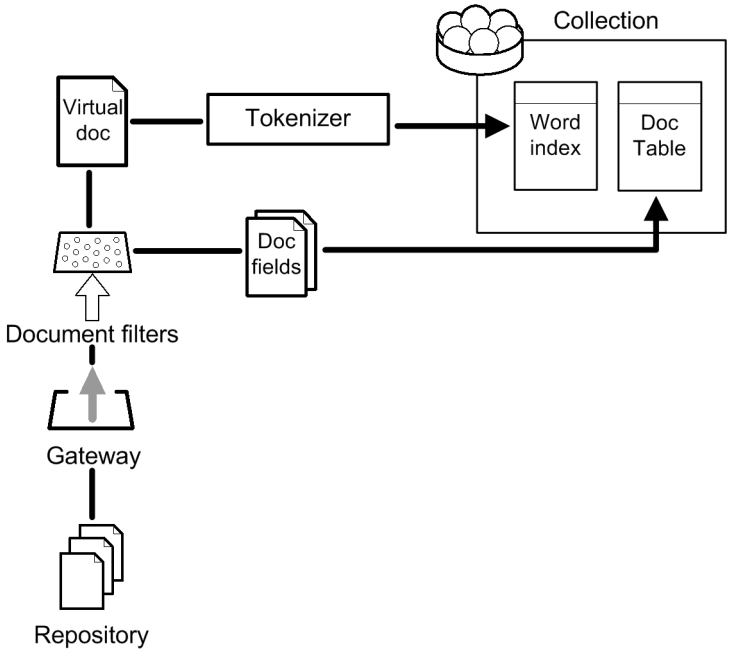
The Indexing Process

Figure 3-1 summarizes the process of indexing a set of documents in a repository.

1. One by one, repository documents are opened (by K2 Spider by or some other indexing tool).
2. Each opened document's format—such as Microsoft Word, Lotus 1-2-3, or HTML—is automatically identified.
3. K2 invokes the correct *document filter* for the format and extracts a *word index* from the document's content.
4. K2 extracts document *metadata* (fields or attributes) for indexing into the collection's *document table*. Several predefined metadata fields are supported, including author, date and title. Custom fields are also supported. You define custom fields with a name and data type, such as text, date, or numbers.

   By extracting field values through gateways, K2 preserves the structure of structured and semi-structured documents so that users can target desired content more precisely.

Figure 3-1 Indexing documents into a collection



The collection's document table also includes an identifying key (such as physical file system address or URL) for each document in the collection.

A collection can optionally store data for document zones and fields. A *zone* is a named region of text in a document, such as the area in the `<h2>` or `<body>` tags in an HTML file. A collection that includes zones allows users to search these regions of a document for specific content.

For more information about building collections, see the *Verity Command-Line Indexing Reference*, the *Verity Collection Reference*., and the *Verity K2 Dashboard Administrator Guide*.

Note Verity also provides API s for indexing collections from within an application. For more information, see the discussion of the Collection Indexing API in "Client APIs" on page 117.

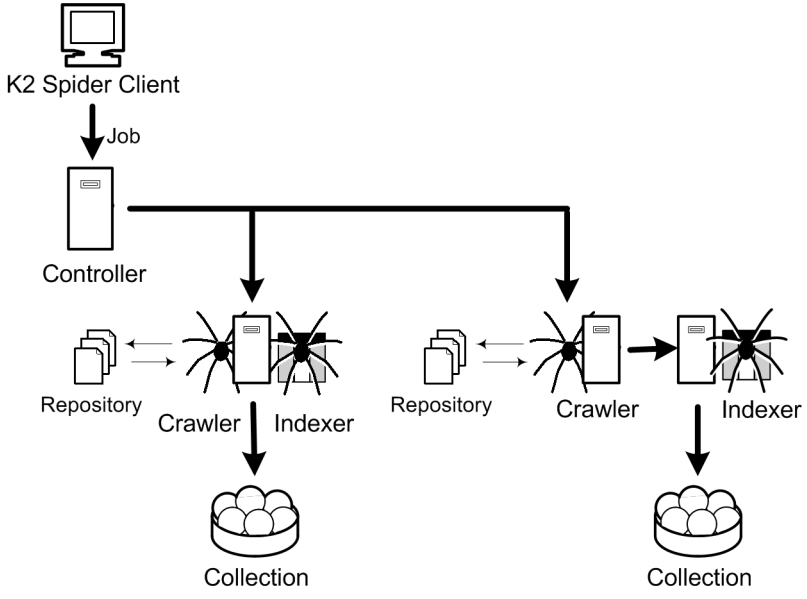
K2 Spider

The Verity K2 Spider manages the indexing process that builds your collections. To build a collection, K2 Spider uses gateways to gain access to data and documents contained in repositories.

For example, if an enterprise wants its knowledge workers to be able to search Documentum files, the Documentum Gateway obtains access for K2 Spider, and then extracts properties, fields, data, and security schemas from the Documentum repository. K2 Spider then communicates with the gateway and gathers these attributes into a collection.

When users query for a Documentum file, K2 searches against the collection and displays a list of results. If a user selects a document for viewing, K2 uses the Documentum Gateway to access the selected file in the repository and then display it.

Figure 3-2 Using K2 Spider for indexing



Distributed Indexing

K2 Spider distributes processes to *controllers*, *crawlers*, and *indexers*. Documents can be indexed in real time as they are created or modified, without disrupting current indexing jobs. K2 crawlers "walk" through information sources as indexes are built. The K2 Spider controller manages the crawlers, indexers, jobs, and workflow. As shown in Figure 3-2 on page 74, the controller, crawlers, and indexers can be on separate machines.

A Spider *job* is the actual process of building the collection. Using optional features, the K2 Spider can also crawl secure Web servers that use SSL certificates and Web servers with cookies.

K2 Spider's distributed architecture provides scalability and fault tolerance. For example, you can configure K2 Spider to index one repository but create two separate, identical collections on different servers. If one server slows down, K2 Spider continues to update the collection on the other server, ensuring up-to-date query results.

Similarly, if K2 Spider detects a corrupted or unreadable document, it does not halt the indexing operation. Rather, it logs the corrupted document and continues to index the remaining documents, guaranteeing that they become part of the collection. You can then use the log information to locate and repair the corrupted documents in the repository.

The K2 administrator controls K2 Spider's indexing through the K2 Dashboard or through the command-line tools `k2spider_srv` and `k2_spider_cli`.

Continuous Indexing

K2 Spider continuously monitors the state of your collections. Its crawlers are persistent and regularly crawl documents based on the indexing frequency you set. They can also be configured to index documents as soon as they are created or modified, and can be integrated with content management system workflow processes. Collections can be updated continuously and automatically, even while searches are in progress. You do not have to repeatedly schedule which documents to index.

Customizing K2 Spider

You can configure K2 Spider to suit your specific requirements through its published APIs. K2 Spider includes C and Java APIs that give you flexibility and control in building indexed collections. The APIs enable you to build your own administrative tool or user interface to control different aspects of the indexing process.

K2 Spider also provides *style files* (configuration files) for setting up indexing preferences. Style files are used to control different aspects of a collection, such as the summary data displayed in query results or the fields available for searching. For both optimal indexing performance and ease of administration, style files can be created and edited through the

StyleSet Editor, a Windows-based interface. The SSM lets you visually explore repositories, define fields to extract for indexing and map repository fields to collection fields. For more information on style files, see "Manually Editing Style Files" on page 66.

Direct Indexing

Verity K2 also includes a programming interface that applications can use to index and update collections without going through K2 Spider. The Collection Indexing Java API uses the K2 Index server to allow your application to submit document content directly for indexing into a collection.

The Collection Indexing API is especially useful in situations such as these:

- No gateway exists for the document's repository.
- You want to implement custom crawling or event triggering not handled by the repository's gateway.
- You want to augment a collection with metadata during or after indexing.
- The content to be indexed is on a remote system from the collection and file sharing is not available.

Note The collection indexing API is not typically used when K2 Spider operations are adequate; using K2 Spider does not require programming.

When using the Collection Indexing API, your application can submit individual documents, or it can submit a list of documents in a special format, called a *bulk insert file* (BIF).

Creating Topic Sets

In K2, a *topic* is a named expression in the Verity Query Language (VQL), designed to locate documents related to a given concept or subject area. For example, a topic named **DaimlerChrysler Cars** might consist of a query expression that searches for that term itself, plus any corporate division names (like **Mercedes Benz**), plus the names of any of its automobile lines (like **Chrysler**), plus the names of any of its individual car models (like **PT Cruiser**). For this simple example, the body of the topic might contain VQL elements like these:

```
DaimlerChrysler Cars <OR> Mercedes Benz <OR> ... <OR> Chrysler
<OR> PT Cruiser <OR> Caravan <OR> ...<OR> Jeep <OR> <WORD<CASE>
Wrangler <OR>...
```

A more specific topic, such as **hybrid cars**, might consist of a query that searches for **Honda Insight** and **Toyota Prius** and possibly technical or legislative terms in documents relating to low-emission vehicles.

Topics can be created manually by domain experts or knowledge workers who understand how to express a concept in terms of search-query strings. They can also be created automatically, using machine-learning tools. An individual topic can be a simple, short expression, or it can be a long and complex one, involving many terms and sophisticated boolean and non-boolean search operators.

Topics can be combined and compiled into groups called topic sets. A topic set can have a flat structure or it can be hierarchical.

Administrators can create topic sets using either Intelligent Classifier or the `mktopics` command-line tool. K2 applications can use topic sets in several ways:

- A topic set can be attached to an application to integrate it into the application's search capabilities.
- A topic set can be indexed into a collection to provide extra-fast searches over the collection for the topic set's terms.
- A topic set can also be used as the set of business rules for populating the taxonomy of a knowledge tree; see "Creating Category Definitions" on page 86.
- Topics are also used as the basis for profile nets; see "Creating Profile Nets" on page 89.

Note Topic sets are commonly used as sets of query expressions only; the topic names are used as search terms but do not necessarily appear in a K2 application's user interface. If you want to implement a topic-set structure that users can directly browse, you can create it in a parametric index, as described in "Relational Taxonomies" on page 108.

For more information on topics and topic sets, see the *Verity Collaborative Classifier Guide* and the *Verity Query Language and Topic Guide*.

Setting Up Parametric Search

Most enterprise information is semi-structured. For instance, a text document (unstructured) commonly includes associated metadata (structured) such as author, content source, date of creation, size, format, and language. Textual product pages in an online catalog commonly include extensive metadata relating to product features.

Parametric search is a Verity search capability in which users can locate information by simultaneously selecting values in structured metadata and searching through unstructured text.

In a typical setup, each document used for searching includes both unstructured data and structured attributes. For example, in the case of documents that describe cars, attributes might include **Color**, **Price**, **Make**, **Model**, **Mileage**, **Location**, and **Year**. Attributes can have numeric, date, or string values. The free-text portion of a search queries the unstructured data, while the parametric-selection portion queries the structured data through its attributes.

For example a parametric-search query for a car might be described like this:

Find **Red** cars **less than $15,000** with "**air conditioning**"

in which **Red** is one of the available values for the **Color** parameter, **less than $15,000** selects a range from the **Price** parameter, and **air conditioning** is a phrase to be searched for in the unstructured text.

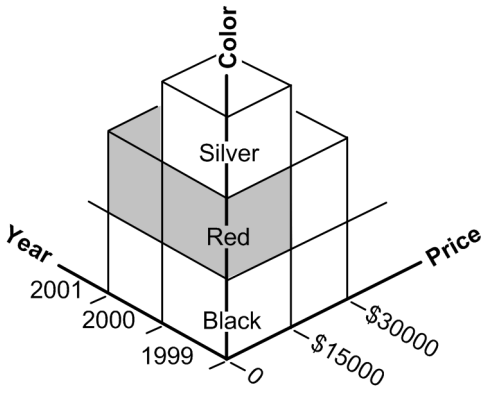
In addition to the ability to combine text queries and parametric values, parametric search can rank results, by either text-query scores or parameter values. The application can then sort the results for the user and allow the user to further refine or broaden the search.

Parametric Indexes

Parametric indexes are the structures that underlie parametric selection. You can think of a parametric index as an extension to a collection's word index; the Verity engine uses it to identify documents matching the requested parameters.

Conceptually, you can view a parametric index as an n-dimensional "parametric cube," a matrix in which each dimension represents a parameter. Figure 3-3 shows a portion of a three-dimensional version, in which the parameters are color, model year, and price.

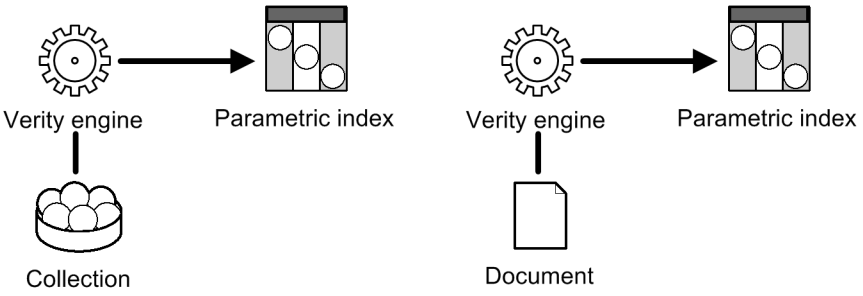
Figure 3-3 A "parametric cube"



Each individual value or range of values for a parameter is called a *bucket*, and each parameter as a whole (each dimension of the cube) makes up a *bucket set*. Each bucket holds references to the documents that have that value for that parameter. For example, in the portion of the cube shown in Figure 3-3, the **Color** bucket set contains three buckets: **Black**, **Red** and **Silver**. The **Red** bucket identifies all documents in the collection that relate to red cars.

In this example, a parametric selection for all red cars costing less than $15,000 and made in either 1999 or 2000 (the shaded area in Figure 3-3) returns only those documents whose field values satisfy all three criteria.

Administrators or knowledge workers can build parametric indexes on top of existing Verity collections or directly from specifically formatted XML documents.



To create a parametric index, the administrator first creates an XML-based *outline file* that specifies the collection or XML fields from which to create the parameters. The administrator can then use the `mkpi` command-line tool to create the parametric index itself. Alternatively, a knowledge worker can use the graphical interface of the Verity Collaborative Classifier to set up the outline file and create the index.

For more information on configuring and using parametric selection, see the *Verity K2 Dashboard Administrator Guide*, the *Verity Collaborative Classifier Guide*, or the *Verity K2 Parametric Developer Guide*.

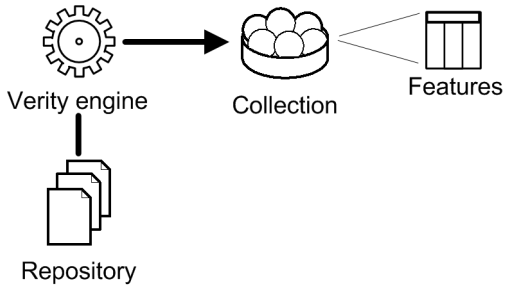
Note parametric indexes can also support user browsing and selection from taxonomies (hierarchical classifications of information). See "About Taxonomies" on page 84 and "About Relational Taxonomies" on page 87.

Extracting Document Features

Document *clustering* is a technique for analyzing a set of documents to create groups (*clusters*) of them that address the same subjects. Document *summarization* is a technique for presenting a list of keywords or a short passage that summarizes the content of a document.

Clustering and content summarization rely on the Verity *feature extraction* technology, which operates during collection indexing. Feature extraction automatically discovers the subjects addressed in a document by performing vector analysis on nouns and noun phrases. Each document's *feature vector* is then stored in the collection for use in clustering and summarization.

Clustering and summarization are collection-level features. Therefore, the administrator must enable them at indexing time, by specifying the generation of feature vectors during collection indexing.



For more information on clustering and summarization, see "Clustering Results" on page 103 and"Returning Document Summaries" on page 104. For more information on feature vectors, see the *Verity Developer's Kit Programming Reference* and the *Verity Collection Reference*.

Extracting Entities

K2 developers can create applications that use a product called Verity Extractor, which is an engine that applications can use to extract *entities*—words or blocks of text that have specific meaning (for example, names, telephone numbers, URLs, addresses, product IDs)—from a document or set of documents.

With Verity Extractor your application can

- Identify and extract elements from document content, based on predefined grammars and rules.
- Use extracted elements to support analytical applications or metadata-enriched search and browse applications.
- Validate documents based on a defined patterns or rules.

The Verity Extractor product includes these components:

- **Core engine.** Supports the extraction of predefined entities using prebuilt resource files (dictionaries and grammars).
  - **Dictionaries**. A dictionary is an XML file that provides a vocabulary for a simple entity, such as a city or country name. A dictionary has a list of headwords; each headword has a set of associated words called synonyms. Verity Extractor uses the dictionary to scan a document and extract the defined entities that match the headwords or synonyms.
  - **Grammars**. A grammar is an XML file that provides rules for complex entities such as URLs or postal addresses. Rules are written in regular-expression format, can be recursive, and can refer to other grammars and dictionaries. Verity Extractor uses the grammar to scan a document and extract entities that match the rules.
- **C API.** Allows you to write a C application that performs entity extractions.
- **Java API.** Allows you to write a Java application that performs entity extractions.
- **Command-line tool (mkve).** Allows you to perform entity extractions from the command line and generate output in multiple formats.

- **Entity extraction filter (flt_ve).** A document filter that extracts entities during the K2/VDK indexing process, allowing entities to be extracted and immediately stored in Verity collection fields.
- **Common entity resource files.** A basic set of pre-packaged, U.S.-specific dictionary and grammar files for common entities such as person, place, organization, address, phone number, email address, date and time.

Figure 3-4 shows the Verity Extractor being used to extract entities into a Verity collection during the indexing process.

Figure 3-4 Verity Extractor (shaded) used during indexing

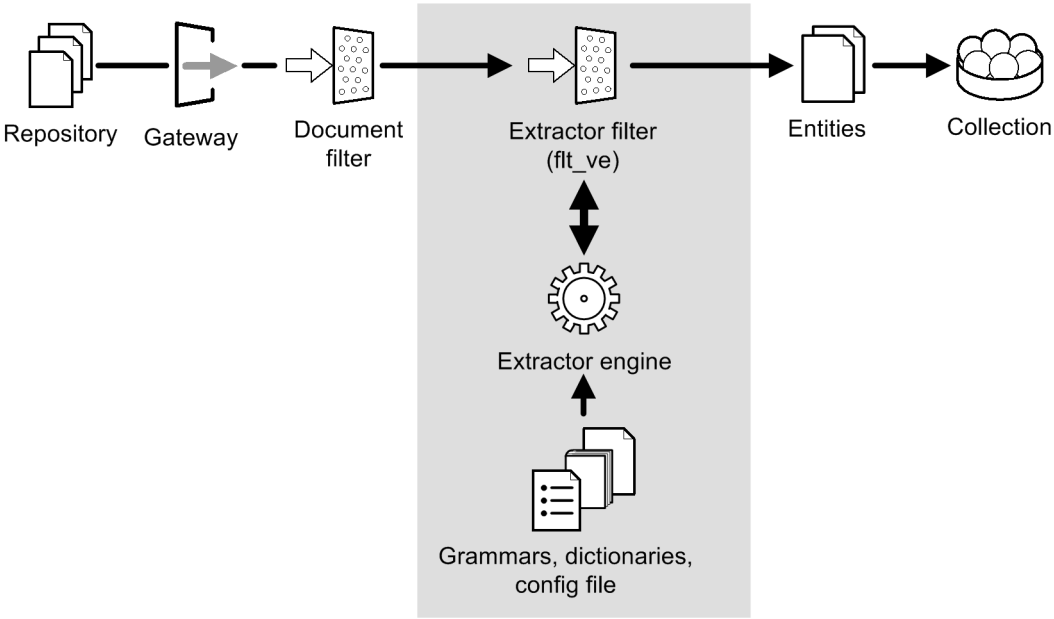


Table 3-1 lists examples of some of the entities that can be extracted with the common entity resource files.

Table 3-1 Some common entities

Entity	Description/Components	Example(s)
Commercial organization	Business name	Verity, Inc.
	Business type	Cardiff Software
	Business designator	
Person(s)	Title	President George W. Bush
	Given name	Harold Potter
	Family name	Mr. and Mrs. John Doe
	Suffix	Mary and John Doe
Place(s)	City	Tokyo
	State	Sunnyvale, California
	Country	United Kingdom
	Region	Northern and Eastern Europe
Postal address		894 Ross Drive, Sunnyvale
		PO Box 9090, Beverley Hills, 90210
Internet address		http://www.verity.com
Money		$200.00
		$50 million
Date	Any time designated by a particular day, month, or year	Monday
		last August
		1/1/2004
		300 B.C.
		July 4th
		Dec. 2000

With Verity Extractor, you can also create your own custom dictionaries and grammars, to refine its capabilities or extend them beyond the common entities.

For complete information about entity extraction and all components of Verity Extractor, see the *Verity Extractor Programming Guide*.

Classifying Information

Besides allowing you to create indexed collections that support different kinds of search, and besides providing tools for extracting various kinds of entities from unstructured data, K2 also helps you to create classification structures that aid users to more readily access the specific information they seek.

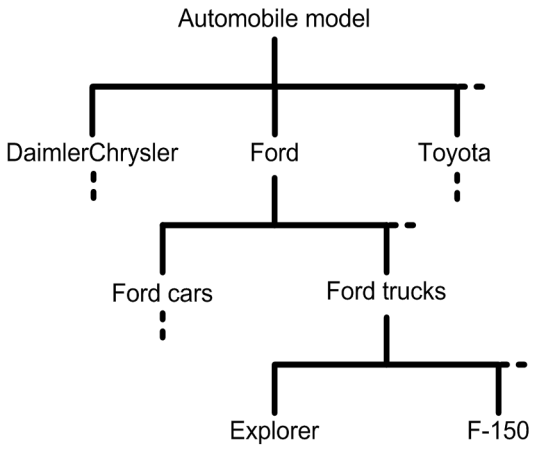
Classifying information into categories and subcategories allows your users to locate individual files by drilling down through the concepts represented by each category until they find relevant documents. It can also help to profile or classify documents for routing purposes.

Verity supports several kinds of information-retrieval structures based on these categorical hierarchies, called taxonomies.

About Taxonomies

A t*axonomy* is a hierarchal organization of categories. It is typically represented as a tree structure or directed graph, with a single, most general category at the top, branching downward to subcategories, which in turn branch into deeper subcategories. For example, Figure 3-5 shows a small portion of a taxonomy that classifies automobiles by manufacturer, product line, and model name.

Figure 3-5 A taxonomy of automobile models



Note This taxonomy is not the only way to classify automobile brand names. You could devise completely different taxonomies that might instead classify automobiles by manufacturing location, consumer market segment, price, or other features. K2 allows the use of multiple taxonomies to categorize the same information; for more details, see "About Relational Taxonomies" on page 87.

Using taxonomies is a flexible approach to classification that can combine domain expertise with automatic techniques to apply a hierarchical structure to information from many different kinds of documents. Creating a taxonomy organizes your information assets into categories that are easy for users to understand and browse. There are four basic steps to implementing taxonomies in K2:

1. Build the taxonomy.
2. Create definitions for each category in the taxonomy.
3. Populate the taxonomy with documents.
4. Use the taxonomy.

Building the Taxonomy

A taxonomy is a skeletal navigation structure of named categories. It defines the views by which you want to organize your content. Thus, for a zoological taxonomy you might have a category **Animal**, with child categories **Bird** and **Cat**. These children may in turn have descendants, and so on.

The result of this stage is typically a tree structure of names. (It is possible, however, for a category to have more than one parent category, in which case the structure is more generally characterized as a directed graph.)

There are several techniques you can use to build a taxonomy:

- **Use a domain expert**. This is a common method for building a taxonomy. A domain expert builds the skeleton and assigns names to the categories. Yahoo! and some other corporations employ this method.
- **Import a taxonomy**. This technique allows users to extract implicit hierarchies from existing URL or file system hierarchies, or metadata, and mirror them in a taxonomy.
- **Purchase an industry taxonomy**. Vendors such as Lexis-Nexis or Factiva provide taxonomies for particular industry segments.
- **Use thematic mapping**. In many enterprises, the information explosion has reached a point where it is impossible for a human to envision all the various themes and topics

represented in the *corpus*—the entire collection or body of knowledge—of the enterprise. K2 provides a *thematic-mapping* capability that extracts key concepts from a set of documents, constructs a taxonomy from them, and assigns the documents to the taxonomy. Thematic mapping also generates human-readable labels for the concepts (categories) in the taxonomy.

For more information on thematic mapping, see the *Verity Knowledge Console Guide*.

Creating Category Definitions

The next step is to attach a definition to each category to control how it is populated with documents. A category definition consists of a mathematical rule against which each document can be evaluated for membership in that category. For example, a very simple rule for the category **Animal** might be "If a document contains the word *paw* or *hoof*, it belongs to this category." The result of this stage is a taxonomy with attached category definitions.

Note For certain types of taxonomies (rule-based taxonomies), category definitions are part of the taxonomy itself and do not need to be attached in a separate step.

There are several methods you can use to create category definitions:

- **Business rules**. A domain expert defines a rule for each category. Verity's powerful topics feature (see "Creating Topic Sets" on page 76) is one means for generating such business rules.
- **Import**. Document membership in categories mirrors a file system or URL hierarchy. This method corresponds to the import technique for building the taxonomy. Both the taxonomy structure and the membership of documents in categories mirrors a specified structure such as a file system.
- **Industry taxonomy**. A standards body, an independent vendor, or Verity Professional Services creates category definition rules for an industry taxonomy. This method corresponds to the industry taxonomy technique for building the taxonomy. Both the taxonomy structure and the membership of documents in categories are specified by the industry taxonomy.
- **Automatic classification**. Verity provides an automatic classification system, called the *Logistic Regression Classifier* (LRC), which uses state-of-the-art machine learning technology to perform this task. The LRC is fed positive or negative example documents for membership in each category. The system learns from these "training documents" and creates a defining rule for each category.

  For more information on the LRC, see the *Verity Knowledge Console Guide*.

Populating the Taxonomy

Once the taxonomy is built and each of its categories is defined, you can populate it with documents. The result of this stage is a fully functional taxonomy, which might be a topic set or portion of a parametric index.

You either accomplish this task either manually or automatically:

- **Manual.** Experts determine what document belongs in what categories, and they populate the category nodes in the taxonomy accordingly. Yahoo! populates its Web taxonomy in this way.
- **Automatic.** A Verity tool evaluates each document against the rule for each category and assigns the document to the appropriate categories in the taxonomy. This approach takes advantage of the business-rules category definitions described earlier.

The best approach is often to use a combination of the automatic and manual methods.

Using the Taxonomy

Studies show that with a well-implemented taxonomy, users use a balanced combination of search and browsing to locate documents. The search could be issued at the top level to filter the categories in which matching documents exist, or scoped within a subset of the taxonomy.

To accommodate the different ways that various groups use information, you can create multiple taxonomies to organize content in ways that make the most sense for each group. For example, separate taxonomies can be created for the sales, marketing, human resources, and engineering departments. This puts information into the context of your overall business model, and adds a valuable dimension to the content discovery process.

About Relational Taxonomies

The use of multiple taxonomies to categorize the same information is implemented as the *relational taxonomies* feature of parametric search. Using relational taxonomies, end users can simultaneously navigate more than one taxonomy, drilling down and navigating to the information they seek in the manner most intuitive to them.

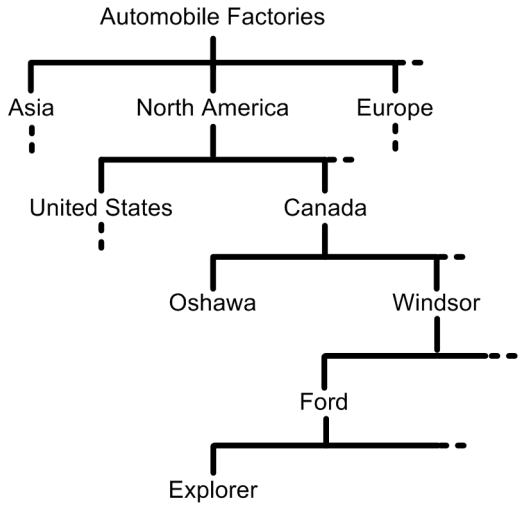
Relational taxonomies are useful in the common situation where a document can logically belong to multiple information structures, or where a business has multiple organizational perspectives for classifying the same information. For example, to help a user find a document about software companies in Sunnyvale, which of the following taxonomies should an application provide?

- **Drill down by location**. Follow links from **USA** to **California** to **Sunnyvale**, hoping to find a **Software Companies** category.
- **Drill down by business category**. Follow links from **Businesses** to **Computers** to **Software**, hoping to find a **Sunnyvale** category.

With relational taxonomies, the user is not bound to the taxonomy structure envisioned by the taxonomy creator. Instead, users can drill down and navigate as they wish, jumping from taxonomy to taxonomy wherever appropriate.

Consider a taxonomy showing the worldwide locations of car manufacturing plants. Figure 3-6 illustrates a portion of that taxonomy.

Figure 3-6 A taxonomy of automobile manufacturing plants



In this taxonomy, automobile manufacturing is organized geographically. Following one path shows that Ford Explorers are manufactured in (among other places) Windsor, Ontario in Canada. Comparing this taxonomy with the one presented earlier in Figure 3-5 on page 84 shows that a relationship clearly exists between the two taxonomies. Both can be followed to arrive at the same document.

Each taxonomy has a static navigation path. However, when two or more taxonomies share overlapping sets of documents (such as documents describing the Ford Explorer in this example), they have a relationship that allows the trees, conceptually, to be traversed dynamically. This dynamic traversal effect occurs because selections from one taxonomy are dependent on the current selections from the other taxonomies.

Consider this traversal path through the two taxonomies:

1. The user starts the traversal from the root (Automobile models) of Figure 3-5 on page 84, then selects the manufacturer Ford. Only Ford-related documents are selected.
2. The user then jumps to the Windsor link in the taxonomy of Figure 3-6. Now, only documents related to Ford vehicles manufactured in Windsor, Ontario are selected.
3. The user navigates to and selects the Explorer link, obtaining a list of documents related to Explorer manufacturing in Windsor.

Conversely, the traversal could follow the opposite path:

1. The user starts the traversal from the root (Automobile factories) of Figure 3-6, then follows the links down to Windsor. Only documents related to vehicles manufactured in Windsor, Ontario are selected.
2. The user then jumps to the Ford link in the taxonomy of Figure 3-5 on page 84. Now, only documents related to Ford vehicles manufactured in Windsor, Ontario are selected.
3. The user follows that taxonomy down through Ford trucks to the Explorer link, obtaining a list of documents related to Explorer manufacturing in Windsor.

For an example of the interface an application might use to present relational taxonomies to the user, see "Relational Taxonomies" on page 108.

Creating Profile Nets

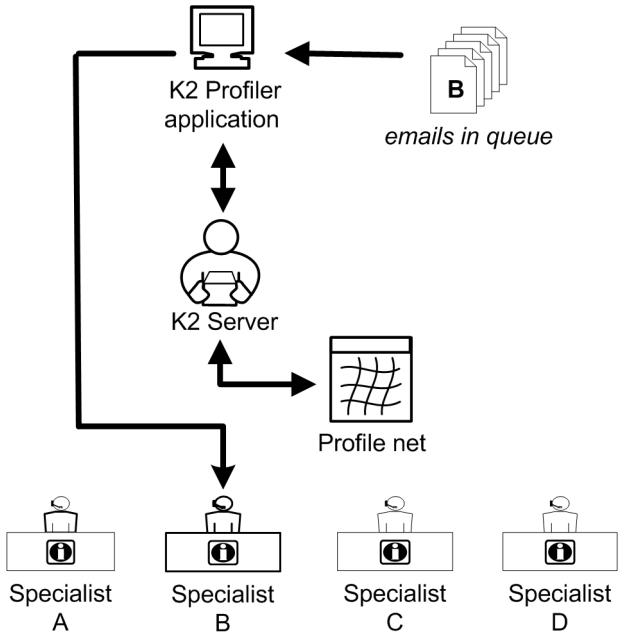
With the *K2* Profile Service, developers can build applications that classify documents for purposes other than taxonomy browsing.

The Profile Service is based on sophisticated technology that matches the terms and concepts in incoming documents to *profile nets*, which are sets of topics (stored queries) called *interest profiles*. The Profiler Service uses interest profiles to determine whether a given document is about a given subject. The application can then assign the document to the appropriate category or categories, which can in turn trigger messaging systems or cause the documents to be sent directly to interested users. The Profiler model is ideal for document classification, message routing, or push technologies.

(Note that, in contrast to the Verity engine, which indexes a number of documents and then evaluates individual queries against them, the Profile Service indexes a number of queries and then evaluates individual documents against them.)

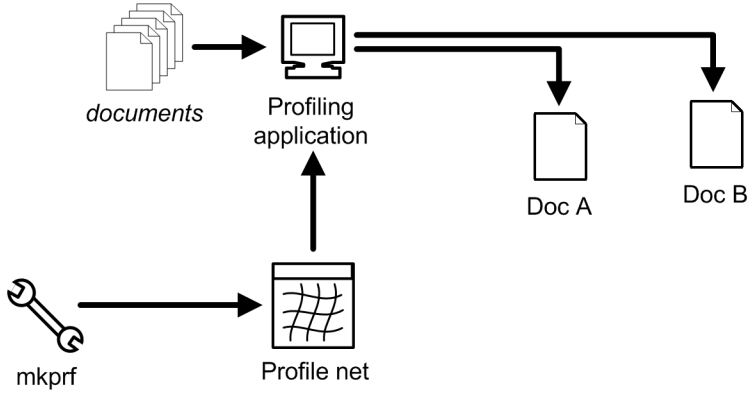
For example, each email message arriving into a customer response system (Figure 3-7) might need to be routed to one of several service specialists. The Profiler application can match the content of each email to the stored interest profiles, and thereby know which specialist should receive the message.

Figure 3-7 Using the K2 Profiler for email routing



In this example, the content of email B most closely matches the profile associated with support specialist B, so the application sends the email to B.

The Verity administrator creates the interest profiles and compiles them into a profile net by using the `mkprf` command-line tool. Each profile in a profile net is a topic. In fact, a topic set can be converted directly into a profile net.



For more information on creating interest profiles and writing a K2 Profiler application, see the *Verity K2 Profiler Programming Guide*.

Providing Recommendations

In the Verity architecture (see Figure 1-2 on page 20), the most sophisticated information access service is *recommendation*, the expression of social networks. Recommendation focuses on the relationships between people in an enterprise, as expressed by the documents they manipulate and the actions they take. The portion of K2 that implements social networks is the Recommendation Engine.

The Recommendation Engine

The Recommendation Engine is a K2 component designed to enhance the end-user's information discovery experience using a powerful and flexible matching engine. In addition, the Recommendation Engine also provides a platform for enabling social networks. Social networks help in modeling people and their relationships with information and other people.

The Recommendation Engine allows the user to find arbitrary correlations across *entities* of different types. Common examples of entities are documents, users and queries. In addition, the Recommendation Engine allows for user-defined entities, such as categories, frequently asked questions (FAQs), virtual documents, and so on. For example in ecommerce applications, you can create entities representing classes of product families (such as BOOKs, CDs, and DVDs) and control what entities are recommended for a given operation, or what entities are used as inputs to a recommendation.

For more information on using the Recommendation Engine, see the *Verity K2 Dashboard Administrator Guide* and the *Verity K2 Recommendation Engine Guide.*

The Recommendation Engine is built on three components that let you take advantage of social networks:

- Tensor Matching Engine
- Entity Profiles
- Transactions

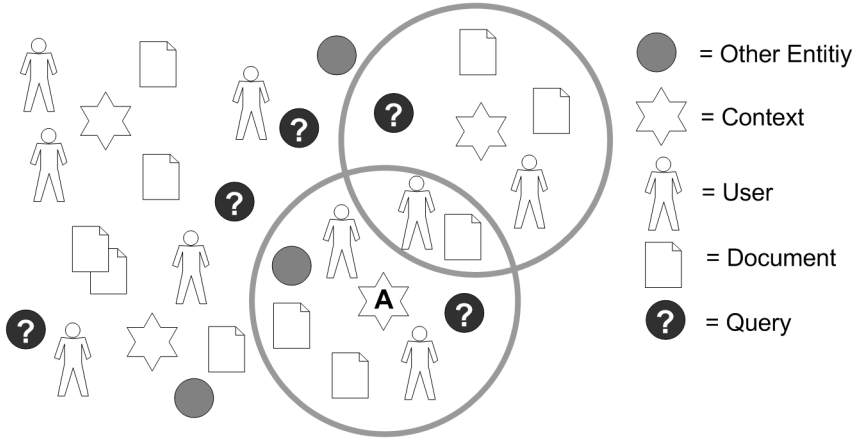
Tensor Matching Engine

The Recommendation Engine maintains entity profiles that are managed and manipulated as vectors using a technology called Tensor Space Analysis (TSA). An *entity profile* is essentially the mathematical representation of a document or user on multiple planes in the tensor space. The *tensor space* creates a dimension for each unique term the Recommendation Engine encounters. In simple terms, each entity is represented by a collection of words. Each word has a weight associated with it, and the Recommendation Engine can measure the proximity of entities by using these weights.

This technology makes it possible for the Recommendation Engine to define an *input context*, which is a combination of a set of profiles based on entities such as the user's identity, the query used, and the document being viewed. The Recommendation Engine represents the input context in the same tensor space as the document, user, and other entity profiles. This allows the retrieval of the closest, or most relevant entities. For example, if profiles are defined for users who are considered experts on a topic, only documents related to users who are experts on the current topic of interest (as defined by the input context) are retrieved by the query. The documents retrieved are those determined to be the most relevant to the input context; for example, the documents marked as "A" in Figure 3-8.

Figure 3-8 illustrates how users, documents, queries, other entities, and context coexist in the tensor space.

Figure 3-8 Tensor space example



Entity Profiles

Entity profiles are dynamic representations that continually adapt to user interactions. The essence of the Recommendation Engine lies in the creation and updating of entity profiles, and the subsequent utilization of these profiles in application programming interfaces (APIs).

A *document profile* can be created over time from the following evolving information:

- the content of the document
- relevance of query terms to the document content

Document profiles are created on a per-collection basis, and reside in a recommendation index called the *RE Doc Index*. The document profiles are initially seeded with the information stored about the documents in a Verity collection. Any fields in the collection can be used to contribute to the initial vectors of the documents in the RE Doc Index.

Over time, a user's interest profile can be affected by information, such as the following:

- a list of interests, submitted via a form
- queries submitted to the system by that user
- documents rated or viewed

User profiles are stored in a recommendation index called the *RE User Index*. It is possible to separate a user's interest profile from the user's expertise. This can be done by creating a user-defined RE Index, of type “expert”, with an alias, say, “Technical Support”. These expertise profiles can be created in the following ways:

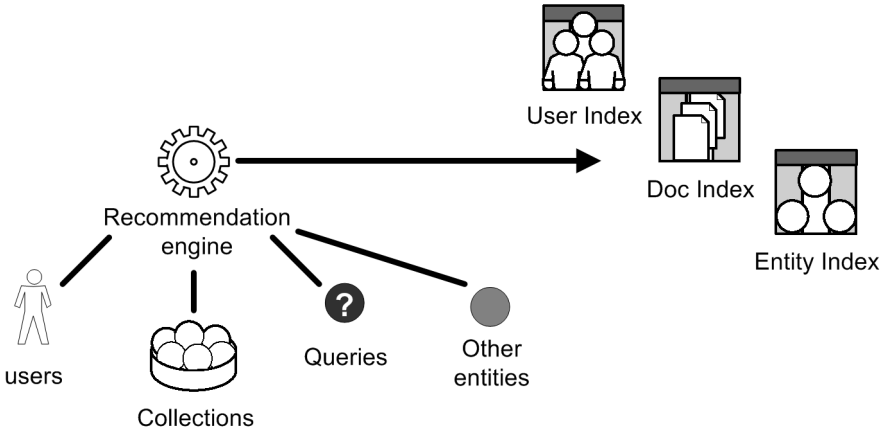
- Manually assembled by the domain expert
- Imported from specific organization databases, such as resume databases
- Automatically generated from documents, such as analyst reports, authored by the experts

The Recommendation Engine automatically maintains the profiles; however, the user can optionally control (edit) the contents of the profile for tuning.

Transactions

The tensors representing users and documents change over time due to user interactions. These interactions are recorded as Recommendation Engine *transactions*, which represent the interactions between people and information in the enterprise and therefore encapsulate the constantly evolving social network. If a user changes projects, and the user's interests change, the system is able to track these changes, so that the system provides improved relevance with minimal administration overhead.

The K 2 Recommendation Engine continually gathers this transaction information and dynamically updates the profiles.



The transaction interface is exercised whenever the user interacts with the system—as when selecting a document of interest, submitting a purchase order, or explicitly rating the relevance of an item of information to the query that retrieved it. Popular documents will appear in many transactions, and therefore their profiles will evolve more rapidly than those of less popular documents.

Recommendation applications can choose which interactions to consider as transactions. While one enterprise will find value in tracking user click streams, another might choose to consider only explicit user ranking and feedback. The application can determine which entities are included in a transaction and what relevance to assign to it.

Setting Up Recommendation Indexes

Implementing a solution that uses the Recommendation Engine involves both developing the application itself and instituting a process for creating and updating the user and document profiles.

Both user and document profiles evolve over time, but it may be important to seed them initially with sufficient information to make them immediately useful. As noted earlier, document profiles are seeded with automatically extracted feature information that is generated when the documents are first indexed into a collection. User profiles do not automatically contain any initial information, but the Recommendation Engine provides an XML import interface you can use to seed that initial information. The information you import can come from many sources, including the following:

- Customer data from a Customer Relationship Management (CRM) system

- Employee data from human resource or LDAP databases
- Job descriptions from organization charts
- Document relationships from clustering algorithms

For more information on the user-visible features a recommendation application can implement, see "Implementing Recommendations" on page 110. For more information or recommendation indexes and entity profiles, see the *Verity K2 Recommendation Engine Guide*.

4

Delivering Information to Users

In most K2 installations, developers—software designers and programmers—are responsible for building the customized K2 applications or portals that deliver enterprise information to users. Developers are also responsible for integrating Verity K2 with the enterprise's other systems.

This chapter contains information primarily of interest to developers. It gives a brief overview of K2 application development. For a more complete introduction to the options and interfaces available to Verity application developers, see the *Verity Developer Getting Started Guide*.

This chapter contains the following sections:

Providing Search Capability

Text search is the most basic and common feature of K2 applications. Once your administrator has built indexed collections of documents and has added higher-level classification and personalization information, your enterprise's users can then take advantage of your application or portal to access the information structures constructed for them.

K2 gives you flexibility in creating your application's presentation interface. You can write your own presentation interface or integrate K2 with presentation interfaces from third-party vendors. To support development, K2 provides Java and C APIs (including a .NET interface), plus sample application components. If your enterprise uses a presentation interface from a portal vendor, such as Oracle, BEA or IBM, you can use these APIs to integrate K2. You can also use the APIs to write your own presentation interface, should your enterprise require it.

The main function of your application will be to support the types of searching, browsing, and recommendation features that are most helpful to your users.

VQL and Search Operators

K2 search is based on the Verity Query Language (VQL). VQL is a rich language for writing queries that return relevant information. VQL includes a large number of operators and modifiers that offer users many options for conducting specific, sophisticated types of search. Your application can implement support for some or all of the available search types.

As one example, the proximity operator `<NEAR>` allows users to search for occurrences of terms that are close to each other (but not necessarily adjacent) in the indexed documents. A query string to perform a proximity search might look like this:

```
history <NEAR> philosophy
```

While it is possible for your application to make such VQL operators directly available to end users, it is more typical (and user-friendly) to hide this level of complexity, instead using checkboxes, buttons, or menus to allow users to select the type of search desired. The user then enters the specific terms to be searched, and your application inserts the appropriate operators and modifiers before submitting the search query to K2.

For some search features described here, support from administrators is also required, in the form of configuration settings made at indexing time so that the necessary extra data structures are created.

By writing complete VQL support into your application, you can turn all your users into search experts even before they start using K2. For complete information about the Verity Query Language and its operators, see the *Verity Query Language and Topic Guide*.

Query Parsers

K2 offers a number of different query parsers for your application to use. Each parser provides different search capabilities. Note that the same search query can yield different results with different parsers.

- **Simple query parser (full-text parser).** This is the standard parser for text searches and for searches over document fields and zones. It has full support for VQL.
- **Free-text query parser.** This parser automatically performs a stemmed search, and it scores results based solely on term frequency in the document. Simple, short words in the query term are stripped out before the search is conducted. This parser does not support VQL operators.
- **Internet-style query parser.** This parser lets users enter familiar Web-search-style commands to find information. It supports case-insensitive searching, quoted phrases, natural-language questions, and the use of plus and minus signs to include or exclude specific terms. The Internet-style query parser does not support VQL operators.
- **BooleanPlus query parser (explicit parser).** This parser is similar to the simple parser in that it supports all of VQL and it supports field and zone searches. Unlike the simple parser, however, this parser requires explicit VQL query syntax in all cases. For this reason, the BooleanPlus query parser typically is not used in end-user search forms.

Other kinds of query parsers can be developed and integrated into K2. Query parsers are described in the *Verity Query Language and Topic Guide*.

Using the simple query parser, your application can apply the types of search described here to one or more indexed collections. Some variations of full-text search are available only if the collections have been indexed with the proper configuration settings.

Implementing Search

The K2 search features provide your application with many choices for giving users precise results. By using the correct type of search, users can quickly find the information they need. Even one-word queries can return accurate results.

Your application conducts searches by making calls to the K2 VSearch Java API or the Verity Client C API.

Simple Search

By default, single words are searched case-insensitively and strings of words are searched as phrases. Your application can force case-sensitive searching by adding the `<CASE>` operator to the user's search term. Your application can force a search for the individual words in a phrase (such as `air conditioning`) by converting the string to a comma-separated list of terms (`air,conditioning`) before submitting the query to K2.

Stemmed Search

Stemmed search is a fuzzy search (an inexact search) that returns occurrences of indexed words whose word stems match the search term. For example, a stemmed search for **dance** returns documents that contain **dance** or **dancer** or **dances**.

To conduct a stemmed search, your application adds the `<STEM>` operator to the user's search terms.

Typo Search

Typo search is a fuzzy search that allows your users to find and retrieve documents even when they misspell the search terms. K2 allows a customizable, limited range of deviation in spelling between search terms and their indexed equivalents. For example, a typo search for **contiment** might return documents containing either **continent** or **condiment**.

To conduct a typo search, your application adds the `<TYPO>` operator to the user's search term.

Synonym Search

Synonym search is a fuzzy search that returns occurrences of the search term or any of its synonyms. The synonyms are listed in a thesaurus file. For example, a synonym search for **brave** might return documents that contain **brave** or **courageous** or **fearless**.

To conduct a synonym search, your application adds the `<THESAURUS>` operator to the user's search term. A properly constructed and compiled thesaurus file must be installed. K2 is delivered with default thesauruses for some languages; for others, the administrator may have to create a thesaurus file, as described in "Using Thesauruses" on page 103.

Soundex Search

Soundex is a fuzzy search that retrieves documents containing terms that are phonetically similar to the search term. For example, a Soundex search for **Joan** might return documents containing either **Jean** or **Jane**.

To conduct a Soundex search, your application adds the `<SOUNDEX>` operator to the user's search term.

Word stems and indexed synonyms can be used in search terms, and relevancy ranking to assign a user-identified level of importance to documents. Wild cards can also be employed when they know only a few characters or a characteristic of a string being searched.

Wildcard Search

In *wildcard search*, users can substitute a wildcard character when they know only some of the characters of the term they are searching for. For example, a wildcard search for **ta*l** returns documents that contain **tail** or **tall** or **tactical**.

To conduct a wildcard search, your application adds the `<WILDCARD>` operator to the user's search term.

Language-Specific Search

K2 includes support for search and display of documents in multiple languages. An application can be licensed to support one or more *locales*, each of which allows a user to search according to the rules of the locale's language.

Also, if your K2 installation is using the multilanguage locale (see "Locales" on page 55), users of your application are able to conduct stemmed searches in any of the locale's languages for which your installation is licensed.

For example, a single collection might include documents in English, French, and Japanese. The user can select a language and enter a term, and then your application can construct a query term like this:

```
<LANG/fr><STEM>fort
```

in which case only documents containing French words whose stem is **fort** will be returned.

Accent-Insensitive Search

Depending on the language of the documents being searched, searches with the simple query parser are by default either accent-insensitive or accent-sensitive. In an *accent-insensitive search,* using the search term **resume** returns documents that contain **resume** or **resumé** or **résumé**. In an accent-sensitive search, using the search term **resume** returns only documents that contain **resume**.

Converting your application between accent-insensitive and accent-sensitive searching is not a programming task; it is a configuration task performed by the administrator, as explained in the *Verity Locale Configuration Guide*.

Using Stop-Word Lists

Stop-word lists are files that contain words that should not be searched for. Verity supports several different types of stop-word lists. Some are applied to the collection at indexing time to prevent the stop words from appearing in the index, whereas others are applied at search time to strip the stop words from the search query.

You can create stop-word lists as part of the application-development process, or the administrator can create and modify stop-word lists as part of maintaining the K2 installation.

Stop-word lists are described in more detail in the *Verity Collection Reference* and the *Verity Locale Configuration Guide*.

Providing Topic Search

The administrator can create topic sets and index them into collections (see "Creating Topic Sets" on page 76). If your application is using the simple query parser, submitted queries are automatically compared to topic names and, if there is a match, the topic search is used—that is, the topic's query is submitted to the Verity search engine instead of the user's original term.

Using Thesauruses

Presenting Search Results

Most K2 applications implement a search-results page, on which the names of documents that match the user's search terms are listed, along with information about—and a link to—each result document.

Results can be paginated, both to avoid requiring the user to scroll through long lists of results and to allow the application to present results faster.

Your application's search-results page can also include the features described in this section.

Clustering Results

K2 applications can use document *clustering* to group related documents on a search results page. Clustering documents into groups can help give the user a sense of the main subject areas covered in a set of search results. For example, if only one of several document groups in the search results is of interest, the user can quickly focus on the most interesting documents without wasting time scanning the rest.

Clustering operates by analyzing the feature vectors in a document set and clustering documents that are more semantically similar to each other than they are to the documents in other clusters. Each document is assigned to one and only one cluster.

Document clustering is an inherently ambiguous process. There is no one "correct" grouping of documents into clusters. The number of clusters can be fixed in advance, or it can be automatically determined by the Verity engine based on the application's preference for cluster granularity. Documents are clustered on the basis of their text content only, and not on the basis of meta-information such as title or other fields.

When displaying a cluster, a Verity application can also display the most important keywords for the cluster itself, to further help the user to quickly find the most relevant information.

Returning Document Summaries

Document *summarization* is a feature that can be used to generate content summaries for documents listed on the search-results page. The goal of automatic document summarization is to accelerate the browsing of search results returned by the Verity search engine.

By presenting a short summary for each document in a results list, a Verity application can help users quickly assess the relevance of the returned documents without wasting time loading and skimming the full text of the documents.

The following types of document summarization are available in K2:

- **Static summaries**. K2 applications can support two types of static summarization:
  - Simple summarization displays information from the beginning of a document, for example the first 400 bytes.
  - Content summarization generates summaries by selecting sentences from the document that are indicative of the overall theme of the text. This kind of summarization relies on Verity feature extraction, as described in "Extracting Document Features" on page 80.

  Static summaries are enabled by the administrator and created at collection-indexing time.

- **Passage-based summaries**. A passage-based summary consists of one or more passages (sentences or phrases) from the document, each of which contains instances, usually highlighted, of the search terms that were used to locate the document. For example, with passage-based summary enabled, searching for the term *report* in a collection might yield a result like this:

  Installed Reporting Components
  ...components in the tree: The **report** server. This is a standard K2 Server whose... one **report** server in a K2 domain. Its alias is **report**_server. The ... **report** index is attached to the **report** server. There is only one **report** index in ...

  Your application displays a passage-based summary for a document by making calls to the Client C API or the VSearch Java API. Passage-based summaries must be enabled by the administrator at the time of indexing.

For more information on summarization, see the *Verity Developer's Kit Programming Reference*, the *Verity Collection Reference*, and the *Verity K2 Client Programming Guide*.

Providing Spelling Suggestion

Spelling suggestion can be used to propose corrections to mistyped words in a user's query. If a search returns no or few results, the application can display a message on the search results page, listing a suggested alternate query. For example, if the user searches for "helo wonderful worlld" and that phrase returns no hits, your application could respond with

Are you searching for "**hello** wonderful **world**"?

on the search-results page. To go farther, your application could also provide additional suggested alternates to the user's keywords:

Are you searching for "**hello** wonderful **world**"?

More suggestions: **hello, hell**, **Hilo**, **word**, **world**, **whorl**

The suggestions presented to the user are taken from the word indexes of the collections being searched. All suggested keywords can, therefore, be guaranteed to occur in at least one of the collections the user is searching.

Note In Version 5.5, spelling suggestion is supported for only single-byte languages.

To display spelling suggestions to the user, Your application makes calls to the Client C API or the VSearch Java API. To support the spelling suggestion feature, structures must be created when the collection is indexed. To generate these structures, the administrator can use the `mkvdk` command-line tool, or the administrator can set up a user-defined job (UDJ) to run from the K2 Dashboard. For more information on the required structures, see the *Verity Command-Line Indexing Reference*. For more information on user-defined jobs, see the *Verity K2 Dashboard Administrator Guide*.

Retrieving and Displaying Documents

When a user of your application clicks a link on the search results page, your application typically displays the content of the document referenced in the link. Your application can take full responsibility for performing the display, or you can use the powerful document viewing and highlighting service provided by K2. This viewing service has the following advantages:

- It converts indexed documents to HTML format for viewing directly in the user's browser.

  The viewing service also allows you to specify which template to use for HTML conversion. This capability lets you customize the conversions or tailor the output to different output devices (such as Web browsers, PDAs, and so on.).

- It includes the ability to highlight the search terms in the document.
- It supports URL-based retrieval from all Verity gateways and Web servers, Lotus Domino servers, and Microsoft Exchange Web servers. If desired, retrieved documents can have links replaced and be dynamically highlighted.
- It also supports retrieval of documents in raw, binary format from all gateways (for example, PDF documents from Notes Domino servers), in case you want your application to display the document in native format through a browser plug-in.
- It supports display of sub documents that are linked from the main document. This can include HTML links and attachments.

  Note The viewing service supports display of only those subdocuments that are indexed. It cannot display documents that are in the repository but not indexed into the collection.

- It works within the existing K2 security framework (including document-level and collection-level security).

Your application makes use of the viewing service by making calls to the K2 VView Java API.

Implementing Parametric Search

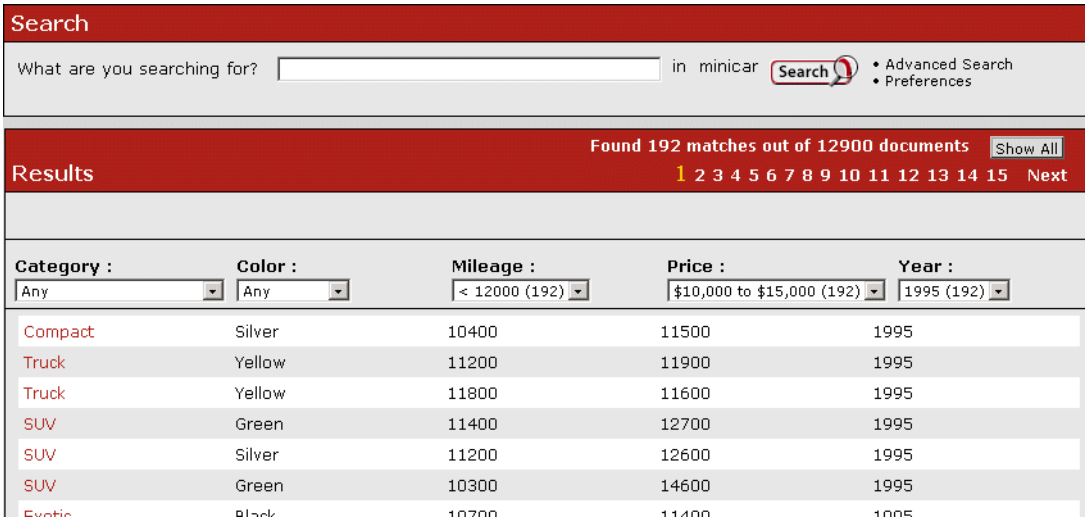
Parametric search is available to the user if your application supports it and if the administrator has created a parametric index for your repository data, as noted in "Setting Up Parametric Search" on page 77.

Parametric Selection

Supporting parametric search in your application typically involves creating a user interface that supports both parametric selection (for the user to select field values) and text search (for the user to search document content) over the specific collection on which the parametric index is built.

For example, a user might search for a used car by selecting values for categories such as color, mileage, price, and year, as show in Figure 4-1.

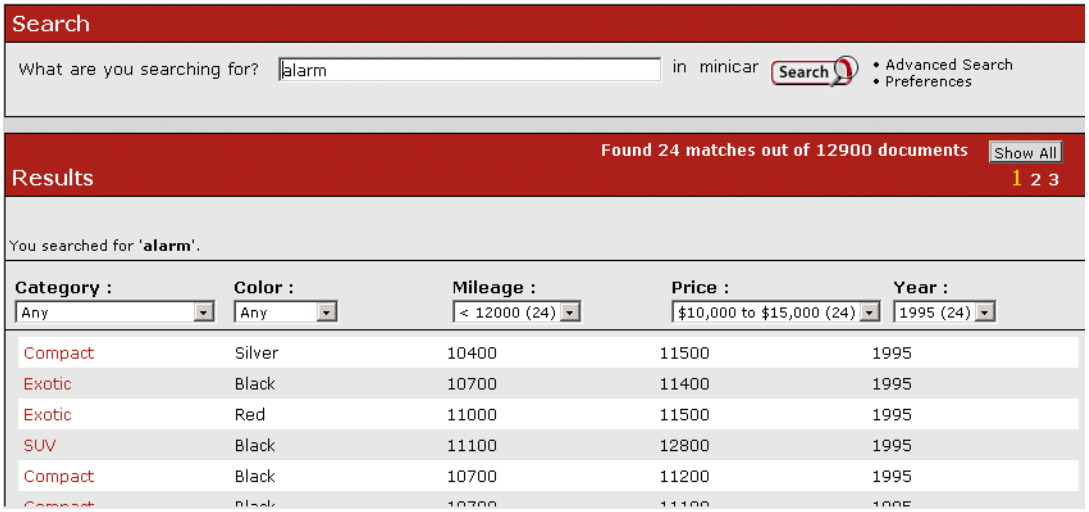
Figure 4-1 Pure parametric selection



Your application can also provide sorting capabilities to allow the user to specify how results should be ordered, whether by relevancy ranking or by the values in one or more fields.

If your application supports text search as well as parametric selection, results are restricted to the documents that match the currently selected parameters plus the search terms. In the example shown in Figure 4-2, the user searches for the name of a desired feature (a car alarm) in addition to setting parameters.

Figure 4-2 Text search combined with parametric selection



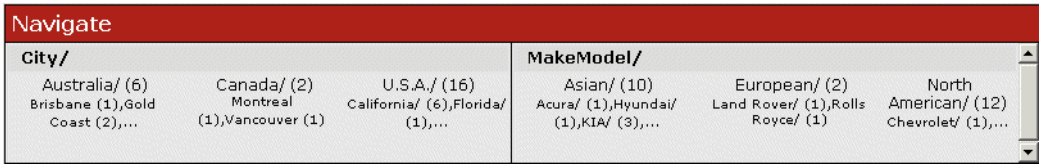
Your application conducts parametric searches by making calls to the K2 Parametric Java API.

Relational Taxonomies

You can enhance your parametric-search application to support relational taxonomies (see "About Relational Taxonomies" on page 87). With relational taxonomies, your users can navigate (drill down) through multiple taxonomies at the same time. This implies that when your results page displays the results of the most recent parametric selection, it should also display links to allow the user to browse further, based on any of the taxonomies to which the current result set belongs.

For example, Figure 4-3 shows the display of two automobile-related taxonomies: one of dealership locations, and one of manufacturer, make, and model. Links to nodes at the current level and two levels downward are displayed. (The number of documents in each node appear in parentheses.)

Figure 4-3 Displaying taxonomies for user browsing

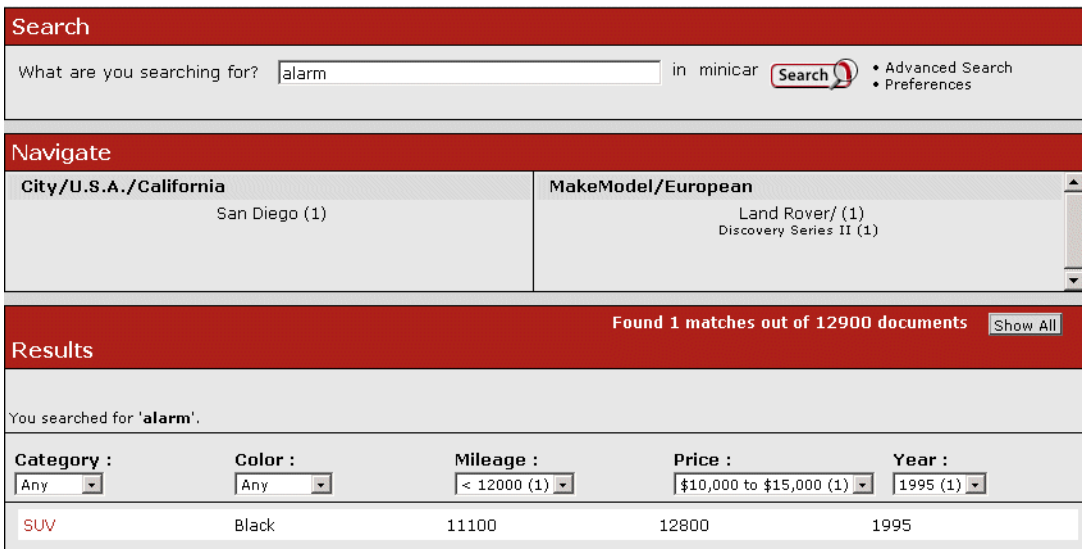


In this situation, the user can start browsing one taxonomy, then switch the other, and even back again, until arriving at a desired category or document.

Your application can dynamically render the taxonomy links presented to the user. For example, if the user clicks the Canada link on the right side of Figure 4-3, the resultant taxonomy display need not show the links for Australia and U.S.A., because those categories cannot be searched downward from that point.

Figure 4-4 shows the result of browsing the automotive taxonomies to select California dealers of European cars, then performing the same search and selections as in Figures 4-1 and 4-2.

Figure 4-4 Taxonomy browse combined with text search and parametric selection



Supporting relational taxonomies requires a parametric index that contains multiple taxonomies. Administrators can use the Verity Collaborative Classifier or other Verity tools to create the taxonomies and insert them into the parametric index.

Your application conducts parametric selections by making calls to the K2 Parametric Java API. Note that the Verity search engine can return the categories (topics) to which a document belongs when it returns a parametric selection result. Your application can use that information to display links for further browsing the associated taxonomies.

Implementing Profiling

Use of profile nets to classify or route documents is available to your application once the administrator has created the necessary interest profiles (see "Creating Profile Nets" on page 89).

Depending on its purpose, your application may or may not need an end-user interface for document classification. To perform the classifications, it makes calls to the Profiler Java API.

Implementing Recommendations

Your application can use the Recommendation Engine to provide recommendations and social-network information to users once the administrator has created the necessary user and document profiles (See "Providing Recommendations" on page 91).

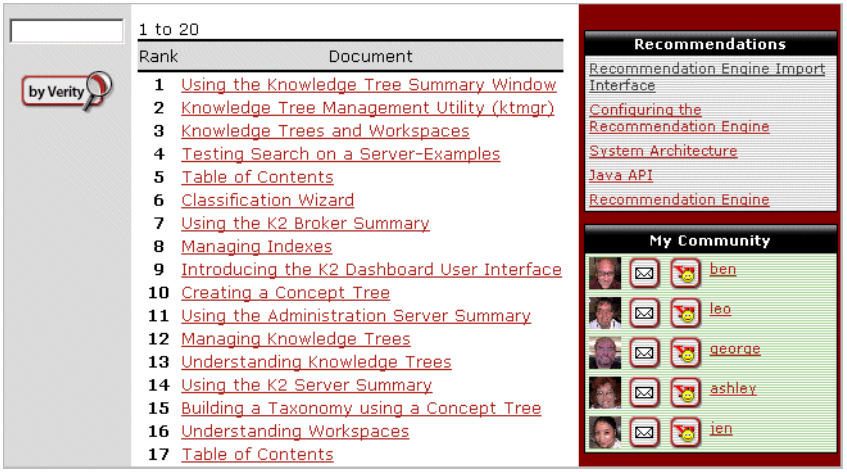
For example, the Recommendation Engine can provide recommendations for documents relevant to a user's context by analyzing what documents users have accessed or authored, what searches they have issued, which department they belong to, who their colleagues are, what documents the colleagues have accessed, and so on.

The Recommendation Engine currently makes the following features available to users of Recommendation applications.

- **Adaptive Ranking**. When the user executes a search, the Recommendation Engine returns a result set that is ordered based on the document content but also on prior access patterns from the users of the application. Thus, a document that is extremely popular with users who have issued similar queries during a certain period of time moves up in the rankings, but then drops off in the rankings when its popularity subsides.

Your application's search-result pages could include areas for displaying adaptive ranking of the results, document recommendations, and user community. Figure 4-5 shows a portion of the search results page from a sample Recommendation application. The adaptive ranking is shown in the **Rank** column beside the document links.

Figure 4-5 Ranking, recommendation, and community



- **Document Recommendation**. When the user views the results of a search, the Recommendation Engine displays a personalized list of documents to the user. The list consists of documents that most closely represent the query plus the user's profile.

  In Figure 4-5, recommended documents are under the **Recommendations** heading beside the search results.

  The recommendations can improve over time, due to *context-based personalization*. Different people performing the same search may be looking for different things. For example, people from different parts of the world may mean different things by the term "football" (soccer vs. American football). User profiles can be used to tune the results of a given request, and the system can track user interactions and automatically update the profiles.

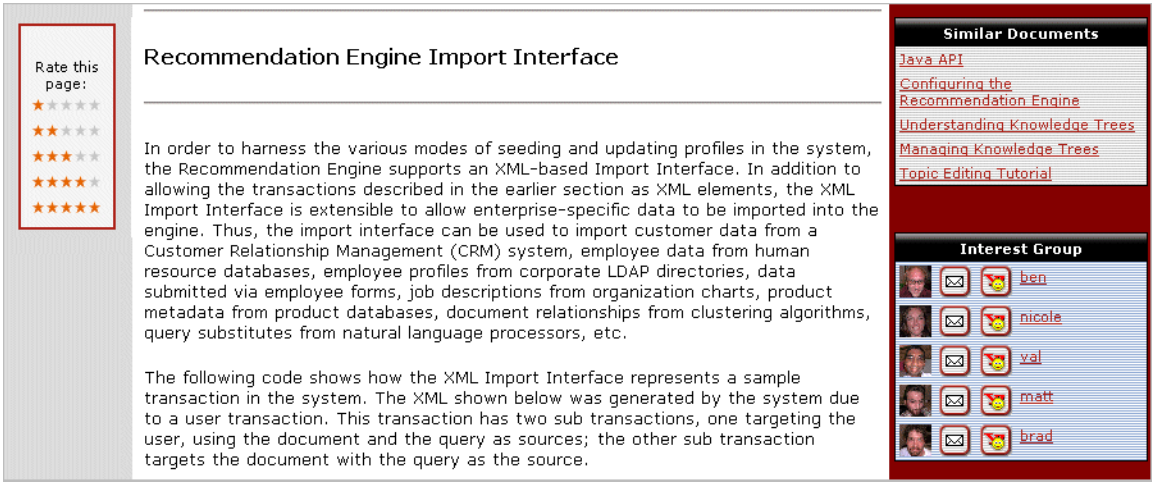
- **Community Identification**. When the user views the results of a search, the Recommendation Engine can provide a personalized list of other users whose profiles are closest to the current context (user's profile plus query).

  In Figure 4-5, similar users are identified under the **My Community** heading beside the search results.

- **Document Similarity**. When the user views the content of a document, the Recommendation Engine can provide a personalized list of other similar documents. The list shows the documents (or products, in the case of a catalog application) most relevant to the current context (document plus user's query).

  Your application's document-display pages could include areas for displaying a list of similar documents and the locations of other interested users (experts). It could also allow for rating of the current document to perform transactional updating of profiles. Figure 4-6 shows a portion of the document-viewing page from a sample Recommendation application. The list of similar documents is shown in the **Similar Documents** column beside the document content.

Figure 4-6 Document similarity and expert location



- **Expert Location**. When the user views the content of a document, the Recommendation Engine can also provide a list of other people most interested in the subject. The list identifies users whose profiles most closely match the current context (document plus user's query).

  Expert location takes recommendations a logical step beyond adaptive recommendation. As K2 analyzes the patterns of interaction between users and documents, it uncovers implicit relationships between the various users in a system. Based on this analysis, expert location can suggest experts in the enterprise whom the user can consult for a particular context. For example, to a scientist viewing a research report on widgets, it can provide a list of widget experts in the scientist's domain.

  In Figure 4-6, the subject-matter experts are identified under the **Interest Group** heading beside the document content.

- **Transactional Updating**. One way to use transactions to update profiles is by allowing users to rate documents. When the user views the content of a document, the application can also display a field or control in which the user can score the relevance of the document to the user's query. The application then uses the Recommendation Engine transaction API to update the document's profile.

  In Figure 4-6, the transactional update is triggered by clicking the rating links (stars) under **Rate This Page** beside the document content.

- **Entity Editing**. A powerful feature of the Recommendation Engine is that entities can be edited. Users can update their own user profiles directly. Administrators, or other users with special authorization, can update the profiles of other entities, such as categories. Even a query profile can be updated, so that the query becomes relevant to other contexts as well. Usually, the profiles are encrypted within the engine for privacy reasons, but the application can choose to set the profiles to be available in unencrypted form in order to use this editing capability.

- **Entity Export.** It is possible to export a set of entities into an XML file. This allows the entities to be examined, processed further via other scripts and software tools, such as data mining tools, and imported back into the engine or to another instance of the engine running on a different machine or at a different location. The function allows entities to be exported by RI type or specific to a given alias (RI Index).

  Note Entity export can be accomplished by running the `mkre -export` command.

- **Concept-Based Retrieval**. One of the limitations of keyword-based retrieval systems is that if a certain keyword is not found in a given document, the system does not retrieve it. A document's representation changes over time based on user interactions, such as queries, even if the document content itself has not changed. This allows domain-specific usage of the information to change document profiles. K2 applications can recommend relevant documents for query terms or concepts that may not exist in the document (or even the entire corpus). This happen automatically over time, without any specific administration/management effort.

- **Session-Based Profiles**. Session-based profiles are temporary and dynamic user profiles that can be used to track relevant searches and purchases so that similar products can also be recommended. Recommendations are geared the user's current context, rather than using historical information. Thus, recommendations made to a user browsing a catalog can be based on earlier transactions within the same session.

- **Dynamic Taxonomies**. The recommendation engine allows extremely simple creation of user-defined categories. These user-defined categories can live in their own namespace, or can be integrated into the organization's taxonomy via the other taxonomy management tools provided with K2.

Your application obtains recommendation information and performs transactional updates by making calls to the `VRecommendation` Java API and `VTransaction` API.

5

Developing Your Application

In most K2 installations, software designers and programmers are the ones responsible for building customized K2 applications or portals that deliver enterprise information to users. Programmers are also responsible for integrating Verity K2 with the enterprise's other systems.

This chapter contains information primarily of interest to developers. It gives a brief overview of K2 application development. For a more complete introduction to the options and interfaces available to Verity application developers, see the *Verity Developer Getting Started Guide*.

This chapter contains the following sections:

Developing K2 Applications

For developing K2 applications, you can use the Verity Component Framework or you can use the Java or C APIs. (You can also use the APIs when programming in a .NET language such as C#.)

These APIs allow you to access and customize the features of K2 to help build your client application. Lower-level APIs are also available for developing driver software modules.

Using the Component Framework

The simplest and most-recommended method for creating a K2 application may be to construct it using the Verity Component Framework. With the Frame work you can easily build software components and link them together into a functioning application.

A Component Framework application implements a search capability using the Verity K2 search engine. You create a Component Framework application by defining components and related entities that define the user interface for your application and by placing calls to `VComponent` methods in a set of JSP or .NET pages to render the interface.

You define the application and its components by using the Application module of the Verity Business Console (see "Managing Information With the Business Console" on page 66). and integrate them into your JSP or .NET pages using the following procedure:

1. Create a new application using the Application module of the Business Console.
2. Create a JSP or .NET page and set it up to use your components.
3. Specify a cascading style sheet (CSS) to use with your JSP or .NET page.
4. Create one or more components in the Business Console Application module. For example, create a search box (Search component) and a results list (Results component).
5. Add the `VComponent.draw` method to your JSP or .NET page, to render the component in a browser window.
6. Publish your application from the Business Console.

See the *Verity Component Framework Developer Guide* for more details.

Using Java-Language K2 APIs

Many K2 applications have a user interface written in Java/JSP. For collection search, programming examples in Java are provided in the *Verity K2 Client Programming Guide*. For parametric indexes, programming examples of JSP interface elements are provided in the *Verity K2 Parametric Developer Guide*. For recommendations, examples of JSP interface elements are provided in the *Verity K2 Recommendation Engine Guide*.

K2 Java API Guides provide easy integration into a wide range of application servers and environments, including Java Server Pages, the Java™ Platform, and the .NET environment.

The K2 Java APIs provide a high-level object exposure that allows clients to communicate over a socket with a K2 Broker or K2 Server. These clients can be based on Java objects (using Java Server Pages).

Client APIs

- **VSearch Java API**. Interface to Verity search functionality; used to perform collection-level search and tree search, to obtain system information, to authenticate access to repositories, to perform K2 login, and to obtain user/group information.

  Documented in the `VSearch` JavaDoc document.

- **VView Java API**. Interface to the Verity viewing service; used to convert indexed documents to HTML format—based in various templates—and display the results to the user.

  Documented in the `VView` JavaDoc document.

- **Parametric Java API**. Interface to the parametric engine and to parts of ODK; used to manipulate parametric indexes, conduct parametric selection, and render results.

  Documented in the `Parametric` JavaDoc document and the *Verity K2 Parametric Developer Guide*.

- **ODK Java APIs**. Interfaces to the Organization Developer's Kit (ODK); used to provide classification features for documents, including taxonomies and topic sets. Includes programmatic support for collection indexing, thematic mapping, and the Logistic Regression Classifier (LRC). ODK makes use of the Parametric Java API to implement taxonomies within a parametric index.

  Documented in the `Parametric` JavaDoc document and the *Verity Organization Developer's Kit Programming Guide*.

- **Profiler Java API**. Interface to the K2 Profiler service; used to evaluate documents against a set of predefined profiles (queries) to obtain a list of profiles matched to each document, plus relevance scores.

  Documented in the `Profiler` JavaDoc document and the *Verity K2 Profiler Programming Guide*.

- **VRecommendation Java API**. Interface to the Recommendation Engine; used to find desired users or documents given an input context.

  Documented in the `VSearch` JavaDoc document.

- **VTransaction Java API**. Interface to the Recommendation Engine; used to provide feedback to the Recommendation Engine to update document or user profiles.

  Documented in the `VSearch` JavaDoc document.

- **FDK Java API**. Interface to the Federated Developer's Kit; used to access the federated infrastructure and Verity Federator, to execute simultaneous searches on multiple search engines.

Documented in the Verity Federated Developer's Kit documentation.

Administration APIs

- **VAdministration Java API**. Interface to K2 administrative functionality; used to change and manage the K2 system configuration. This API provides the same functionality as K2 Dashboard or the `rcadmin` command-line tool.

  Documented in the `VAdministration` JavaDoc document.

- **VIndex Java API**. Interface to the K2 Spider controller; used to administer indexing of collections. This API provides the same functionality as the K2 Spider C API.

  Documented in the `VIndex` JavaDoc document.

Using C-Language K2 APIs

The K2 Developer product includes C APIs that allow application programmers to quickly and efficiently create customized user applications for searching and displaying the results of search operations. For collection search, programming examples in C are provided in the *Verity K2 Client Programming Guide*.

The K2 client C API uses a simple, connectionless HTTP-style protocol, which reduces latency and overhead for most common operations by bundling common API usage scenarios into a single call. As a consequence, latency is greatly reduced, leading to highly responsive applications.

The K2 client API is specifically designed to support asynchronous operation, so that a single instance of the client application can accept user input while the K2 Server processes the request.

The K2 application-level C APIs include the following:

- **Client C API**. Interface to Verity search functionality; used to perform collection-level search and tree search, to obtain system information, to authenticate to repositories, to perform K2 login, and to obtain user/group information.

  Documented in the *Verity K2 Client Programming Guide*.

- **Viewing Service C API**. Interface to the Verity viewing service; used to convert indexed documents to HTML format—based in various templates—and display the results to the user.

  Documented in the *Verity K2 Viewing Service Programming Guide*.

- **Spider C API**. Interface to the K2 Spider controller; used to administer indexing of collections. This API provides the same functionality as the VIndex Java API.

Documented in the *Verity K2 Spider Programming Guide*.

- **Profiler C API**. Interface to the K2 Profiler service; used to evaluate documents against a set of predefined profiles (queries) to obtain a list of profiles matched to each document, plus relevance scores.

  Documented in the *Verity K2 Profiler Programming Guide*.

- **Administration C API**. Interface to K2 administrative functionality; used to change and manage the K2 system configuration. This API provides the same functionality as K2 Dashboard or the `rcadmin` command-line tool.

  Documented in the *Verity K2 Administration Programming Guide*.

- **Security API**. Interface to the Verity Ticket Server; used to add or remove administrators or users and to manipulate security information.

  Documented in the *Verity K2 Security Programming Guide*.

- **ODK C APIs**. Interfaces to ODK; used to conduct parametric selections, use thematic mapping and the LRC, and manipulate topic sets, parametric indexes, and parametric trees.

  Documented in the *Verity Organization Developer's Kit Programming Guide*

.NET Development

Verity K2 allows .NET developers to write C#, J#, or Visual Basic applications that call the Verity APIs and run natively in the Microsoft .NET Common Language Runtime (CLR). The Verity-provided dynamic link library `k2dotnet.dll`, which is installed as an option when you install K2, is required.

Note Documentation for Verity APIs is provided only in Javadoc format.

Developing VDK Applications

Verity also provides APIs for development at the lower-level VDK layer. These APIs are available in C language only.

Developing for the VDK layer means that you are calling the Verity engine directly. Therefore, the scalable, distributed functionality of the K2 Broker/K2 Server architecture is not available.

- Documented in the *Verity Developer's Kit Programming Reference*:
  - **Session suite.** Provides a context for VDK operations.
  - **Security suite.** Enables document-level security. The API provides the ability to identify users, their credentials and certificates, and the ability to authenticate users against documents in repositories.
  - **Service suite.** Provides the ability to perform operations in the background.
  - **Collection Maintenance suite.** Provides the ability to create collections, index them, and perform other maintenance functions.
  - **Search Processing suite.** Provides basic search and retrieval capabilities.
  - **Query Parser suite.** Provides the ability to specify how the Verity engine interprets query parameters.
  - **Document Access suite.** Provides the ability to access documents, read their fields, and highlight their contents.
  - **Clustering and Summarization suite.** Provides advanced search capabilities, including the ability to cluster similar documents and summarize the subject of a document.
  - **Assists suite.** Provides the ability to assist end users when they specify queries.
  - **Knowledge Base suite.** Provides the ability to define sets of search criteria that populate knowledge bases.
  - **Transitory Fields suite.** Provides the ability to create temporary fields in a document.
- Documented in the *Verity Profiler Programming Guide*:
  - **Profiler suite.** Provide the ability to evaluate documents against a set of predefined profiles (queries) to obtain a list of profiles matched to each document, plus relevance scores.
- Documented in the *Verity Organization Developer's Kit Programming Guide*:
  - **Parametric suite.** Provides the ability to administer parametric indexes and their bucket sets.
  - **Taxonomy suite.** Provides the ability to create, edit, and navigate taxonomies.
  - **Topic Set suite.** Provides the ability to create, edit, import and index topic sets.

- **Thematic Mapping suite.** Provides the ability to extract concepts and build a taxonomy and topic set from a corpus of documents.
- **Logistic Regression Classification suite.** Provides the ability to automatically create topics from example documents.

Developing Driver Modules

Verity provides limited C-language APIs for developing low-level driver modules for the Verity DDA layer. Most driver modules (gateways, document filters, and locales) are developed by Verity itself.

The following are the DDA-level C APIs:

- **Gateway Driver APIs.** Support the development of custom gateways that access new kinds of information repositories.

  Documented in the *Verity Gateway Developer's Kit Programming Reference*.

- **Custom Filter APIs.** Support the development of custom document filters that read and process new document formats.

  Documented in the *Verity Developer's Kit Programming Reference*.

A

Contacting Verity Technical Support

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

Telephone: (403) 294-1107

Fax: (403) 750-4100

Email: tech-support@verity.com

Web: http://www.verity.com

Opening a Technical Support Incident (TSI)

To open a Technical Support Incident (TSI) with Verity's Technical Support, have your Entity ID number available and ready (or provided in your initial e-mail). Technical Support cannot assist you if you cannot provide a valid Entity ID number. The Entity ID number can be found on your License Key Sheet which was supplied when you received your Verity product. Please try to have as much information as possible ready for the Technical Support Specialist, including

- The product you are currently using.
- The product's version number.
- What you are having problems with.
- Any commands you are running.

For example, if you are having problems indexing with `mkvdk`, you will need to supply the full `mkvdk` command you are using. Be prepared to send in your complete set of style files for the collections in question. These files provide important insight on how your collection is built.

If your problem includes a Verity Product that produces error code numbers and messages, please carefully copy the entire error message (be precise with the spelling) and have it ready to send to Technical Support. Often the error number is accompanied by text. The text is generally more important than the actual error number and will often provide a clearer understanding of your problem.

If you are having difficulties searching, you will probably be asked to use `rcvdk` and/or `rck2` to try to replicate the problem. These two tools are quite valuable in tracking down problems related to both indexing and searching. It is to your advantage to become familiar with them. You might also be asked to send in your search templates and/or client search code. If you are having difficulties indexing a particular document, or type of document, you will need to send the documents to the Technical Support Specialist.

In some cases, you will be asked to send your entire collection. This is not uncommon and is sometimes the only way to narrow down a problem. We realize that some clients' collections are very large, or contain sensitive data, but having the collection locally in Technical Support can be critical to resolving the issues at hand. In these cases Verity will supply instructions for FTPing the collections to Technical Support. If necessary, Tech Support Management can sign a non-disclosure agreement (NDA) to protect your sensitive data.

When you open a TSI with Technical Support, your TSI will be given a number. This number is used to track individual cases and will link you with your support specialist. On subsequent phone calls, mention the TSI number to the technician who answers the phone and you will be immediately transferred to your support specialist. If using email, please include the TSI number as part of the subject line. This helps to ensure that the correct technician picks up your email quickly.

Product Support Hints

- To find out version numbers when using `mkvdk`:

  Type `mkvdk` on the command line. The first line of output will contain the relevant information, such as

  ```
  mkvdk - Verity, Inc. Version 4.5.1 (_nti40, Aug 8 2002)
  ```

- To find out version numbers when using `vspider`:

  Type vspider on the command line. The first line of output will contain the relevant information, such as

  ```
  vspider - Verity, Inc. Version K4.5.1 (_nti40, Aug 8 2002)
  ```

- To find out version numbers when using K2:

  Type `k2server` on the command line. You may need to actually start the server to get the full information. The output will contain the relevant information, such as

  ```
  K2SERVER - Verity, Inc. Version 4.5.1 ( Build 20020809 )
  ```

Glossary

accent-insensitive search	A type of search that includes all accented variations of a letter in the search term. In accent-insensitive search, the search term *si* would find all instances of both *si* or *sí*, for example. Conversely, in accent-sensitive search, the search term *si* would find only instances of the unaccented *si*.
Active Server Page (ASP)	A Windows-specific file for generating Web pages. It contains a combination of server-side scripting, HTML, and COM components. The programming language can be Visual Basic or **JavaScript** and the Web server must be ASP aware. Compare **Java Server Page (JSP)**.
adaptive ranking	The scoring and ranking of documents based on the historical behavior of users who have issued similar searches.
administrator	Design, install, configure, and maintain the K2 installation. They may also create collections and taxonomies.
Administration Server	A repository for, and synchronizer of, configuration information. In a K2 domain, there is one Administration Server for every host. Compare Master Administration Server.

authentication The process of identifying a user by passing credentials to a secure server, such as a K2 Server, K2 Broker, or K2 Ticket Server.

`browse` A command-line tool that lists the contents (field names and values) of a collection's **document table**.

bucket A value or range of values for a parameter in a **parametric index**. It identifies a set of like documents for **parametric selection**.

bucket set The set of all **buckets** associated with a parameter in a **parametric index**.

category A subject of interest used to collate documents that are relevant to the subject. Logically, a category represents a node in a taxonomy.

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.

classification The process of assigning documents to categories in a **taxonomy**.

cluster A group of documents related by similarities in their content.

clustering The process of automatically defining the **clusters** in a set of documents. Clustering uses Verity **feature extraction** technology.

collection The set of index files and other 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.

collection indexing job A specification of an **indexing** process, including which documents to index and the times when indexing should occur. Also called K2 Spider **job**. Compare **user-defined job (UDJ)**.

collection-level security	A security feature that controls a user's access to a collection as a whole. Collection-level security relies on the user's group membership in the enterprise's native security system. Compare **document-level security**.
concept tree	A hierarchy of key concepts generated by **thematic mapping.**
content organization	See **classification**.
controller	A K2 Spider process that manages **crawlers** and **indexers** for indexing.
corpus	A large collection or set of documents. An enterprise's set of repositories can be considered its corpus.
crawler	A K2 Spider process that gathers document data for indexing.
crawling	The process of seeking out documents in a repository to determine if they are valid candidates for **indexing**. If those documents contain links to other documents, K2 can be configured to follow those links and also crawl the target documents
developer	Create search applications and implement user interfaces that leverage Verity search, classification, and social-network technologies.
`didump`	A command-line tool that displays a list of the words in a collection's **word index**.
distributed search group	A **search group** in which some servers or brokers in the group are from a different domain.
document filter	A Verity software module that can read documents in one or more specific formats (such as PDF, XML, or 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-level security — A security feature that controls which documents can either (1) appear in search results for a particular user, or (2) be viewed by that user. Document-level security relies on the user's access permissions to the individual repositories. Compare **collection-level security**.

document profile — The representation of a document in a **recommendation index**. A document profile is based on the document's **feature vector** and evolves over time from information based on queries that select the document. See also **user profile**.

document table — A table in a **collection** that specifies the location of each indexed document. The document table also contains all metadata (parameters) associated with each document.

domain — A grouping of K2 services consisting of one Master Administration Server and all the K2 services (K2 Ticket Server**s**, K2 Broker**s**, K2 Server**s**, Administration Server**s**, and so on) that it configures.

end users — Use K2 applications to search, browse, and retrieve information.

entity — In the **Recommendation Engine**, a person, document, query, or other object or concept that can be profiled using tensor-space analysis.

feature extraction — The process of automatically discovering the subjects addressed in a document by performing vector analysis on nouns and noun phrases. Feature extraction is performed during indexing.

feature vector — A mathematical structure, constructed during **feature extraction**, that represents the set of subjects addressed in a particular document.

field — A discrete item of document metadata, such as author, title, location, or creation date, in a Verity **collection**.

filter — See **document filter**.

fuzzy search — A search with the ability to retrieve documents containing words with spelling and typographical differences from the search term. Fuzzy search types include **typo search**, **Soundex search**, and **stemmed search**.

gateway	A Verity software module used to retrieve documents from a specific type of repository. K2 includes gateways for local file systems, HTTP, Documentum, ODBC databases, MAPI (MS Exchange), and Lotus Notes.
index	A Verity structure that provides the basis for searching. Examples include collection indexes, knowledge tree**s**, **parametric indexes**, and **recommendation indexes**.
indexer	A K2 Spider process that performs indexing.
indexing	The process of scanning a document to create a word index and to store its metadata (fields and internal zones) into a **collection**.
input context	In the tensor analysis performed by the **Recommendation Engine**, the combination of a query, the document being viewed, and the user's identity.
intellectual capital management	The process of combining human knowledge and experience (both implicit and explicit) with the information and data in an enterprise for the purpose of exploiting greater value.
interest profile	a VQL query, stored in a profile net, that the Profiler Service compares documents to for the purpose of document classification or message routing.
Java Server Page (JSP)	A file containing Java code mixed with HTML and JavaScript. Used to generate Web pages. Compare **Active Server Page (ASP)**.
job	See **collection indexing job**, **user-defined job**.
K2 Broker	A K2 service that receives client search requests and distributes them to available K2 Server**s**.
K2 Dashboard	A browser-based user interface that enables **administrators** to view and change configuration settings for K2 services from a single computer, even when the K2 services reside on many different computers.
K2 domain	A K2 system consisting of one Master Administration Server and all the K2 services configured by that Master Administration Server. Note that a K2 domain is unrelated to a Windows NT domain.

K2 search group — A set of K2 Brokers and K2 Servers containing one top-level K2 Broker and all the other K2 Brokers and K2 Servers attached below it, possibly including ones in different **K2 domains**. A search request handled by the top-level K2 Broker can be passed to any of the other K2 Brokers and K2 Servers in the search group, including those from other K2 domains.

K2 Server — A K2 service that receives search, viewing, profiling, and recommendation requests and performs searches of collections, knowledge trees, parametric indexes, RE Doc Indexes, and RE User Indexes.

K2 services — The executable processes in a K2 system, such as a K2 Broker, a K2 Server, or a K2 Ticket Server.

K2 Spider — A tool to perform **spidering**. K2 Spider executes through the K2 Server, and thus can perform distributed spidering. Compare Verity Spider.

K2 Spider Client — The executable and command-line tool used to interface with **K2 Spider Servers** to create and manage indexing jobs.

K2 system — A generic term meaning a K2 installation. It may be either a K2 domain or a **K2 search group**.

K2 Ticket Server — A K2 service that is used to implement secure access to collections, search results and documents. The K2 Ticket Server stores information in memory for users who have been authenticated.

knowledge tree — A structure for organizing documents for navigation to subjects of interest. A knowledge tree consists of a **taxonomy** plus category definitions plus documents attached to those categories.

knowledge worker — A librarian or domain expert that makes decisions about what information sources to make available to users of a K2 installation. They index collections, and create and populate taxonomies.

language identification filter — A **document filter** (`flt_lang`) used by the multilanguage locale to assign a language to a document before indexing.

locale — 1. A geographic or political region that shares the same language and customs. 2. See **Verity locale**.

Logistic Regression Classifier (LRC) A Verity tool that creates a **category** definition from a set of positive and negative exemplary documents.

Master Administration Server An Administration Server that is the central hub for K2 configuration information. A K2 domain must have one and only one Master Administration Server.

metadata Data that describes other data. For example, `Author` and `Size` could be metadata for a Microsoft Word document. Fields in Verity collections contain document metadata that can be searched for.

mirroring The creation of multiple duplicate collections attached to different K2 Servers. K2 Spider can be configured to create mirrored collections.

`mkprf` A command-line tool for building and maintaining **profile nets**.

`mksyd` A command-line tool used to build a thesaurus from a thesaurus control file.

`mktopics` A command-line tool for building and updating **topic sets**.

`mkvdk` An all-purpose command-line **collection** maintenance tool.

multilanguage locale A **Verity locale** (`uni`) that supports multiple languages simultaneously. See also **single-language locale**.

no results filtering A setting for **document-level security** in which all found documents are displayed in results lists, regardless of user access rights. Compare **results-list filtering**.

noun phrase A group of words (for example, *due process* or *court of law*) that functions as a noun. Part-of-speech processing during **indexing** can lead to the automatic extraction of noun phrases, which can be used in document **feature extraction**.

OTL file See **topic outline file (OTL)**.

outline file 1. An XML file that specifies the structure of a **parametric index**. 2. See **topic outline file (OTL)**.

parallel querying The ability to simultaneously search multiple collections. K2 Server and K2 Broker support parallel querying.

parametric index An index that supports **parametric selection**.

parametric selection	The ability to search for documents based on the values of one or more document parameters, combined with full-text search on document content.
passage-based summary (PBS)	An automatically generated document summary that consists of text passages in which the search term appears, typically highlighted.
profile	See **document profile**, **interest profile**, **user profile**.
profile net	A set of stored **interest profiles** (queries) against which the **Profiler Service** evaluates documents.
Profiler Service	A K2 service that evaluates an incoming stream of documents against the interest profiles in a profile net. Developers can use Profile Services in applications such as message routing and document classification.
proximity search	A type of search that returns documents in which the specified terms are close to each other (for example, in the same sentence or separated by no more than a specified number of words).
`rcadmin`	A command-line tool used to administer K2. It has similar functionality to the **K2 Dashboard**.
`rck2`	A command-line tool used to connect to K2 Server**s** for searching **collections** and other Verity **indexes**.
`rcvdk`	A command-line tool used for searching **collections** and displaying documents.
RE Doc Index	A data file that contains the profiles of the documents in a collection.
RE User Index	A data file that contains the profiles of a set of users on a host.
Recommendation Engine	The K2 component that provides **recommendations**.
recommendation index	A data file that contains entity profiles used by the Recommendation Engine**.**
relational taxonomies	A Verity feature in more than one **taxonomy** is applied to a set of information. Relational taxonomies allows users to simultaneously navigate through the different taxonomies.

repository A group of documents that are all stored in the same location and accessed through the same protocol, such as a file system. Repositories can include relational databases or proprietary storage systems such as Microsoft Exchange folders or Lotus Notes databases.

results-list filtering A setting for **document-level security** in which results lists show only those documents that a user can retrieve. Compare **no results filtering**.

score A numerical value indicating the degree of match between a document and a query. Scores, usually expressed to the end user as a decimal number between 0 and 1, are calculated during Verity search or Profiler operations. Scores are based on numerous factors, including the number of times search/query words appear in the document, their location in the document, and their proximity.

search group A grouping of K2 services consisting of one top-level K2 Broker plus all the other K2 Brokers and K2 Servers attached to it.

search worker A software module in **federated search** that connects to and retrieves information from a particular kind of information source.

session-based profile Temporary and dynamic **user profiles** that can be used to track relevant searches and purchases so that similar products can also be recommended.

single-language locale A **Verity locale** that supports only one language. Most locales are single-language. Compare **multilanguage locale**.

social network A model of the explicit and implicit relationships between the people in an organization and the documents they create, modify, access, search, and organize.

Soundex search A kind 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.

spidering The process of **crawling** and **indexing** the contents of a repository.

stemmed search A kind of search that locates all words that share the same word stem. For example, a stemmed search for the term *dance* would find all occurrences of *dance*, but also all occurrences of *dances* and *dancer*.

stop-word list A file containing search terms that should be ignored. Verity supports several types of stop-word lists, some used at indexing time and others at search time.

style file A file used to configure the indexes and fields in a **collection**.

StyleSet Editor The Verity application that enables administrators to create and modify style files.

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**.

taxonomy The hierarchical organization of categories. A taxonomy defines a structure for accessing data.

tensor A multidimensional mathematical structure used by the **Recommendation Engine** to construct a weighted representation of the significant subjects and actions of a document or user.

tensor space A multidimensional space to hold tensors used by the **Recommendation Engine**.

thematic mapping A process that automatically discovers the key concepts in a collection of documents and maps the hierarchical relationships between them.

thesaurus A dictionary of synonyms. Each **Verity Locale** supports use of a thesaurus for searching. In a synonym search, all occurrences of the search term and any of its synonyms are returned.

ticket A temporary access pass granted by the K2 Ticket Serverto a user for as long as the user is logged in.

topic A stored query expression written in the Verity Query Language (VQL). Topics are used to model concepts of interest in a classification task, or to enable users to quickly find information without having to compose sophisticated queries. See also **topic set**.

topic outline file (OTL)	A text file that defines the structure of a **topic set**. Topic outline files have a file extension of `.otl`.
topic set	A grouping of **topics** that have been compiled for use by a Verity application. For classification tasks, a topic set contains one or more topics used to classify documents in a collection.
transaction	A modification of one or more entities in a **recommendation index**. For example, a transaction may make a document more relevant to a particular query due to user input.
typo search	A kind 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 double-byte character sets. The Unicode standard encodes the characters for all major modern languages in one character set. There are various implementations of portions of the Unicode standard. The implementation used by the Verity **multilanguage locale** is UTF-8.
universal filter	A document filter that determines the file type of the incoming document and then invokes a suitable helper filter for extracting the available text and metadata.
user defined job (UDJ)	A specification, created by the administrator, of a command-line tool to be executed plus its associated arguments. Jobs can be scheduled and chained. Compare **collection indexing job**.
user profile	The representation of a user in a **recommendation index**. A user profile is created over time from information such as documents authored by the user, interests submitted, queries asked, and documents rated or viewed. See also **document profile**.
VDK	1. Verity Developer's Kit, the API that enables developers to build Verity functionality into their products. 2. The programming core on which most Verity applications are built.
Verity Intelligent Classifier	An application for creating, viewing, editing, and testing **topics** and **taxonomies**.

Verity locale — A software module that allows Verity applications to operate on documents in a specific language or set of languages. A locale provides one or more capabilities that may include tokenization, stemming, part-of-speech recognition, and thesaurus use. See also **single-language locale**, **multilanguage locale**.

Verity Query Language (VQL) — Verity's standard language for creating search queries.

`vspider` — A command-line tool that provides document **indexing** capabilities. See also K2 Spider.

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 all documents containing occurrences of words that start with *abc*.

word index — A **collection** index that lists all words that appear in the collection's documents and the location of every instance of each word.

worker — See **search worker**.

zone — A named region of a document. Examples are HTML tags such as `TITLE`, `BODY`, and `H1`, and email fields such as `TO`, `FROM`, and `SUBJECT`. Zones can be made searchable in collections and can also be saved as collection fields.

Index

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

Z