

# Verity® K2 Toolkit Search System Administration Guide V2.2

July 20, 2000  
Verity, Incorporated  
894 Ross Drive  
Sunnyvale, California 94089  
(408) 541-1500  
Part Number DM0451

Copyright 2000 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<sup>®</sup>, KeyView, the Verity search button, and the running man logos are trademarks of Verity, Inc. in the USA and numerous other countries.

Sun, Sun Microsystems, the Sun logo, Sun Workstation, Sun Operating Environment, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a trademark of UNIX Systems Laboratories.

Macintosh is a registered trademark of Apple Computer, Inc.

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.

InfoSoft is a registered trademark of InfoSoft International, Inc.

Deluxe English US Electronic Thesaurus © 1994 by InfoSoft International, Inc. Adapted from the Oxford Thesaurus © 1991 by Oxford University Press and from Roget's II: The New World Thesaurus © 1980 by InfoSoft International, Inc. All rights reserved. Reproduction or disassembly of embodied programs and databases prohibited.

LinguistX<sup>™</sup> from Inxight Software, Inc., a Xerox New Enterprise Company, © 1996-1997. Xerox<sup>®</sup>, Inxight<sup>™</sup> and LinguistX<sup>™</sup> are trademarks of Xerox Corporation and Inxight Software, Inc. LinguistX<sup>™</sup> contains patented technology of Xerox Corporation. All rights reserved.

*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.

Part number DM0451

# Table of Contents

## Preface

Content Summary.....	viii
Conventions Used.....	x

## Chapter 1 K2 Search System

K2 Search System Overview.....	1-2
Advanced Search and Retrieval.....	1-3
Verity Document Filters.....	1-4
Verity Locales.....	1-4
Scalability.....	1-5
Simple Design with Single K2 Server.....	1-5
Advanced Design with K2 Broker and Multiple K2 Servers.....	1-5
Load Balancing.....	1-7
Intelligent Routing of Search Requests.....	1-7
Parallel Data Architecture.....	1-8
Flexible System Design.....	1-10
Redundancy.....	1-10
Ping Communications.....	1-10
Multiple Language Search.....	1-10
Verity Topics.....	1-11
Operation and Administration.....	1-13
K2 Broker and K2 Server Administration.....	1-13
Self-Monitoring Features.....	1-13
Server Status.....	1-13
Collection Status.....	1-14
Remote Administration.....	1-14

## Chapter 2 K2 System Configuration

K2 Configuration Overview.....	2-2
Document and Collection Management.....	2-5
K2 Collection State.....	2-5
K2 Document Key.....	2-5
Using Collections with K2 Server.....	2-6
Load Balancing.....	2-6
Distributing Collections.....	2-6

## Chapter 3 Setting Up K2 Servers and K2 Brokers

Setting Up a K2 Server .....	3-2
k2server Command-line Tool .....	3-2
k2server.ini Configuration File .....	3-3
Server Section .....	3-3
Server Administration Keywords .....	3-3
Search Thread Keywords .....	3-5
Collection Sections.....	3-6
Setting Up a K2 Broker .....	3-9
k2broker Command-line Tool.....	3-9
k2broker.ini Configuration File .....	3-11
Broker Section .....	3-11
Node Section .....	3-13
Sorting Results at the Collection Level .....	3-14
Result Caching.....	3-15
Using rck2 as a Search Client .....	3-16
rck2 Syntax.....	3-16
rck2 Command Options.....	3-16

## Chapter 4 Collections and Search Performance

How Collections Work.....	4-2
Collection Partitions .....	4-2
Attributes and Fields .....	4-3
The Word Index .....	4-4
Using mkvdk .....	4-7
Using the merge Utility.....	4-9
Merging Collections .....	4-9
Splitting Collections .....	4-9
Using the Incremental Squeeze Feature .....	4-11

## Chapter 5 Verity KeyView Filters

Key Features .....	5-2
Supported Formats .....	5-3
KeyView Filters—Limitations.....	5-5
New Features and Enhancements .....	5-6
Headers and Footers .....	5-6

## Chapter 6 XML Support

Requirements for Data Files .....	6-2
Implementation Summary.....	6-3
XML Filter.....	6-3
Style Files .....	6-3
Style File Configuration .....	6-4

style.uni File.....	6-4
style.xml File.....	6-4
style.ufl File .....	6-6
style.dft File .....	6-6
Indexing XML Documents.....	6-7
Indexing using mkvdk .....	6-7
Searching using rcvdk.....	6-7

## Chapter 7 Verity Locales

Verity Locales and their Components .....	7-2
Predefined Locales.....	7-2
Custom Locales .....	7-2
Tokenization for Locales other than English.....	7-2
Locales from Verity Partners.....	7-2
About Verity Locales .....	7-3
Verity Locales Using LinguistX.....	7-4
Upgrading from IntelliScope to LinguistX Locales .....	7-4
Using the THESAURUS Operator.....	7-5
Installing Predefined Locales .....	7-6
Localized Query Language.....	7-8
Using English Query Language for Locales other than English.....	7-10

## Appendix A Reference for mkvdk

Overview.....	A-2
Default Behavior .....	A-2
Document Path Names in Collections .....	A-2
Basic Syntax .....	A-3
Basic Operations.....	A-4
Optimization, Modes, and Service Options .....	A-7
Advanced Features .....	A-11

## Appendix B Using Verity Topics

Using mktopics to Create Virtual Collections .....	B-2
mktopics Syntax Reference.....	B-3
Topic Set Limits.....	B-5

## Appendix C Date Formats

Export Date Format .....	8
Import Date Format .....	10

## Table of Contents

# Preface

The *K2 Toolkit Search System Administration Guide* is designed to help you configure, maintain, and use the K2 Toolkit to build a K2 advanced search system for your enterprise. To make the best use of this document, you should be familiar with Verity collection building and search technology. The following Verity documentation provides additional background information:

- *Verity Collection Building Guide*
- *Verity Query Language Reference Guide*

In addition, Verity maintains a Web site providing current information about Verity products and technology, which you can reach with the following URL:

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

# Content Summary

The following summary describes the contents of each chapter in this manual.

## **Chapter 1: K2 Search System**

This chapter introduces the features of the K2 Toolkit and how to use the toolkit to develop an enterprise-scale K2 search system. The K2 Toolkit combines enterprise-level performance and scalability with Verity's market leading knowledge retrieval features. Using the K2 Toolkit, you can design a scalable solution that meets your current needs and can grow easily in the future.

## **Chapter 2: K2 System Configuration**

This chapter describes K2 search system configuration and configuration options. Several configuration parameters can be set to control the operation of the K2 server and how it accesses Verity collections (or document indexes).

## **Chapter 3: Setting Up K2 Servers and K2 Brokers**

This chapter describes how to set up, configure, and start K2 Servers and K2 Brokers. A k2server.ini file is used to configure the K2 Server. A k2broker.ini file is used to configure the K2 Broker. These configuration files and startup tools are covered.

## **Chapter 4: Collections and Search Performance**

This chapter provides an overview to Verity collections and the indexing process. Additionally, information about optimizing collection configuration for the best possible search performance is given.

## **Chapter 5: Verity KeyView Filters**

This chapter describes the features of the KeyView document filters included with K2 Toolkit V2.0. These filter support indexing and viewing documents stored in numerous popular desktop publishing, graphics presentation formats.

## **Chapter 6: XML Support**

This chapter describes support for XML and how to use the XML filter to index documents.

## **Chapter 7: Verity Locales**

This chapter describes the numerous predefined Verity locales which you can use to localize your K2 search system.



### **Appendix A: Reference for mkvdk**

This appendix describes the **mkvdk** utility, a standard Verity indexing tool. Using the command-line **mkvdk** tool, you can create Verity collections which are required components of your K2 search system. Command-line syntax and examples are included.

### **Appendix B: Using Verity Topics**

This appendix discusses how to define Verity topics in a topic outline file, and then build a topic set for those topics using the **mktopics** utility. This information is covered in detail in the *Verity Collection Building Guide*.

### **Appendix C: Date Formats**

This appendix describes input and output date formats used in K2 Toolkit.

# Conventions Used

<b>Convention</b>	<b>Usage</b>
<code>Courier type</code>	Used to describe API constructs in structure member descriptions and code examples
<i>Courier oblique type</i>	Used for user-replaceable constructs
<b>Courier bold</b>	Used to denote API data type names in structure member descriptions and for command-line application names.
Palatino	Used in narrative text
<i>italics</i>	Used at the first introduction of a new term, and for book titles

# K2 Search System

This chapter introduces the features of the K2 Toolkit and how to use the toolkit to develop a robust K2 search system. These topics are covered:

- K2 Search System Overview
- Advanced Search and Retrieval
- Scalability
- Load Balancing
- Flexible System Design
- Operation and Administration

# K2 Search System Overview

The Verity K2 Toolkit, designed for application developers, combines the market leading knowledge retrieval features of the Verity Developer's Kit with scalability and enterprise-level performance. Using the K2 Toolkit, organizations can build fault tolerant applications that allow thousands of users to search and navigate hundreds of millions of unstructured documents in electronic form, with nearly instantaneous results.

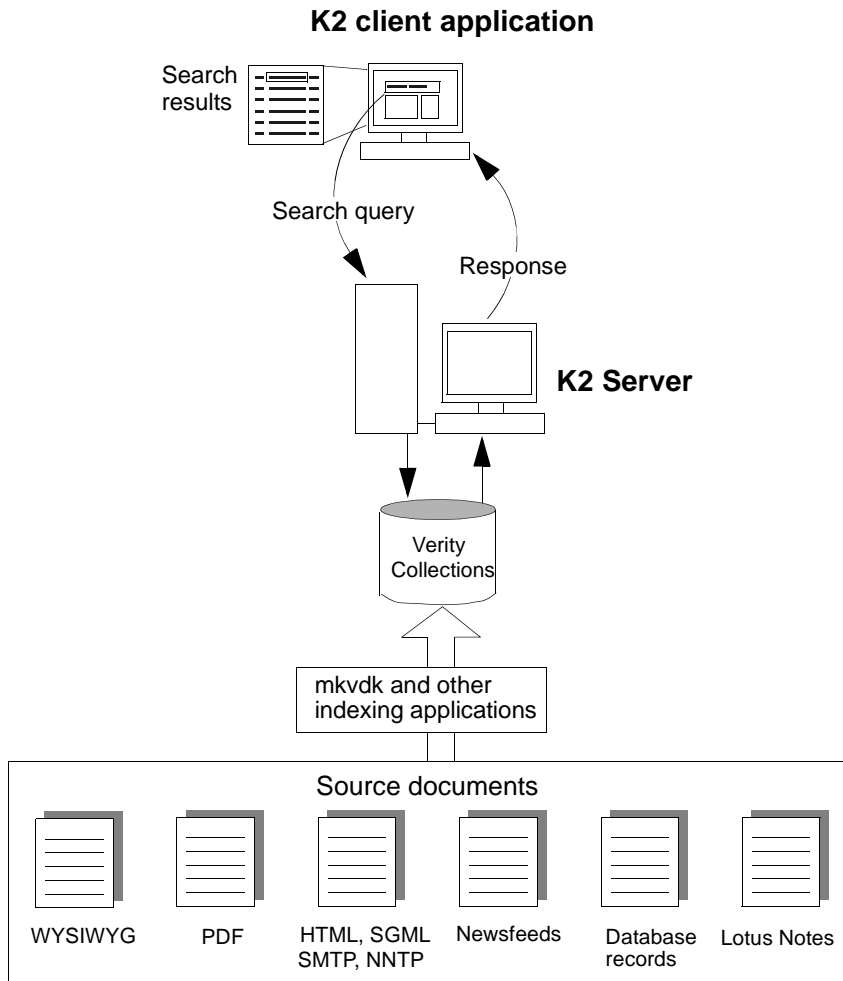
The K2 Server is a multi-threaded C application built around the Verity search engine, providing access to Verity collections and tracking any changes made by indexing applications.

The K2 search system is designed to take advantage of the latest advances in hardware and software technology and provides the following features:

- Multi-threaded architecture
- Support for Verity knowledge retrieval features, including topics
- Continuous operation support
- Incremental squeeze
- Highly scalable

# Advanced Search and Retrieval

The K2 Toolkit offers a high-performance search engine designed to process searches quickly in a high performance, distributed system. As shown in the illustration below, a K2 search system built using the K2 Toolkit has a client/server model. K2 client applications, built using the K2 Toolkit API, provide users access to document indexes stored in Verity collections.



Verity collections provide full-text indexes for information resources on an enterprise-wide basis, including Internet formats, newsfeeds, PDF, SQL databases, Lotus Notes, and WYSIWYG documents, such as MS Word, Adobe FrameMaker, and MS Excel.

## **Verity Document Filters**

Verity KeyView Filters V6.5 are packaged with Verity K2 Toolkit V2.2. The KeyView filters support indexing and viewing documents stored in many popular desktop publishing, word processing, and presentation formats. For information about the KeyView filters, see Chapter 5, “Verity KeyView Filters.”

This release includes a new Verity-supplied filter for XML. For complete information, refer to Chapter 6, “XML Support.”

Verity filters for ASCII, HTML, and PDF formats are included with K2 Toolkit as well. For complete information about these filters and their default configuration, refer to the *Verity Collection Building Guide*.

## **Verity Locales**

Verity provides many predefined locales that you can use right away. Locales have several components including a linguistics package (stemmer, tokenizer, natural language processing capabilities). The linguistics package is based on technology from Inxight™ LinguistX™.

Verity packages support for each language (or region) as a separate locale. A Verity locale includes all the components specific to the language, such as a character set mapping, date formats, a message database, a linguistics package (stemmer, tokenizer, natural language processing capabilities).

For information about Verity locales, refer to Chapter 7, “Verity Locales.”

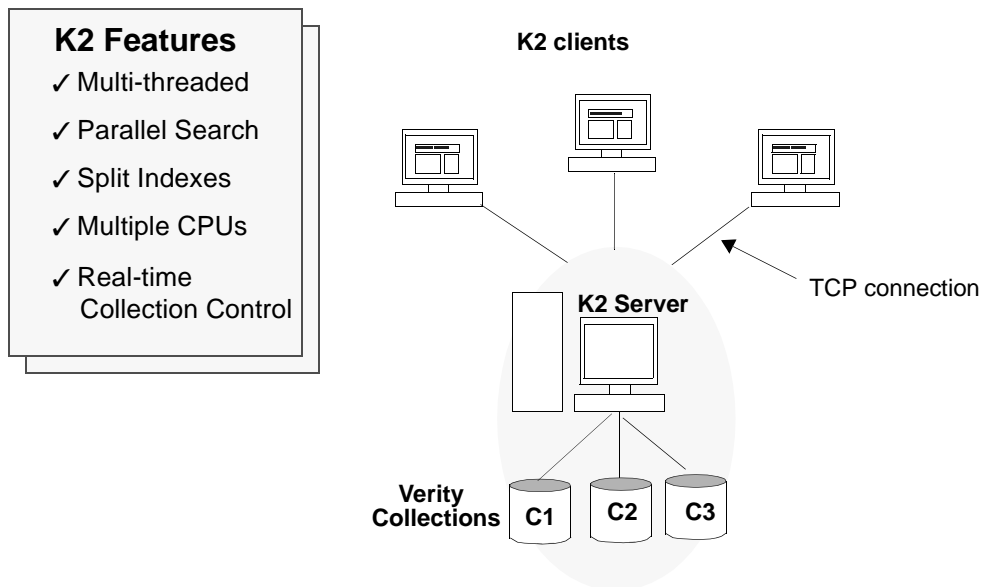
# Scalability

As the number of users grows and the quantity of data increases, K2 systems can keep pace by adding new servers and/or processors. A K2 search system designed today can be expanded gracefully in the future to accommodate more users, documents, or queries with no degradation in performance.

Scalability is supported by the K2 multi-tier parallel computing architecture which takes advantage of advanced symmetric multiprocessing (SMP) and exploits the multi-threading and concurrency features of today's leading SMP hardware and software platforms. The multi-tier architecture includes configurable components, including the K2 Server, K2 Broker, and K2 client application (built using the K2 Toolkit API).

## Simple Design with Single K2 Server

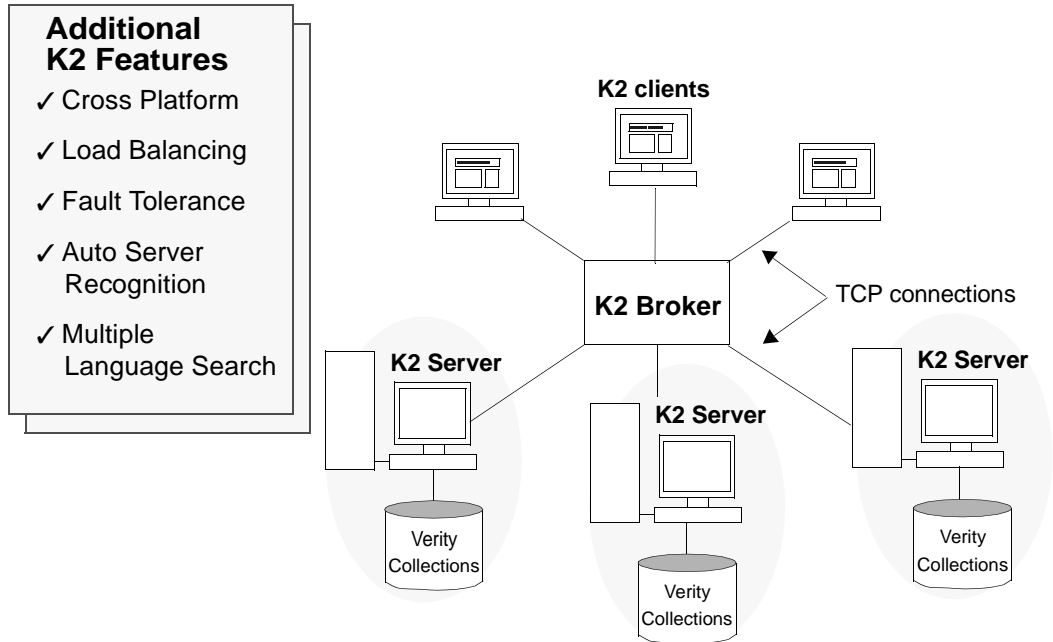
In a simple K2 system, client requests are passed to a single, multi-threaded K2 Server.



## Advanced Design with K2 Broker and Multiple K2 Servers

In an advanced K2 system, one or more K2 brokers consolidate client requests and pass them on to multiple servers, which use the K2 multi-threaded, multiprocessing architecture to concurrently search over numerous collections, providing efficient access

to millions of documents. As shown in the following illustration, the K2 Broker acts as an intermediary between the K2 Server and K2 clients, allowing a client request to be distributed to the next available server.



TCP connections are used to connect the K2 Broker with clients and servers, allowing the K2 Broker to run on the same machine with a client or server or to run on a dedicated system. The K2 Broker supports the growth and extension of an information retrieval system in a virtually unlimited manner and provides the redundancy required to avoid a single point of failure.



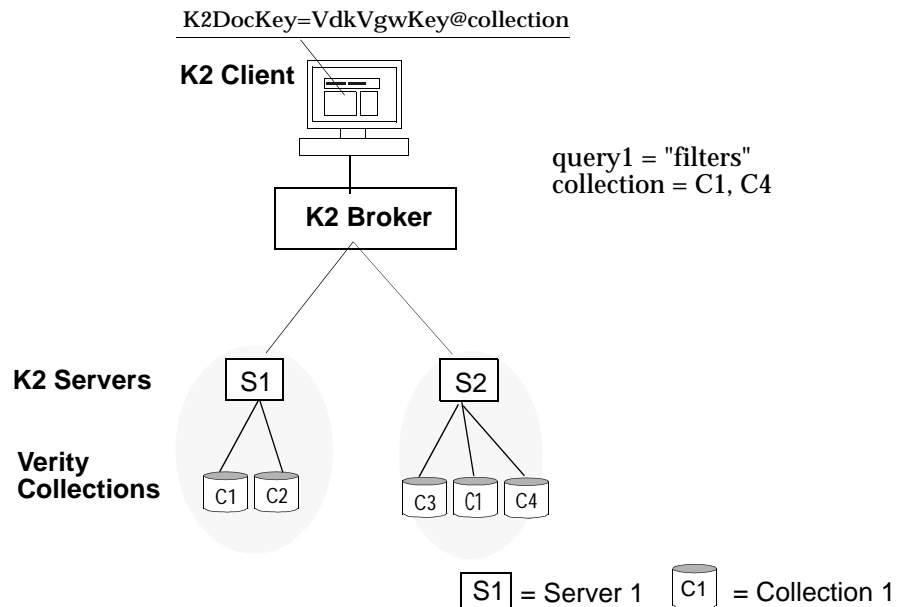
# Load Balancing

The K2 search system implements concurrent load balancing to spread the processing load across multiple servers. These K2 Toolkit features support efficient load balancing to ensure quick response times and system stability:

- Intelligent Routing of Search Requests
- Parallel Data Architecture

## Intelligent Routing of Search Requests

Depending on the server availability, the K2 Broker determines whether to distribute a single search request to one or more servers. If a search request consists of a query with two or more target collections, the K2 Broker can send the search to one or more servers. Given the K2 search system configuration shown below, if the search request is query1 with the target collections 1 and 4, the K2Broker can divide the search request into two parts: query 1 to Server 1 (S1) for Collection 1 (C1) and query1 to Server 2 (S2) for Collection 4 (C4).



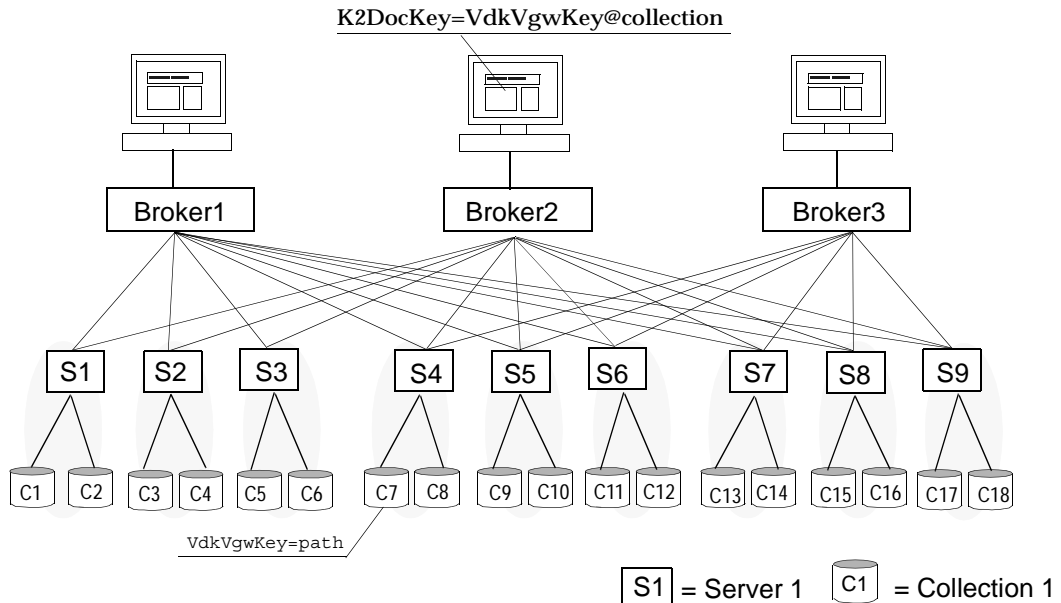
Given the search request as query1 with the target collections C1 and C4, the K2 Broker can process the request in two different ways:

- Send the whole search request to Server 2 (S2)
- Split the search request and send query1 for Collection 1 (C1) and query1 for Collection 4 (C4) at Server 2 (S2)

The K2 Broker always distributes a search request in the most efficient way based on its monitoring of server load. The number of distribution options is dependent upon how you distribute your collections across servers.

## Parallel Data Architecture

K2 servers and brokers can be used in a multi-tiered, brokered system, which can allow fast, distributed search response for a practically unlimited number of search requests from end users. The figure below shows a K2 search system that has three brokers, nine servers, and 18 collections.



This multi-tiered approach allows search requests to be distributed to as many servers as necessary to maintain optimum user response time. It also allows you to create redundant processing paths to ensure uninterrupted service, even in the event of hardware or software failure of a particular server or broker.

In the illustrated system, every collection is available to any client connected to Broker 1 and 2. The system configuration allows clients connected to Broker 3 to search collections managed by six of the nine servers. Each document in the system is identified by a unique external key, which in the illustration consists of the document path name plus an identifier for the collection, server, and broker used to find the document.

By default, a search request issued by a client in this system can be processed by every server for every collection in the system, allowing fast, global searches. If the collections are built correctly, the results of a search by a client connected to any broker in the system will be the same. To make this possible, the collections must have a consistent structure and should be segmented to optimize search performance.

# Flexible System Design

The K2 development framework provides optimized C language APIs for client development and tools for configuring the search system design and performance. The following system design and performance features can be implemented using tools supplied with the K2 Toolkit:

- Redundancy
- Ping Communications
- Multiple Language Search
- Verity Topics

## Redundancy

The K2 Toolkit's parallel data architecture enables the system administrator to duplicate collections across K2 servers. Collection redundancy offers these benefits:

- Enables the K2 Broker more options for optimizing search performance
- Supports continuous search over data, even when servers are coming on-line and off-line

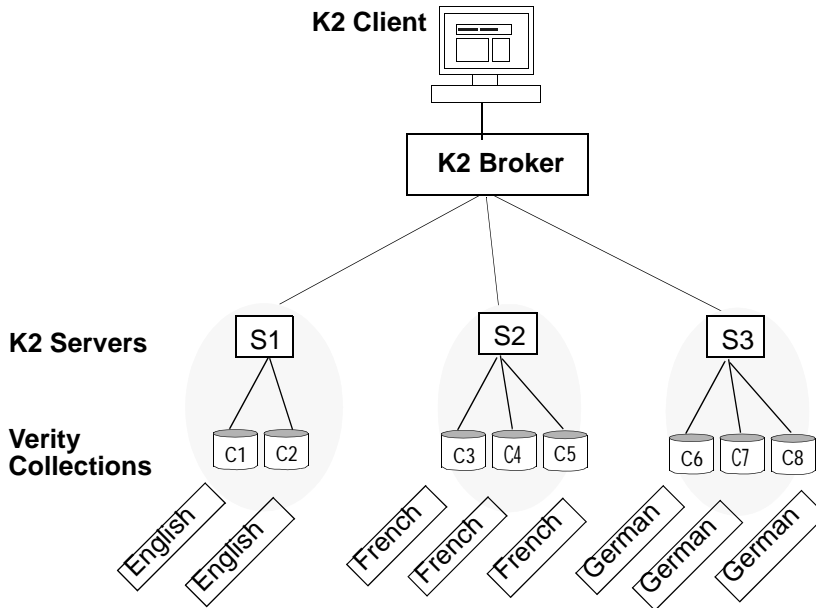
## Ping Communications

The new ping functionality synchronizes K2 Broker/K2 Server communication and helps to optimize system performance. The ping functionality is optional, although recommended.

The K2 Broker is always looking for servers. Using ping communications, K2 Server can be configured to ping the K2 Broker on a regular basis. When implemented, the K2 Server pings the K2 Broker until the connection is re-established. If a TPC/IP connection is interrupted, the K2 Server pings the K2 Broker until communication is restored. The K2 Broker expects to receive ping notifications within a specified period of time. If the K2 Broker does not receive ping notifications within the specified period of time, the K2 Broker stops sending search requests to the server until the K2 Broker/K2 Server connection is restored. The ping mechanism ensures that search requests are handled in the most efficient way.

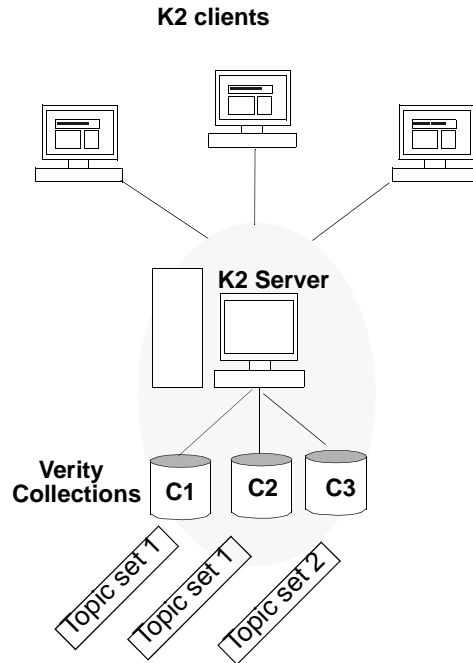
## Multiple Language Search

The K2 Toolkit supports simultaneous multiple language search. Each K2 Server in a K2 search system can be set up to search collections of documents in a particular language, as illustrated below. The S1 server searches English collections, the S2 server searches French collections, and the S3 server searches German collections.



## Verity Topics

A Verity topic is a pre-defined query which can be reused and shared among users. Verity topics can be indexed against a collection to provide performance benefits to end users. A K2 search system can include multiple topic sets. In the illustration below, Topic set 1 was indexed against collections C1 and C2, and Topic set 2 was indexed against collection C3.



For details about how to build collections and topic sets for use in a multi-tiered brokered system, refer to Appendix B, “Using Verity Topics.”

# Operation and Administration

The K2 search system supports fault tolerant 24x7 operations. System stability remains even if servers are unexpectedly brought off line.

Before you startup a K2 search system, you need to set up each K2 Broker and K2 Server to be used in the system, built your K2 client application using the K2 Toolkit APIs, and index your documents into collections. The K2 Toolkit includes a Verity command-line indexing tool called **mkvdk**. For a complete description of **mkvdk** with syntax and examples, see Appendix A, “Reference for **mkvdk**.”

Basics about the K2 search system administration and operation are covered below:

- K2 Broker and K2 Server Administration
- Self-Monitoring Features
- Remote Administration

## K2 Broker and K2 Server Administration

For each K2 Server and K2 Broker, you set up a configuration file which identifies the server or broker to the system. The configuration file for the K2 Server is called `k2server.ini`, and for the K2 Broker it is called `k2broker.ini`. To start each K2 Server, you run a tool called **k2server**. To start each K2 Broker, you run a tool called **k2broker**. The **k2server** and **k2broker** tools are command-line tools which take a port number and configuration file as input.

For complete information about the configuration files (`k2server.ini`, `k2broker.ini`) and the command-line tools (**k2server**, **k2broker**), refer to Chapter 3, “Setting Up K2 Servers and K2 Brokers.”

The administrator can add and remove K2 brokers and servers from a K2 search system at any time, without affecting system performance.

## Self-Monitoring Features

At startup, the K2 Broker looks for all available servers and assumes all servers are alive. As search requests are forwarded to the broker from the clients, the broker determines which server can return results most efficiently before it forwards the search request to a server.

### Server Status

The K2 Broker can recognize when individual servers are disconnected and reconnected.

## Collection Status

During operation, the K2 Broker maintains status information about each collection. The collection status can be:

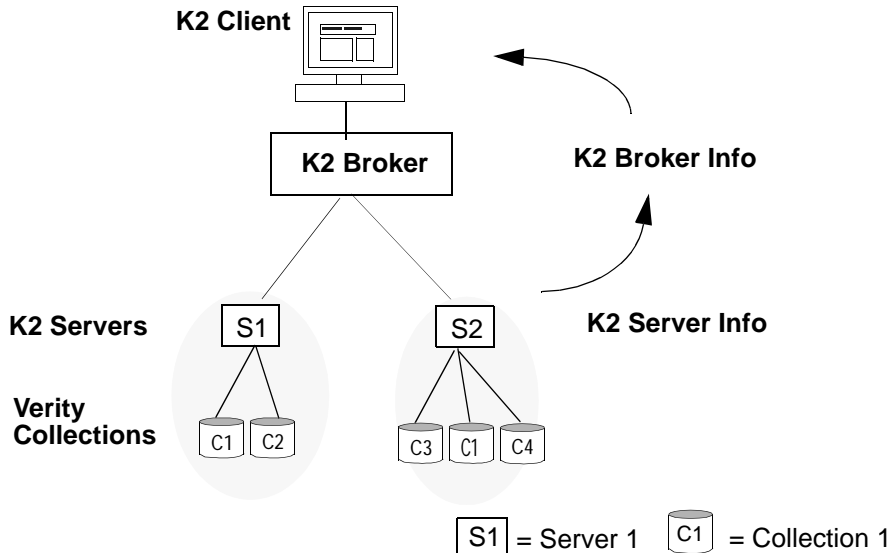
- **On-line**—The collection is on-line and available for searching.
- **Off-line**—The collection is off-line and is not available for searching. When a collection is set off-line, any queries currently running complete using these resources; subsequent queries do not see the resource.
- **Hidden State**—The collection is in a hidden state. Hidden collections can be added to the system and tested, but not searched by end users.

The collection status is set for each K2 Server in the server configuration file, called `k2server.ini`.

The K2 search system automatically makes real-time changes to a collection's status during operation. For example, if the network connection between the K2 Broker and a K2 Server goes down, the K2 Broker changes the collection status to off-line until the connection is restored. Using the K2 client API, a client application can query the status of a collection in the system at any time.

## Remote Administration

The K2 search system includes remote administration features. During operation, servers and collections can be enabled and disabled for search remotely. Also, the entire system configuration can be obtained remotely.





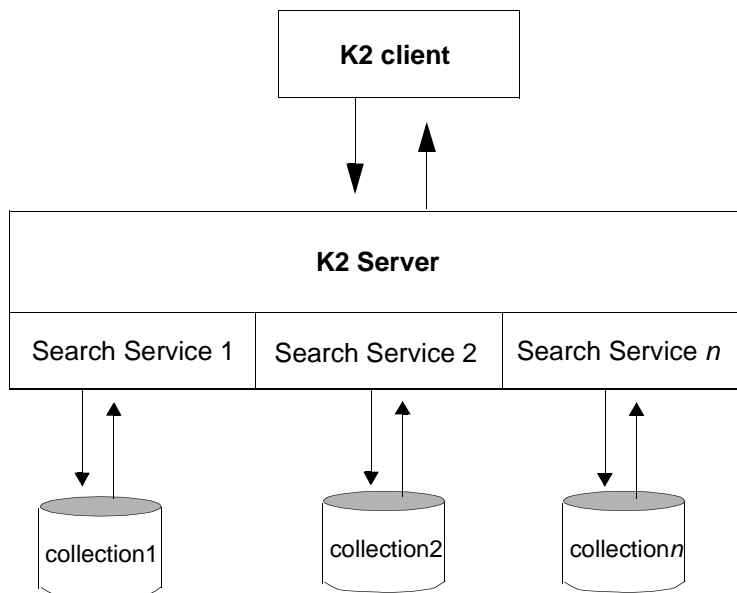
## **K2 System Configuration**

This chapter describes K2 search system configuration and configuration options. These topics are covered:

- K2 Configuration Overview
- Document and Collection Management
- Using Collections with K2 Server

# K2 Configuration Overview

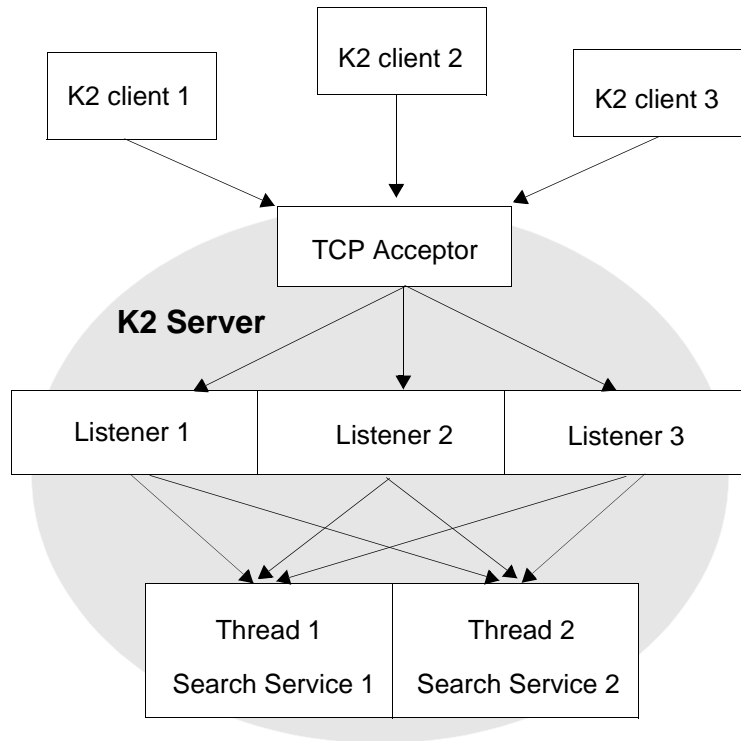
As shown in the following illustration, the K2 Server dedicates one search service for each collection.



The configuration file parameters let the server administrator control all aspects of the K2 Server's configuration, including the path name for each collection (or collection map file). The K2 Server architecture supports high-performance searches of very large document sets. Parallel search services are created by defining multiple collections for a source document set and listing each collection in the K2 Server configuration file.

All the collections appear as a single collection to users and user applications, but each search request can be processed by multiple search services. This design takes advantage of SMP hardware and multithreaded operating systems, allowing greater system resources to be dedicated to a search request. Search response time can be improved by simply adding more processors and system memory.

As shown in the following illustration, the K2 Server provides a TCP acceptor that receives client connection requests and connects each client to a separate listener service for each concurrent connection. The Listener receives client requests and passes the requests to the next available search service.



The number of listeners determines how many concurrent client connections the server can support. Each search service can process one search request for each thread assigned to it. The number of threads multiplied by the number of search services determines the number of concurrent searches that the K2 server can perform. For example in the system illustrated, the number of threads assigned to the two search services is one and the number of listeners is three. You need to be sure there are enough listeners for the clients. If all three clients were to submit requests at about the same time, then the third client would be rejected by the K2 server.

Because one listener is required for each concurrent search request, the number of listeners should always be equal or greater than the number of threads.

*Note: an extra Listener thread must be assigned if you wish to create a concurrent K2Watch client connection for monitoring or controlling the server.*

To optimize search performance on a dedicated search server, you should allocate enough listeners to support the largest number of concurrent client connections and enough threads to optimize CPU utilization. If you have a much larger number of registered users than the number of listeners you assign to support concurrent users, you should include logic in your K2 client application that causes the client to wait for the

next available listener, and to time out a connection when it becomes inactive. The default behavior of the K2 client connection (`K2ConnectNew`) is to remain open because it is inefficient to repeatedly close and open a TCP connection.

# Document and Collection Management

The K2 Toolkit includes robust document and collection management features, as described in the section below.

## K2 Collection State

New to this release, the K2 system maintains information about the state of each collection in the system. A collection can have one of these states:

- Collection off-line
- Collection in a hidden state
- Collection on-line

In the hidden state, collections can be primed and tested, but are not yet available for searching by users. When collections are set off-line, any queries currently running complete using these resources; subsequent queries do not see the resource.

The K2 collection state information helps the K2 search system to run efficiently. The K2 client application can request state information at any time so application users and/or search system administrators can benefit from up-to-date information on server/broker performance.

## K2 Document Key

The K2 document key identifies a Verity document key to the K2 Broker. The K2 document key is a `CSTRING` data type in this format:

*VdkVgwKey@collection*

where *VdkVgwKey* represents the Verity document key assigned to the document during indexing, *collection* is the collection name where the document exists.

K2 document keys are used by the K2 Server, Broker, and Client components. Document keys are managed internally between the K2 Server and Broker. A K2 client application needs to specify a document key when a user wants to view a hit document. To display a document, the application supplies a K2 document key for in the `K2ReadDocNewArgRec` structure that is input to the `K2ReadDocNew` function.

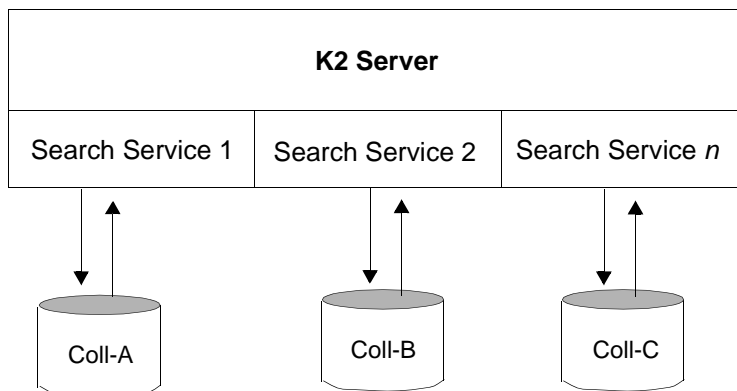
A K2 client can get a number of document read errors, including “Invalid document” and “Invalid collection.”

# Using Collections with K2 Server

You can balance the search load and optimize search performance by allocating threads to different search services, or by distributing and segmenting your collections to separate server machines.

## Load Balancing

In the simplest case, collections are built sequentially over time, using separate sources, as shown in the following diagram:



Each of the collections in this diagram indexes documents from a specific source, such as a newfeed or file system directory. In the system illustrated, global search requests are efficiently distributed to different search services. However, because search requests apply to the entire group of collections accessed by a K2 Server system, the attribute table for each collection should contain a field that identifies the source of a particular document. This allows users to issue search requests for information from a specific source.

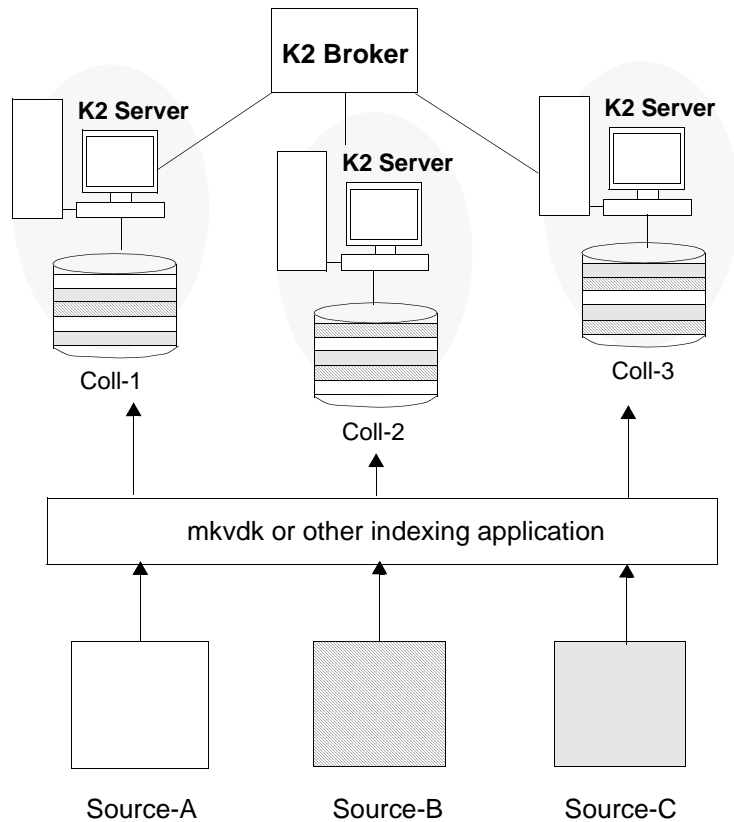
If users issue a large number of search requests restricted to one source, the associated search service may become overloaded, while other search services are under utilized. One way to handle the extra loading is by allocating extra threads to the larger and more popular collections. If requests for a collection can no longer be handled by a single server machine, you can divide the collections among a group of servers.

## Distributing Collections

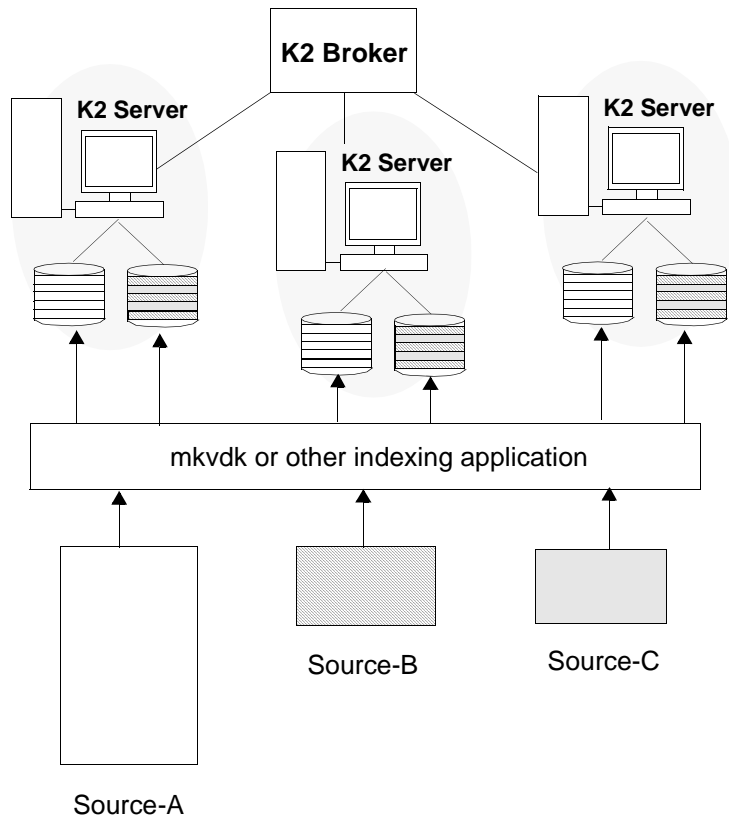
If you have collections of approximately equal size, you can simply distribute the collections to as many server machines as you have available and use one or more instances of the K2 Broker to balance and distribute search requests to the different

machines. Again, global searches will be handled efficiently, but if a disproportionate number of search requests are restricted to a specific collection, the overall load will not be in balance.

To balance the load under these circumstances, you can segment your collections into equal sizes, with each collection containing indexes for documents from different sources. The following illustration shows how this might look in a system with three K2 Server nodes and a single K2 Broker.



With this configuration, search requests for a particular source are distributed to all the available servers and search processes. This configuration will help balance the load over a number of machines. However, you cannot allocate more threads for search requests over a particular source, because the sources are not segregated in separate collections. If you want to allocate more processing resources to handling search requests over a specific source, which is very large and/or very popular, you can segment the indexes for this collection into dedicated collections, as shown in the following illustration:



In this example, the documents in Source-A are far more numerous and popular than those in Source-B and Source-C. Accordingly, **mkvdk** has been used to create three collections indexing documents from Source-A, and three collections indexing documents from both Source-B and Source-C. Additional threads can now be allocated as required to maintain good response time for requests restricted to documents in Source-C.

When creating segmented collections, it is also important to distribute the index entries for documents of different ages. This helps to further balance search requests, which tend to focus on more newer source documents. In the illustration, the collections for Source-A have been “striped” according to date, so that each collection contains an even balance of index entries for older and newer source documents.

To create segmented collections, run **mkvdk** with a separate bulk submit file for each group of source documents. The **mkbulk** utility can be used to create a bulk submit file for a specific group of source documents.



# 3

## Setting Up K2 Servers and K2 Brokers

This chapter describes how to set up, configure, and run K2 Servers and K2 Brokers. These topics are covered:

- Setting Up a K2 Server
- Setting Up a K2 Broker
- Sorting Results at the Collection Level
- Using rck2 as a Search Client

# Setting Up a K2 Server

You start a K2 Server using a command-line tool called `k2server`. The `k2server` tool takes as input the name of a server configuration file called `k2server.ini`. The section below describes how to use the `k2server` tool with the `k2server.ini`.

## k2server Command-line Tool

The K2 Server is started from the command line or from a script in the Unix environment and is integrated as a service within the Windows NT environment. The server is designed to run with a minimum of intervention. Most configuration parameters are set in a configuration file, which can be given a user-assigned name (the default file name is `k2server.ini`).

Command-line arguments include the name of the configuration file, the TCP port for incoming connections and the verbosity level for informational messages. The K2 Server has a warm restart capability, designed to keep the server's well-known TCP port open in case of a crash and to allow changes in the configuration file to be initialized without killing the primary server process.

The K2 Server is started by the using the following command:

```
k2server [<option1> <option2> ...]
```

The options available for this command are summarized in the table below.

Keyword	Permitted values	Function
<code>-port &lt;value&gt;</code>	Positive integer	Identifies the TCP port number for use by the K2 Acceptor. To run the K2 Server as an NT service, use the <code>-ntservice</code> keyword and do not specify a port number using the <code>-port</code> keyword.
<code>-iniFile &lt;filename&gt;</code>	Any valid filename	Identifies the filename to use as the configuration file for this instance of the K2 Server.
<code>-verbose &lt;value&gt;</code>	0=status 1=informational 2=verbose 3=debug	Determines the amount of information contained in the K2 Server system messages.

Keyword	Permitted values	Function
-iniEmit <filename>	Any valid filename	Creates a sample configuration file.
-ntService <value>	1=load as NT service 0=remove as NT service	Used to load or remove the K2 Server as an NT service. When set to 1, the server is loaded as an NT service. When set to 0, the server is removed as an NT service. NOTE: To run the K2 Server as an NT service, do not specify a port number using the -port keyword.

## k2server.ini Configuration File

The K2 Server configuration file called the K2server.ini file is composed of a series of blocks. The first block, [Server], provides keywords that control the behavior of the entire server. Each subsequent block, (in the form [Coll-1], [Coll-2], and so forth) controls each collection and search service configured for the server.

### Server Section

The following table describe the keywords that can be used in the [server] section of the server configuration file. A sample configuration file (k2server.ini) is provided with the K2 Server executable.

### Server Administration Keywords

Keyword	Description
serverAlias=	An arbitrary name used to identify the server.
numThreads=	Default number of search threads to be started in the server process. If too many threads exist, the system can run out of memory; if too few threads exist, then searches will be blocked and forced to wait for a Verity engine thread to become free. The value of numThreads is based on hardware resources and system needs.

## Setting Up K2 Servers and K2 Brokers

### Setting Up a K2 Server

Keyword	Description
<code>maxFiles=</code>	<p>The maximum number of file handles that can be opened by a specific search thread. The default value for <code>maxFiles</code> is dependent on the limits of the OS used. The <code>maxFiles</code> value affects how file handles are shared between the operating system and the search engine. The <code>maxFiles</code> and <code>numThreads</code> values together can be used to tune system performance.</p> <p>These values can be set for a server: [server] <code>numThreads=4</code> <code>maxFiles=100</code></p> <p>The above entries for a K2 Server cause the system to support a maximum of 4 concurrent searches, with 100 file handles allocated for each search thread.</p> <p>The search engine determines default values per operating system. For large or fragmented collections, it is recommended that you explicitly set a value for <code>maxFiles</code>.</p>
<code>numListeners=</code>	<p>Maximum number of clients that can connect to the server at one time. The <code>numListeners</code> value must be equal to or greater than the sum of all <code>numThreads</code> values specified by all K2 Brokers in the K2 search system. The <code>numThreads</code> value is set for a K2 Broker in the <code>k2broker.ini</code> file.</p>
<code>portNo=</code>	<p>TCP port number for client connections. The value of <code>portNo</code> is the same value assigned to <code>portNo</code> in the <code>k2broker.ini</code> file that identifies the broker referring to this server.</p>
<code>broker(n)=</code>	<p>Brokers to ping on startup. Multiple brokers may be specified. For example: <code>broker(1)=machinea:9900</code> <code>broker(2)=machineb:9901</code></p>
<code>maxColSize=</code>	<p>The maximum width of the fields to return to the results list, in bytes. Default is 2048 bytes.</p>

## Search Thread Keywords

Keyword	Description
<code>vdKHome=</code>	Directory containing Verity resources.
<code>vdKSortingFlag=</code>	A flag indicating whether the Verity engine will sort at the collection level. Valid values are: NO   False   0 to not perform sorting at the collection level (default) YES   True   1 to perform sorting at the collection level To implement sorting at the collection level you must set <code>vdKSortingFlag</code> to YES in the <code>k2server.ini</code> file (in the [server] section) and the <code>k2broker.ini</code> file (in the [broker] section). See “Sorting Results at the Collection Level” later in this chapter for details about how collection level sorting is implemented.
<code>sortTruncDocs=</code>	Maximum number of documents to consider when sorting.
<code>accessProfile=</code>	Security Access Profile specified in the form of a query expression. The security access profile represents the access question that a document must pass in order for users to have access to it.
<code>topicSet=</code>	Default path name to a directory for the default topic set, which is an indexed set of topics. The value of <code>topicSet</code> identifies the default topic set to make available to clients at start-up by every search service.
<code>knowledgeBase=</code>	Default path name to a knowledgebase map file, which identifies numerous topic sets (indexed topics). The value of <code>knowledgeBase</code> identifies the topic sets (multiple) to make available to clients at start-up for every search service).
<code>charMap=</code>	A string that names the character set to use for strings that are sent into the server, and are generated by the server. This string must correspond to the name of a <code>.cs</code> file in the root of the <code>common</code> directory that configures a character set and its mappings. For example, if your application should use character set 8859 for all of its interactions with the server, then set this <code>charMap</code> to the string <code>8859</code> . Valid values include, but are not limited to, the character sets supplied by Verity: <code>850</code> (default) for code page 850; <code>8859</code> for code page 8859; <code>mac</code> for Macintosh systems.
<code>locale=</code>	The name of the locale (combination of language, dialect, and character set) to use for all internal Verity engine operations. This name must correspond to a subdirectory in the <code>common</code> directory where the configuration file for the locale is found and where the message database and other locale-specific files are located. Leaving this keyword null means the server will use the default internal locale, which is “english” written in the “850” character set.
<code>resultCacheTimeout=</code>	Timeout in milliseconds for the result cache. Timeout occurs after 60 seconds or when the cache overflows based on <code>resultCacheQuota</code> .

Keyword	Description
resultCacheQuota=	The number of slots per segment for the result cache. The result cache is composed of 16 segments, each of which has a number of slots for caching items in: K2SearchNew, K2SearchRecv, K2DocReadBatch. Timeout occurs after resultCacheQuota value * 16. If resultCacheQuota=10, each of the segments has 10 slots. Note that since a search operation involves a call to K2SearchNew and a call to K2SearchRecv, an additional slot is used. For more information, see "Result Caching."
resultCacheEnabled=	A flag indicating whether the result cache is enabled. Valid values are: Yes   True   1 to enable the result cache No   False   0 to disable the result cache (default). By default, the cache is not enabled.
resultCacheMaxInBytes=	Amount of memory, in bytes, to use for the cache.

## Collection Sections

The K2 Server initializes a separate search service for each collection that you identify in the server configuration file. To add one or more collections to the configuration file, enter a separate block of keywords for each collection in the following format:

```
[Coll-n]
collPath=<pathname>
topicSet=<topicset>
knowledgeBase=<knowledgeBase>
numThreads=<value>
maxFiles=<value>
onLine=<value>
maxColSize=<value>
locale=<language>
charmap=<charmap>
inputDateFormat=<format>
```

Increment the block label for each collection that you configure, starting with `Coll-0`. The following table lists the keywords used to configure each collection and search service.

Keyword	Description
collPath=	The path name identifying the collection home directory.
collAlias=	An arbitrary name used to identify the collection.

Keyword	Description
topicSet=	The path name to a directory for the default topic set, which is an indexed set of topics. The value of <code>topicSet</code> identifies the default topic set to make available to clients at start-up by every search service. If not specified, the value of <code>topicSet</code> from the [server] section is used.
knowledgeBase=	The path name to a knowledgebase map file, which identifies numerous topic sets (indexed topics). The value of <code>knowledgeBase</code> identifies the topic sets (multiple) to make available to clients at start-up for every search service. If not specified, the value of <code>knowledgeBase</code> from the [server] section is used.
numThreads=	The number of concurrent searches for the collection. If not specified, the value of <code>numThreads</code> from the [server] section is used.
maxFiles=	The maximum number of files that can be opened by a specific search thread for a collection. If not specified, the value of <code>maxFiles</code> from the [server] section is used. The <code>maxFiles</code> and <code>numThreads</code> values together can be used to tune system performance. These values can be set for a collection: [Coll-0] <code>numThreads=4</code> <code>maxFiles=100</code> The above entries for collection 0 cause K2 to support a maximum of 4 concurrent searches, with 100 file handles allocated for each search thread.
onLine=	A flag indicating whether the server starts up with the collection on-line. Valid values are: 0 to start the server with the collection off-line; 1 to start the server with the collection in a hidden state; 2 to start the server with the collection on-line (default). In the hidden state, collections can be primed and tested, but are not yet available for searching by users. When collections are set off-line, any queries currently running complete using these resources; subsequent queries do not see the resource.
maxColSize=	The maximum width of the fields to return to the results list, in bytes. If not specified, the value of <code>maxColSize</code> from the [server] section is used.
charMap=	A string that names the character set to use for strings that are sent into the server, and are generated by the server. This string must correspond to the name of a <code>.cs</code> file in the root of the <code>common</code> directory that configures a character set and its mappings. If not specified, the value of <code>charMap</code> from the [server] section is used.  For example, if your application should use character set 8859 for all of its interactions with the server, then set this <code>charMap</code> to the string <code>8859</code> . Valid values include, but are not limited to, the character sets supplied by Verity: <code>850</code> (default) for code page 850; <code>8859</code> for code page 8859; <code>mac</code> for Macintosh systems.

<b>Keyword</b>	<b>Description</b>
<code>locale=</code>	The name of the locale (combination of language, dialect, and character set) to use for all internal Verity engine operations. This name must correspond to a subdirectory in the common directory where the configuration file for the locale is found and where the message database and other locale-specific files are located. If not specified, the value of <code>locale</code> from the <code>[server]</code> section is used.
<code>inputDateFormat=</code>	The input date format to be used. If there is no specified value for <code>inputDateFormat</code> , the default is MDY (Month-Day-Year), a numeric format. See Appendix C for more information.

There can be n-collection specs.

```
[Coll-0]
collPath=/z/app/doc1/
topicSet=/z/app/topics1/
knowledgeBase=/z/app/topics/top1.kb
onLine=2
```

```
[Coll-1]
collPath=/z/doc2/
topicSet=/z/topics2/
knowledgeBase=/z/app/topics/top2.kb
onLine=1
```



# Setting Up a K2 Broker

You start a K2 Broker using a command-line tool called `k2broker`. The `k2broker` tool takes as input the name of a server configuration file called `k2broker.ini`. The section below describes how to use the `k2broker` tool with the `k2broker.ini`.

## k2broker Command-line Tool

The K2 Broker is started from the command line or from a script in the Unix environment and is integrated as a service within the Windows NT environment. The server is designed to run with a minimum of intervention. Most configuration parameters are set in a configuration file, which can be given a user-assigned name (the default file name is `k2broker.ini`).

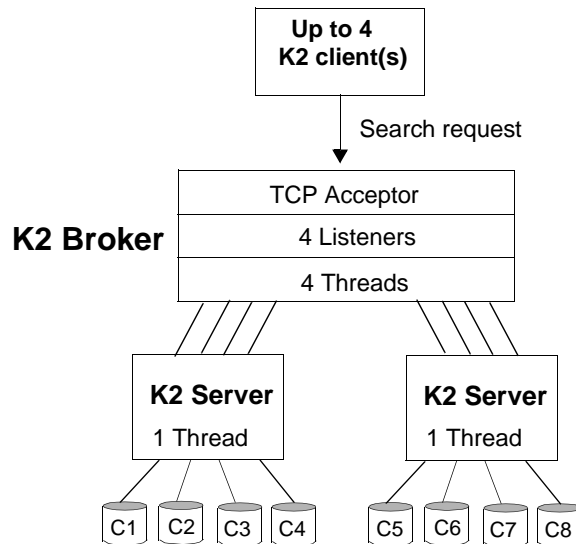
Command line arguments include the name of the configuration file and the TCP port for incoming connection. The K2 Broker is started by using the following command:

```
k2broker [<option1> <option2> ...]
```

The options available for this command are summarized in the table below.

Keyword	Permitted values	Function
-port <value>	Positive integer	Identifies the TCP port number for use by the K2 Acceptor.
-iniFile <filename>	Any valid filename	Identifies the filename to use as the configuration file for this instance of the K2 Broker.
-verbose <value>	0=status 1=informational 2=verbose 3=debug	Determines the amount of information contained in the K2 Broker system messages.
-iniEmit <filename>	Any valid filename	Creates a sample configuration file.
-ntService <value>	1=load as NT service 0=remove as NT service	Used to load or remove the K2 Broker as an NT service. When set to 1, the broker is loaded as an NT service. When set to 0, the broker is removed as an NT service.

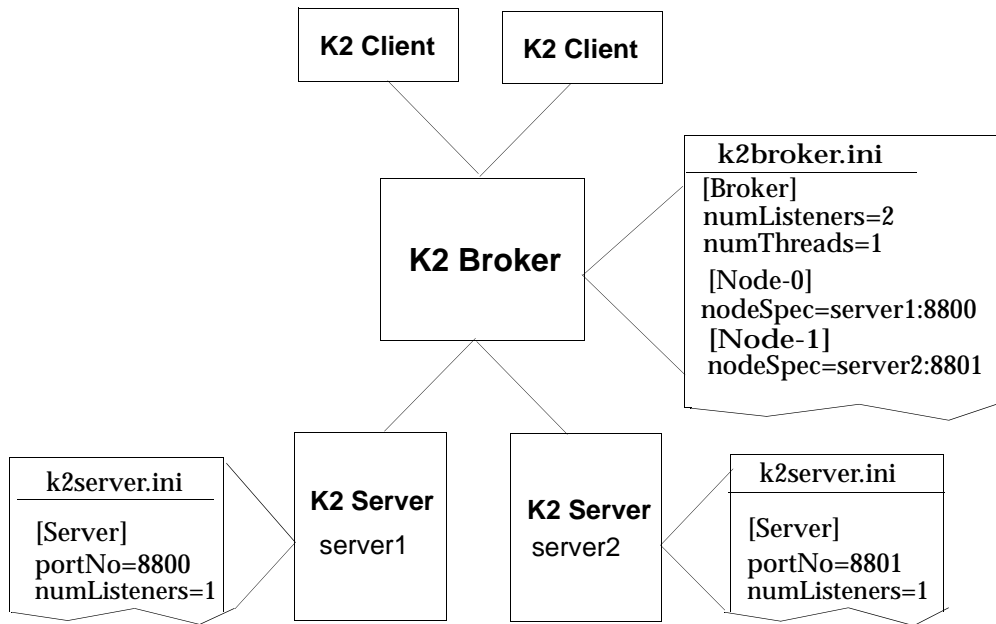
The K2 Broker is used within a multi-tiered, brokered, information retrieval system to balance and distribute search requests among multiple servers. As shown in the following diagram, the architecture of the K2 Broker is similar to the K2 Server, but instead of searching over collections, the K2 Broker searches over servers.



As shown in the preceding diagram, the K2 Broker uses four threads for each server, which each have a total of four threads allocated for search services (one thread for each of four collections). Like the K2 Server, the K2 Broker requires one listener for each concurrent incoming connection.

*Note: an extra Listener thread must be assigned if you wish to create a concurrent K2Watch client connection for monitoring or controlling the broker.*

TCP/IP host names and TCP port numbers are used to configure the connection between the K2 Broker and the servers. In the following illustration, the K2 Broker is configured for two concurrent client connections (`numListeners=2`) and for distributing search requests to two servers, as indicated by the two node definitions. The value for the `numthreads` should be less than equal to the number of listeners as are specified in the `k2server.ini` file.



The entry in the [Nodes] block of the `k2broker.ini` file matches the value of `portNo` in each server's `k2server.ini` file.

## k2broker.ini Configuration File

You configure the K2 Broker using a configuration file, similar to the one provided for the K2 Server. The K2 Broker configuration file is divided into the following sections:

- [Broker] configures the communication services, including the port number, the number of threads, the log file and verbosity levels.
- [Node-*n*] identifies the individual search servers, including host names and TCP port numbers for each server. Increment the block label for each node that you configure, starting with [Node-0].

### Broker Section

The following table summarizes the keywords in the [Broker] section that control the general configuration and operation of a K2 Server instance.

Keyword	Description
<code>vdkHome=</code>	Directory containing Verity resources.
<code>portNo=</code>	TCP port number for client connections. User selected port number.

Keyword	Description
numListeners=	Number of client listener threads that connect at any one time.
numThreads=	The default number of search threads. This value is used only if the numThreads parameter is not set for a particular [node] section.
vdkSortingFlag=	A flag indicating whether the Verity engine will sort at the collection level. Valid values are: NO   False   0 to not perform sorting at the collection level (default) YES   True   1 to perform sorting at the collection level To implement sorting at the collection level you must set vdkSortingFlag to YES in the k2server.ini file (in the [server] section) and the k2broker.ini file (in the [broker] section). See “Sorting Results at the Collection Level” later in this chapter for details about how collection level sorting is implemented.
connTimeout=	Timeout in milliseconds for a server connection. Minimum is 2000 milliseconds. A connTimeout value set to less than 2000 is automatically set to 2000. Each server can override this timeout specification. The value for connTimeout is the default timeout for each server node.  NOTE: The connTimeout value is ignored by the K2ConnectNew function, which has an internal, speed-optimized mechanism that controls timeout.
resultCacheTimeout=	Timeout in milliseconds for the result cache. Timeout occurs after 60 seconds or when the cache overflows based on resultCacheQuota.
resultCacheQuota=	The number of slots per segment for the result cache. The result cache is composed of 16 segments, each of which has a number of slots for caching items in: K2SearchNew, K2SearchRecv, K2DocReadBatch. Timeout occurs after resultCacheQuota value * 16. If resultCacheQuota=10, each of the segments has 10 slots. Note that since a search operation involves a call to K2SearchNew and a call to K2SearchRecv, an additional slot is used. For more information, see “Result Caching.”
resultCacheEnabled=	A flag indicating whether the result cache is enabled. Valid values are: Yes   True   1 to enable the result cache No   False   0 to disable the result cache (default). By default, the cache is not enabled.
resultCacheMaxInBytes=	Amount of memory, in bytes, to use for the cache.

## Node Section

The K2 Broker registers a search server for each node that you identify in the broker configuration file. To add one or more collections to the configuration file, enter a separate block of keywords for each node in the following format:

```
[Node-0]
nodeSpec=<hostname>:<port>
onLine=<2 for on-line; 0 for off-line>; 1 for Hidden>
```

Increment the block label for each server that you configure, starting with `Node-0`. The following table lists the keywords used to configure each server.

Keyword	Description
<code>nodeSpec=</code>	Identifies the server that is to be registered with the K2 Broker in the form <code>&lt;hostname&gt;:&lt;port&gt;</code> where <code>hostname</code> is the fully qualified host name (or IP address) and the TCP port number.
<code>numThreads</code>	Number of concurrent searches to send to the K2 Server from this K2 Broker. This value should be less than or equal to the number of listeners specified in the <code>k2server.ini</code> . If not specified, the value of <code>numThreads</code> from the <code>[node]</code> section is used.
<code>connTimeout</code>	Timeout for this specific server. Default is 2000 milliseconds. This value can be increased if the server has a slow response.
<code>onLine=</code>	A flag indicating whether the K2 Broker starts up with the servers on-line. Valid values are: 0 to start the broker with the servers off-line; 2 to start the broker with the servers on-line (default). Allows you to start the broker with servers on-line (2) or off-line (0).

There can be n-collection specs.

```
[Node-0]
numThreads=2
connTimeout=15
nodeSpec=server2:9902
onLine=2
```

```
[Node-1]
numThreads=2
connTimeout=10
nodeSpec=server2:9903
onLine=2
```

# Sorting Results at the Collection Level

The `vdkSortingFlag` keyword is a flag that indicates whether the Verity engine will sort at the collection level. The `vdkSortingFlag` keyword is valid in the `k2server.ini` file in the `[server]` section and in the `k2broker.ini` file in the `[broker]` section. Valid values are:

- NO | False | 0 to not perform sorting at the collection level (default)
- YES | True | 1 to perform sorting at the collection level

To implement Verity engine sorting at the collection level, you must set the `vdkSortingFlag` keyword to YES in both configuration files, the `k2server.ini` file and the `k2broker.ini` file.

Verity engine sorting at the collection level can improve the quality of the results presented in results lists for some K2 search systems. K2 search systems that can benefit from this feature are ones which process searches under all conditions below occur:

- Searches implement a sort specification (`sortSpec`) based on a primary sort field other than “score”
- Searches set a maximum documents limit (`maxDocs`) for the results list size
- Searches return a number of hits (`numHits`) greater than `maxDocs`

When all conditions above are implemented in a K2 search system, the results lists are likely to not contain the most relevant results using the default K2 sorting behavior. This is because during the merging process, some results are removed from the results list presented to users. Results that are ranked lower than the `maxDocs` number of results (since `maxDocs` is greater than `numHits`) are removed. Using the Verity engine sorting at the collection level changes the default K2 sorting behavior to be like the sorting behavior in Verity Developer’s Kit (VDK).

The disadvantage of using `vdkSortingFlag` is that VDK-style sorting will make search processing slower.

# Result Caching

Result caching can be set for the K2 Broker, in the `k2broker.ini` file, and for the K2 Server, in the `k2server.ini` file, using these keywords:

- `resultCacheEnabled`
- `resultCacheTimeout`
- `resultCacheQuota`
- `resultCacheMaxInBytes`

The `resultCacheQuota` keyword specifies the number of slots per segment for the result cache. The result cache is composed of 16 segments, each of which has a number of slots for caching items in: `K2SearchNew`, `K2SearchRecv`, `K2DocReadBatch`. Timeout occurs after `resultCacheQuota` value \* 16.

If `resultCacheQuota=10`, each of the segments has 10 slots. Note that since a search operation involves a call to `K2SearchNew` and a call to `K2SearchRecv`, an additional slot is used.

The segment that a cache item resides in is determined by creating a 4bit checksum of the source parameters. For example, a cached `K2SearchNew` request would have a checksum generated from the `K2SearchNewArgRec` that was given as input to the search and this is kept alongside the resultant `K2SearchElement` at the K2 Broker or K2 Server.

When considering the distribution of a set of 4bit checksum for sixteen searches, it is unlikely that all sixteen 4bit checksum variations will be generated. Hence, setting the `resultCacheQuota` to a low number will result in earlier cached results being removed before a full sixteen are cached.

It is recommended that you initially set the values for the result caching parameters very high for testing. For example:

```
resultCacheTimeout=60000  
resultCacheQuota=1000  
resultCacheMaxInBytes=16500000
```

You should see the interaction of the K2 Broker and K2 Server diminish and then you can back off the number to a level appropriate for your system.

# Using rck2 as a Search Client

The `rck2` command-line tool allows you to search collections associated with a K2 Server in a K2 Search System.

## rck2 Syntax

The syntax used to start `rck2` from the command line is:

```
rck2 -server <servername> -port <portno>
```

Syntax Element	Description
-server <servername>	The server name for the K2 Server to attach to. The server name is defined in the <code>k2server.ini</code> file. The collections attached to this server will be searched by <code>rck2</code> .
-port <portno>	The port number where the K2 Server (specified in <code>-server</code> ) is running.

## rck2 Command Options

The command options for `rck2` are shown below.

rck2 Command	Description
<code>p &lt;sortspec&gt;</code>	The sort specification for the search results. By default results are sorted by Score. Multiple fields must be specified in a space-separated list using <code>asc</code> or <code>desc</code> to indicate ascending or descending order. For example: <code>p score desc title asc</code>
<code>m &lt;maxdocs&gt;</code>	The maximum number of documents to return in the results list.
<code>c &lt;collections&gt;</code>	The list of collections to search. Multiple collections must be specified in a space separated list. For example: <code>c coll1 coll2 coll3</code>
<code>f &lt;fields&gt;</code>	The list of fields to retrieve. For example: <code>f k2dockey title date</code>
<code>s &lt;query text&gt;</code>	The query (or question) to be used to process the search. The query can be expressed as words and phrases separated by commas. Additionally, the query can include Verity query language, operators and modifiers, as described in the <i>Verity Query Language Reference Guide</i> and the <i>Verity Search Tips Online Guide</i> .
<code>g &lt;collection&gt;</code>	Display collection information.
<code>d &lt;k2dockey&gt;</code>	Display fields for the K2 document key specified.



## Setting Up K2 Servers and K2 Brokers

### Using rck2 as a Search Client

<b>rck2 Command</b>	<b>Description</b>
<code>v &lt;k2dockey&gt;</code>	Stream the document and display it with highlights.
<code>r &lt;docstart&gt;</code>	Display results starting with the first result in the results list. Fields specified using the <code>f</code> command are displayed. Docstart indicates the first result to be displayed. For example, <code>r 10</code> displays results starting with the 10th document in the results list.
<code>b &lt;docstart&gt;</code>	Display results based on the last field selection.
<code>i</code>	Display information about the K2 Server including nodes and collections.
<code>x &lt;score precision&gt;</code>	Set score precision to 8 or 16 bit. By default, 16 bit precision is used.
<code>h</code> or <code>?</code>	Display online help for the <b>rck2</b> command options.

**Setting Up K2 Servers and K2 Brokers**  
Using rck2 as a Search Client

# 4

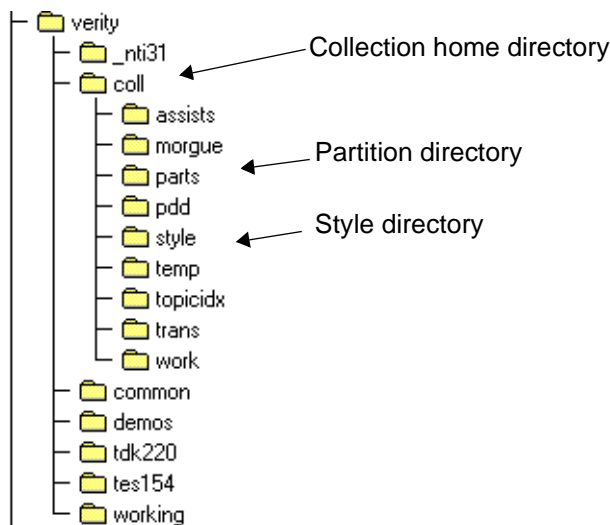
## Collections and Search Performance

This chapter describes how to optimize K2 Server and K2 Broker search performance. The main factor affecting the speed and efficiency of searches performed using the K2 Server and the K2 Broker is the way that collections are built. For this reason, the first section in this chapter describes the architecture of Verity collections and how `mkvdk` can be used to create collections. This chapter is divided into the following sections:

- Overview to Collection Building
- Using the merge Utility
- Using the Incremental Squeeze Feature

# How Collections Work

Each collection is contained within a collection home directory, which contains all of the subdirectories and files required for searching a set of documents. You name the collection home directory when you create a new collection, using any file name permitted by your operating system. The following illustration shows the subdirectories contained within the collection home directory for a typical collection.



## Collection Partitions

The `parts` directory and the `style` directory, shown in the previous illustration, are two of the most important of the subdirectories in every collection. The `parts` directory contains index and attribute tables, divided into separate partitions. The `style` directory contains the configuration files that control the way the collection is generated.

In the `parts` subdirectory, the word index is identified by the extension `did`, while the attribute table uses the extension `ddd`. An 8-digit number identifies the partition to which each pair of word indexes and attribute tables belong. The `pdd` subdirectory contains the partition index that identifies the partitions currently used in the collection.

The index and attribute tables in a single partition can contain information about as many as 64,000 documents. Partitions allow real-time updating of the collection while keeping the collection available for searching by users. To improve search performance, smaller partitions can be consolidated into a single larger partition. To do this, use the

index tuning options provided by `mkvdk`. The optimum size of partitions for a collection depends on the overall size of the collection, the average size of documents, and how often new documents are added to a collection.

## Attributes and Fields

As mentioned earlier, an attribute table is built for each partition in a collection, to allow field searching and for use in formatting search results and source documents for display. The fields within the attribute table are defined by the following collection styles:

- `style.ddd`, which defines fields used internally by the Verity engine (identified by an initial underscore character `_`)
- `style.sfl`, which defines standard fields (many of which are commented out to limit the size of the attribute table)
- `style.ufl`, which is used to define custom fields that are not included in `style.sfl`.

The value of each field can be filled in from source documents or can be provided explicitly, using the `mkvdk` bulk submit option. You can view the contents of the attribute table for a partition by using the following command:

```
browse 00000001.ddd
```

The system displays the following menu of options available for the `browse.exe` utility.

```
D:\VERITY\colltest\parts>browse 00000001.ddd
BROWSE OPTIONS
  ?) help
  q) quit
  c) Number of entries in field
  _) Toggle viewing fields beginning with '_'
  v) Toggle viewing selected fields
  ##) Display all fields in specified record number
Dispatch/Compound field options:
  n) No dispatch
  d) Dispatch
  s) Dispatch as stream
Action (? for help):
```

To display the values of the attributes in a specific document, enter the sequential number of the record. To reduce the number of fields displayed, you can eliminate the fields used internally by the Verity search engine by typing a `v`. To display these fields, type an underscore character.

The following partial display of the results of the `browse` utility includes the fields used internally by the Verity search engine, which start with an underscore `_` character.

```
18 _SECURITY_MI      WRM-unsg ( 4) = 0
19 _SECURITY_MX      WRM-unsg ( 4) = 0
20 _INDEX_DATE_MI    WRM-date  ( 4) = 13-Jul-1997 09:43:33 am
21 _INDEX_DATE_MX    WRM-date  ( 4) = 13-Jul-1997 09:43:33 am
22 DOC                DSP-text  (-1) = d:\verity\doc1.htm
23 DOC_FN             VAR-text  ( 44) = d:/verity/doc1.htm
24 DOC_OF             FIX-unsg  ( 4) = 0
25 DOC_SZ             FIX-unsg  ( 4) = 4294967295
26 DOC_FN_OF         FIX-unsg  ( 4) = 76
27 DOC_FN_SZ         FIX-unsg  ( 2) = 44
28 VdkVgwKey         VAR-text  ( 44) = d:/verity/doc1.htm
29 VdkVgwKey_IX      FIX-unsg  ( 3) = 1
30 VdkVgwKey_MI      WRM-text  ( 44) = d:/verity/doc2.htm
31 VdkVgwKey_MX      WRM-text  ( 44) = d:/verity/doc3.htm
32 VdkVgwKey_OF      FIX-unsg  ( 4) = 76
33 VdkVgwKey_SZ      FIX-unsg  ( 2) = 44
34 Title             VAR-text  ( 20) = Tools for Searching
...

```

Fields may be permanent or transitory and their definition may be internal or external. The values of transitory fields are provided by the Verity search engine and change with each search request. Permanent fields are defined internally by collection style files or externally by other applications, such as databases.

## The Word Index

You can view the contents of the word index for a partition by using **didump.exe**:

```
didump 00000001.did
```

The display provides an alphabetical listing of the words in the word index, shown below.

```
didump - Verity, Inc. Version 2.5.1 (_nti40, Jul 7 1999)
Text          Size      Doc      Word
A             10       3        4
a            34       5       24
abbreviations 4         1         1
about         4         1         1
acronym       5         1         2
acronyms     4         1         1
actual       4         1         1
administrator 3         1         1
advance      3         1         1
...
```

The meaning of each column in the display is as follows:

- Size—the number of bytes used by the Verity engine to store information about the word
- Doc—the number of unique documents in which the word appears
- Word—the number of occurrences

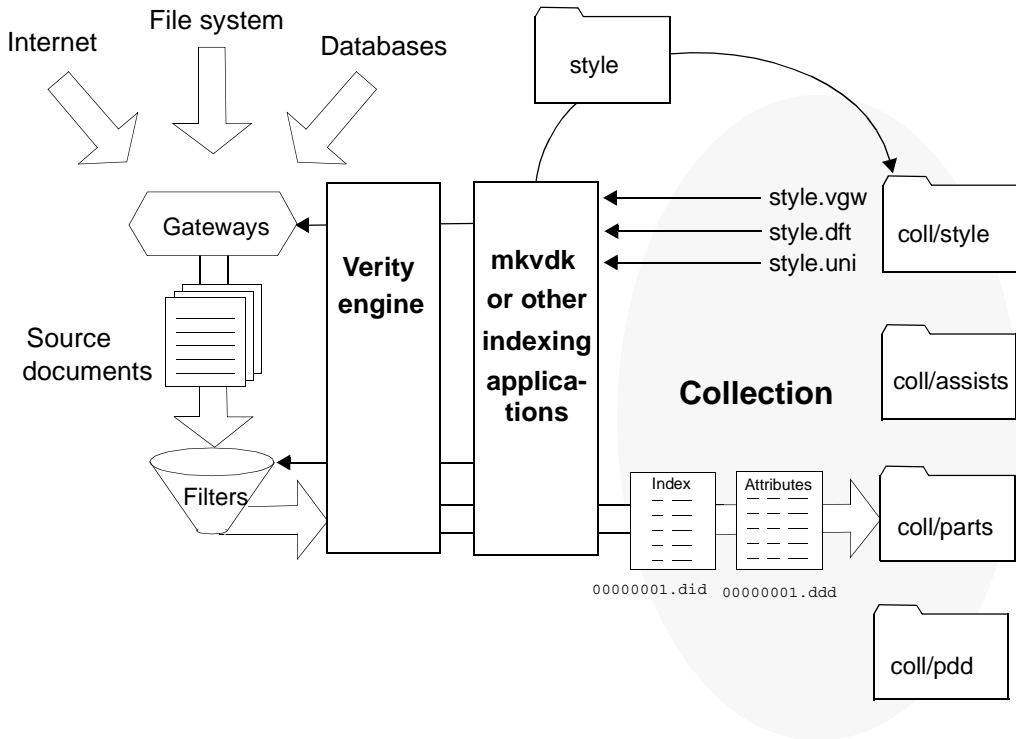
To view the occurrences of a specific word or pattern, enter a command using the `-pattern` option, as in the following example:

```
didump -pattern acronym 00000001.did
```

The **didump** utility will display information about the number of occurrences of the word “acronym.” You can display the number of occurrences of a word in a sentence or paragraph by using the verbose (`-v`) option.

## The Indexing Process

To make a set of documents available for searching, you create a new collection using an indexing application, such as `mkvdk`, and then identify the new collection to the K2 Server. The following illustration shows how an indexing application interacts with the Verity search engine and the components of a collection.



The Verity engine uses a gateway to access the files (or other repositories) containing the document set to be indexed. By default, the file system gateway is used to index documents residing on file systems in various operating system environments. When indexing source documents requiring a different access method, such as HTTP or SQL, the indexing application specifies the appropriate gateway, identified in `style.vgw`.

After opening a document using the gateway, the search engine uses the appropriate filter to perform the following operations:

- Convert the document to indexable text
- Create a field token containing the name and value of each field in the document
- Create a word token for each indexable word in the document

Field tokens are used to populate the attribute table, while word tokens are used to generate the word index. This process is described in further detail in Lesson Two.



## Using mkvdk

The **mkvdk** utility is an indexing application, provided with K2 Toolkit, that can be used in various ways to create and maintain collections. It is a command line utility that can be used within other applications or shell scripts to provide more sophisticated scheduling and other capabilities. The following is the basic syntax of the command:

```
mkvdk -collection path [option] [dockey]
```

Multiple options and dockeys can be included, as needed. If *dockey* is a list of files, it should consist of an at-sign (@) followed by the file name that contains a simple list of files, as in @filelist. The options for **mkvdk** are described in Appendix A.

The following operations occur when you use **mkvdk** to create a new collection:

1. New collection directories are created and the specified style files are copied to the *style* subdirectory.
2. The style file settings are read and the required information is passed to the Verity search engine.
3. The gateway is used to open the document files, which are parsed according to the settings in various style files.
4. A new partition is created, which includes an index and an attribute table.
5. Assist data is generated, which may include a spanning word list.

When problems occur during an operation, **mkvdk** writes error messages to the system log file (*sysinfo.log*). You can direct error and other messages to the console by using **mkvdk** with the *-outlevel* option. You can direct messages to a file of your choice by using the *-loglevel* and *-logfile* options.

The format of the log file is shown below:

Date	Time	Level	Code	Component	Description
(Sun Jul 13	10:06:40	1997):	Info	M2-0106	(Document Index): Writing document index

You can use the log file to view details about what happens during the collection building process. Use the **mkvdk -loglevel** and specify the numeric identifier for the message level you want, as summarized in the following table.

## Collections and Search Performance

### How Collections Work

Type	Number
Fatal	1
Error	2
Warning	4
Status	8
Info	16
Verbose	32
Debug	64

To calculate the numeric parameter, add up the numbers for the message types you want to include. The default for both `-outlevel` and `-loglevel` is 15, which selects fatal, error, warning, and status messages (1+2+4+8).

# Using the merge Utility

The **merge** utility lets you combine multiple collections into a single, large collection. This is useful for merging smaller collections built from different sources into one collection. You can then use the **merge** utility again to break up the collection into smaller collections of a roughly uniform size.

Breaking up a large collection helps to optimize K2 Server search performance, because it allows many brokers, servers, and search services to perform multiple concurrent search requests over the different collections. After breaking up a large collection, you can also discard older collections to reclaim limited disk storage space and to further improve search performance.

To obtain help for the **merge** utility, enter the following command:

```
merge -help
```

*Note: after running the **merge** utility, you must use the **mkvdk -optimize** option to rebuild the collection contents, including the word index and ngram index.*

## Merging Collections

The following is the syntax for using the **merge** utility to merge multiple collections into a single collection:

```
merge <newCollection> <srcCollection1> <srcCollection2>  
[srcCollectionN]
```

The utility reads `srcCollection1`, `srcCollection2` and so on and merges them into a single collection with the directory name given for `newCollection`. If the directory name given for `newCollection` doesn't exist, then it is created.

## Splitting Collections

The following is the syntax for using the **merge** utility to split a single large collection into smaller collections:

```
merge-split <srcCollection> <newCollection1> <newCollection2>  
[newCollectionN]
```

The utility reads `srcCollection` and splits it in roughly equal-sized pieces, using the filenames given for `newCollection1` and so on.

If you want to split a very large collection into a large number of new collections, you can use the following option instead of explicitly naming each new collection:

```
merge -split -number newCollection srcCollection  
newCollectionXX
```

## Collections and Search Performance

Using the merge Utility

The utility reads the collection identified by `srcCollection` and splits it into the number of segments specified by the `-number` option. The name of the first new collection is generated by appending the first two letters in the alphabet (aa) to the directory name given for `newCollection`. Each subsequent filename is generated by incrementing one of the appended letters (up to zz) for a maximum of 676 partitions. For example, if the value of `-number` is 3, and the value of `newCollection` is `Collection1`, the collections are named, `Collection1aa`, `Collection1ab`, and `Collection1ac`.

*Note: the maximum length of the directory name given for `newCollection` is 2 characters less than the length allowed by the filesystem.*

# Using the Incremental Squeeze Feature

The incremental squeeze is a collection optimization feature. Using incremental squeeze, the application administrator can save on the disk space required for squeezing a collection.

Incremental squeeze uses significantly less disk space to squeeze a collection than a normal squeeze. If normal squeezing is performed, the disk space required can be up to double the size of a collection. When a squeeze is performed on a collection, all of the collection's partitions that have deleted documents are recreated without the deleted documents so that continuous access to the collection is possible. After the reap interval time, the original partitions (with the deleted documents) are removed. Before the reap interval time, both old and new partitions exist, sometimes occupying almost double the space.

Using incremental squeeze, the Verity engine first brings down the collection, and then squeezes the partitions in the collection one by one. After each partition is squeezed into a new partition, the corresponding old partition is removed immediately before the next partition is squeezed. After squeezing all of the partitions, the engine brings the collection back up. The behavior of incremental squeeze ensures that the extra disk space required to squeeze a collection is equal to the size of the largest partition at most.

To implement incremental squeeze, you run the **mkvdk** command-line tool with a set of style files, including a `style.plc` file that has a special `/incremental_squeeze=YES` entry. The **mkvdk** syntax used to invoke the incremental squeeze feature is:

```
mkvdk -collection coll -optimize squeeze
```

where `coll_name` represents the collection name.

The `style.plc` entry for incremental squeeze is specified as an attribute to the indexing mode used. A sample `style.plc` file is below.

```
$control: 1
  policy:
  {
    mode: default
      /inherit=generic
      /incremental_squeeze=yes
  }
```

With the sample `style.plc` file, the Verity engine uses the incremental squeeze feature when an **mkvdk** call is made with `-optimize squeeze`. The default indexing mode does not implement incremental squeeze.

**Collections and Search Performance**  
Using the Incremental Squeeze Feature

# 5

## Verity KeyView Filters

Verity KeyView Filters V6.5 are packaged with this version of the Verity K2 Toolkit. The KeyView filters support indexing and viewing documents stored in many popular desktop publishing, word processing, and presentation formats. This chapter covers these topics:

- Key Features
- Supported Formats
- New Features and Enhancements

# Key Features

The KeyView filters offer these key features:

- Threadsafe filtering of simultaneous documents
- Fast, reliable performance
- Filters for popular formats, including word processing, desktop publishing, spreadsheets and presentations
- Viewing with highlights and navigational features
- Memory management
- Multiplatform support
- Automatic extraction of OLE properties as Verity fields



# Supported Formats

The KeyView feature list table below identifies supported formats and filter names. For each format, information about meta data extraction is provided. These keys are used in the feature list table:

- **Format**—The document format and platform limitations. All KeyView filters are supported on all Verity-supported platforms with one exception. The Lotus Word Pro filter is supported on the Windows NT platform only.
- **Version**—The version(s) supported.
- **Filter Name**—The filter executable used to filter the document format.
- **Summary Info**—A characterization of the summary information extracted for the document format. For each filter format,
  - YES indicates OLE summary information properties are extracted.
  - PARTIAL indicates the filter is invoked to extract partial meta information, whatever meta information it can.
  - NO indicates the filter does not extract meta information.
- **Character Set**—Specifies whether the filter passes character set information.
  - YES indicates the filter can determine the character set information is in and pass that information up stream.
  - NO indicates the filter does not pass any character information.

## Features List

Format	Version	Filter Name	Summary Info	Character Set
ASCII text	All	afsr	NO	NO
Applix Spreadsheet	v4.0 to 4.4	assr	NO	NO
Applix Word	v4.2, 4.37, 4.4	awsr	NO	NO
MS Excel csvr format		csvsr	NO	NO
DCA-RTF	SC23-0758-1	dcasr	NO	YES
IBM DisplayWrite	v1.0, v1.1	dw4sr	NO	YES
HTML	All	htmlsr	PARTIAL	YES
Lotus 1-2-3	'96, '97	l123sr	PARTIAL	YES
Lotus AMI Pro	v2.0, v3.0	lasr	PARTIAL	YES

## Features List

Format	Version	Filter Name	Summary Info	Character Set
Lotus Word Pro (NT only)	'96, '97	lwpsr	PARTIAL	YES
MS Word for Mac	v4.x, 6.x	mbsr	NO	NO
MS Word for Mac '98	'98	mbsr	YES	NO
Adobe FrameMaker MIF	v5.5	mifsr	NO	YES
MS Word	v1.x, v2.0	misr	NO	NO
MS Microsoft Works	v3.0, v4.0	mswsr	NO	NO
MS Word	v6.0, v7.0	mw6sr	YES	YES
MS Word	v8, 2000	mw8sr	YES	YES
MS Word for DOS	v2.2 to v5.0	mwsr	NO	NO
MS Works Spreadsheet		mwssr	NO	YES
MS PowerPoint PC	v4.0	ppcsr	PARTIAL	NO
MS PowerPoint	'95, '97	pptsr	PARTIAL	YES
Lotus Freelance	'96, '97	przsr	NO	NO
Corel QuattroPro	v7.0, v8.0	qpssr	PARTIAL	YES
MS Rich Text Format		rtfsr	PARTIAL	YES
Corel Presentations	v7.0, v8.0	shwsr	YES	NO
UNICODE		unizr	NO	YES
Lotus 1-2-3	v2, v3, v4	wkssr	NO	YES
Word Perfect for DOS	v5.0, v6.0	wosr	NO	YES
Word Perfect for Windows	v7.0	wp6sr	PARTIAL	YES
Word Perfect for Mac	v2.0, v3.0	wpmsr	NO	YES
MS Excel	v3, 4, 5, '96, '97, 2000	xlssr	YES	YES
XyWrite	v4	xywsr	NO	NO

## KeyView Filters—Limitations

Lotus Word Pro filter is supported on Windows NT only. This filter is not supported on UNIX platforms.

The output of the KeyView filters is in the character set of the current Verity locale. If the filters output characters which are not present in the target Verity locale's character set, the characters appear as ? (question mark) or ~ (tilde). For example, when the “n superior” character is not defined in the target character set, the “n superior” character appears as a ? in the filtered text.

# New Features and Enhancements

This release includes the following new features and enhancements:

- Support for indexing headers and footers in word processing documents. See “Headers and Footers” below for implementation details. When headers and footers are indexed, they can be searched.
- New page break handler improves tokenization. More accurate tokenization improves the quality of searches.
- WordPerfect character mapping is no longer hard coded to code page 1252. Now the WordPerfect filter uses the internal character mapping mechanism used by the other KeyView filters.

## Headers and Footers

The KeyView filters can now index and search header and footer information. When indexed, the header and footer information becomes part of the virtual document. Any full text search will search and display the header and/or footer text.

By default, headers and footers are not indexed unless you make modifications to the collection’s `style.uni` file. Headers and footers will not be displayed unless the filter type definition in the `style.uni` file is edited.

In the `style.uni` file, edit the `/format-filter` modifier (under the `type: statement` for each filter type), by adding the `-headfoot` flag. For example, to support headers and footers in Microsoft Word documents, change the default `style.uni` file, so the `/format-filter` modifier looks like this:

```
type: "application/msword"  
  /format-filter = "flt_kv -headfoot"  
  /charset      = guess  
  /def-charset  = 8859
```

**NOTE:** The `/format-filter` modifier value must be enclosed in quotation marks, as shown above.

# 6

## XML Support

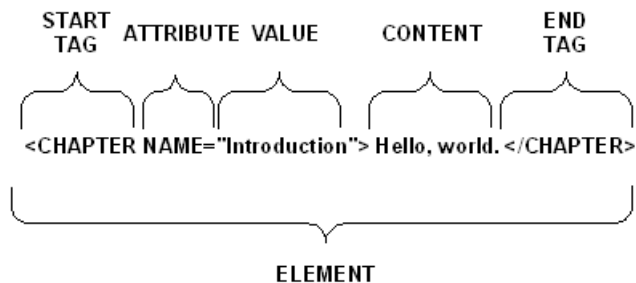
Verity K2 Toolkit V2.2 includes support for XML documents. This chapter describes the new XML filter and how to use it. These topics are covered:

- Requirements for Data Files
- Implementation Summary
- Style File Configuration
- Indexing XML Documents

# Requirements for Data Files

To be properly indexed, XML data files must be well-formed XML documents as specified in the Extensible Markup Language Recommendation (<http://www.w3.org/TR/REC-xml>).

Briefly stated, a well-formed XML document contains elements that begin with a start tag and terminate with an end tag. One element, which is called the root or document element, cannot appear in the content of another element. For all other elements, if the start tag is in the content of another element, the end tag is also in the content of the same element.



The XML data files must have an `.xml` extension if the universal filter is used. If documents do not have an `.xml` extension, you can index XML documents into an XML-only collection by specifying the XML filter in the `style.dft` file.

# Implementation Summary

Verity support for XML documents is implemented by a new XML filter and controlled using a number of style files.

## XML Filter

The new XML filter (`flt_xml.dll`, `flt_xml.sl`, `flt_xml.so`) resides in the `bin` directory for the installed platform.

## Style Files

The following style files are required to enable indexing of XML files. Default style files reside in the `\common\style` directory.

Style File	Description
<code>style.uni</code>	Invokes the XML filter for indexing XML documents.
<code>style.xml</code>	Modifies the default behavior of the XML filter. (optional)
<code>style.ufl</code>	Defines custom fields in XML documents. The fields must also be defined in the <code>style.xml</code> file.
<code>style.dft</code>	Invokes the Verity universal filter by default so all document types can be indexed into one collection. You can modify the <code>style.dft</code> file to invoke the XML filter instead of the universal filter, as described below.

# Style File Configuration

This section discusses style file configuration used to support XML document filtering.

## style.uni File

To index XML documents, the `style.uni` must include the following lines:

```
type: "text/xml"  
  /format-filter = "flt_xml"  
  /charset       = guess  
  /def-charset   = 8859
```

**NOTE:** Some versions of the `style.uni` specify that `text/xml` content be handled by `flt_zone`. This specification should be replaced with the above construct.

## style.xml File

By default, the XML filter indexes regions of the document delimited by XML tags as *zones*, with the zones given the same name as the XML tag. META tags are automatically indexed as *fields* unless they are in a suppressed region.

The `style.xml` file enables administrators to change the default behavior of the indexer for XML documents. Administrators can specify field and zone indexing for regions of the document delimited by XML tags and skip regions of the document delimited by XML tags.

## Command Syntax

```
<command attribute="value"/>
```



## Command Summary

Command	Description
field	Indexes the content between the pair of specified XML tags as field values. By default, the field name is the same as the xmltag value, unless otherwise specified by the fieldname attribute. <i>Attributes:</i> xmltag fieldname index
ignore	Skips indexing of xmltag but indexes the content between the pair of specified XML tags. <i>Attributes:</i> xmltag
preserve	Indexes specified xmltag as a zone if preceded by ignore xmltag="*". <i>Attributes:</i> xmltag
suppress	Suppresses every xmltag embedded within the specified xmltag. <i>Attributes:</i> xmltag

## Command Examples

The following command ignores all XML tags in the document, indexing only the content:

```
<ignore xmltag = "*" />
```

The following command skips indexing the specified xmltag but indexes the content between the start and end tags of the specified xmltag:

```
<ignore xmltag = "section_1" />
```

The following command indexes xmltag as a zone if there is also an ignore xmltag="\*" command:

```
<preserve xmltag = "section_1" />
```

The following command suppresses the entire element identified by xmltag. The tag, attribute, and content are not indexed:

```
<suppress xmltag = "section_1" />
```

The following command indexes the content between the start and end tags of the specified xmltag as a field, which is given the same name as xmltag:

```
<field xmltag = "column_1" />
```

The following command indexes the content between the start and end tags of the specified `xmltag` as a field, which is given the name specified in the `fieldname` attribute:

```
<field xmltag = "column_2" fieldname = "vdk_field_2"/>
```

The following command indexes the content between the start and end tags of the specified `xmltag` as a field, overriding any existing value of the field:

```
<field xmltag = "column_2" index = "override"/>
```

**NOTE:** Both `fieldname` and `index` attributes can be used in a field command.

### **style.ufl File**

If administrators have defined custom fields to be populated in the `style.xml` file, the fields must also be defined in the `style.ufl` file or `style.sfl` file, using standard syntax.

### **style.dft File**

To create a collection that contains only XML documents, administrators can modify the `style.dft` file to invoke the XML filter directly. In this case, the XML documents do not need an `.xml` extension.

The `style.dft` must include the following lines:

```
$control: 1
dft:
{
  field: DOC
    filter="flt_xml"
}
```

# Indexing XML Documents

To prepare for indexing XML documents:

1. Make sure that the XML filter (`flt_xml.dll`, `flt_xml.sl`, `flt_xml.so`) resides in the `bin` directory for the installed platform.
2. Make sure that the `style.uni` contains the directive for invoking the XML filter.
3. If custom fields or zones are required, define them in the `style.ufl` file.
4. Specify custom fields to be populated in the `style.xml` file, as appropriate.

## Indexing using mkvdk

To index XML documents using a command-line indexer, issue these commands:

```
mkvdk -create -style styledir -collection collname  
mkvdk -collection collname file1.xml file2.xml filen.xml
```

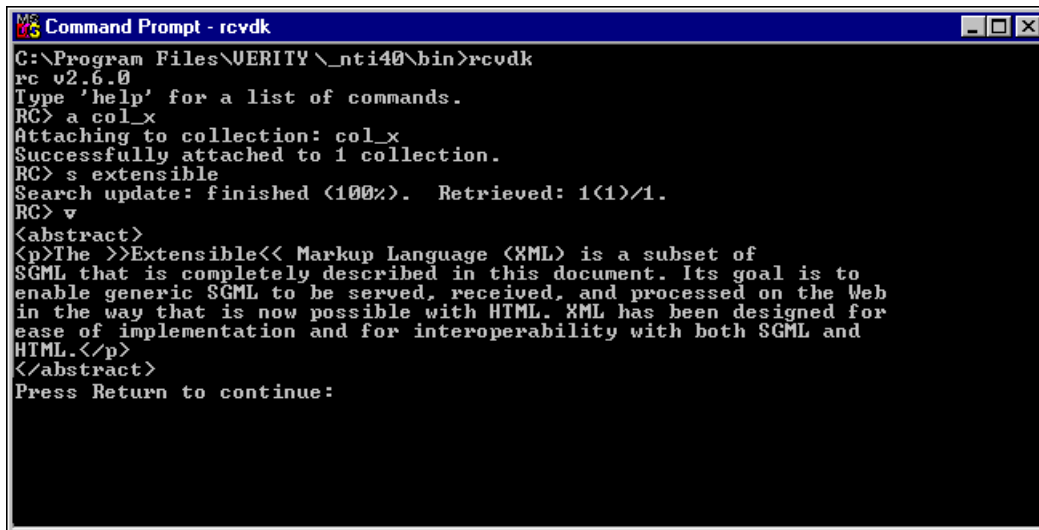
Or using a file list (`flist.txt`):

```
mkvdk -create -style styledir -collection collname @flist.txt
```

The specified style directory must contain the modified `style.uni` and `style.xml` files to enable XML document indexing support.

## Searching using rcvdk

Use `rcvdk` to search and view a collection containing XML documents. The following illustration provides a typical command sequence.



```
Command Prompt - rcvdk
C:\Program Files\VERITY\_nti40\bin>rcvdk
rc v2.6.0
Type 'help' for a list of commands.
RC> a col_x
Attaching to collection: col_x
Successfully attached to 1 collection.
RC> s extensible
Search update: finished (100%). Retrieved: 1(1)/1.
RC> v
<abstract>
<p>The >>Extensible<< Markup Language (XML) is a subset of
SGML that is completely described in this document. Its goal is to
enable generic SGML to be served, received, and processed on the Web
in the way that is now possible with HTML. XML has been designed for
ease of implementation and for interoperability with both SGML and
HTML.</p>
</abstract>
Press Return to continue:
```

## Verity Locales

Verity provides many predefined locales that you can use right away. Locales have several components including a linguistics package (stemmer, tokenizer, natural language processing capabilities). The linguistics package is based on technology from Inight™ LinguistX™.

Verity applications include multibyte support and new localization. The multibyte support enables Verity applications to be localized into most languages, including Asian languages.

# Verity Locales and their Components

Verity packages support for each language (or region) as a separate locale. A Verity locale includes all the components specific to the language, such as a character set mapping, date formats, a message database, a linguistics package (stemmer, tokenizer, natural language processing capabilities).

## Predefined Locales

For English, a Verity locale including a linguistics package is included in the standard product. For languages other than English, you can install and use predefined locales to localize your application. The linguistics package available from Verity incorporates technology from Inxight LinguistX.

Verity Locales Using LinguistX are bundled with K2 Toolkit and are available from Verity upon request. Locales are available for these languages: English, French, German, Danish, Dutch, Finnish, Italian, Norwegian, Portuguese, Spanish, Swedish.

## Custom Locales

Tools and templates are available for creating a custom locale for implementing a custom localization. For further information about building a custom locale, contact your Verity sales representative.

In the case of many Asian languages, the localization process must be complemented by the development of a binary-compatible library that conforms to K2 Toolkit interface requirements for text analysis.

## Tokenization for Locales other than English

For locales such as English that can interpret spaces as word separators and do not require a locale-based tokenizer (word breaker), the search engine can interpret words, phrases, and sentences appropriately. For other locales, such as Japanese and Chinese, the application needs to provide a tokenizer to determine how to break up words and interpret sentences and phrases appropriately. If your application uses a custom tokenizer, some of the query language description in this guide may not be accurate based on your tokenizer implementation.

## Locales from Verity Partners

Some Verity OEM partners have produced custom locales for languages requiring specialized linguistics packages. Currently, locales from our partners are available in these languages: Chinese (simplified and traditional), Japanese, Korean. Other locales are under development now. For information about these locales, please contact your Verity sales representative.

# About Verity Locales

Predefined locales using LinguistX for numerous languages are available from Verity. The LinguistX technology offers these basic components:

- Stemmers—Used to stem words during indexing
- Tokenizers—Used to tokenize documents in streams
- Lexical analysis—Used to do part-of-speech recognition, important for some Verity features like clustering, summarization, QBE parsing
- Phrase extraction—Used to extract phrases, important for some Verity features like clustering, summarization, QBE parsing
- Thesaurus—Allows users to search for synonyms, using the Verity `THESAURUS` operator

**NOTE:** The natural language processing capabilities (lexical analysis and phrase extraction) are used by the Verity engine to perform feature extraction. Without these capabilities, the performance of clustering, summarization, and query-by-example (QBE) parsing within a K2 Toolkit application degrades significantly.

## Verity Locales Using LinguistX

Verity offers predefined locales for several languages.

Language	Locale Name	Locale's Internal Character Set
Bokmal	bokmalx	8859
Danish	danishx	8859
Dutch	dutchx	8859
English	englishx	8859
Finnish	finnishx	8859
French	frenchx	8859
German	germanx	8859
Italian	italianx	8859
Nynorsk	nynorskx	8859
Portuguese	portugx	8859
Spanish	spanishx	8859
Swedish	swedishx	8859

### Upgrading from IntelliScope to LinguistX Locales

If your application used an IntelliScope locale and you upgrading your application using a LinguistX locale, you must reindex your collections.



## Using the THESAURUS Operator

A thesaurus is not included with all LinguistX locales at this time.

If the thesaurus operator is used without a thesaurus present in the locale, the search engine finds stemmed variations of the search term given. The first time the THESAURUS operator is used during a K2 Toolkit session, warnings are reported stating “error initializing vdk20.thd” and “no support for <Thesaurus>”. These messages are warnings only.

If you have topic names with the THESAURUS operator in them and/or you anticipate users will try to use the operator, you can suppress the warning messages by creating an empty thesaurus file and copying it into the locale directory.

1. Create an empty synonyms file, named `myfile.ctl`, which looks like this:

```
$control:1
synonyms:
{
}
$$
```

2. Run the `mksyd` utility to create a thesaurus file:

```
mksyd -syd vdk20.syd -f myfile.ctl
```

As a result, a thesaurus file named `vdk20.syd` is created.

3. Copy the resulting `vdk20.syd` file into the locale directory.

# Installing Predefined Locales

Verity Locales Using LinguistX are bundled on a CD separate from K2 Toolkit. There is a separate installation program for the locales CD. For complete information about running the installation program refer to *Verity Locales Using LinguistX Release Notes*.

The installation program stores Verity locales in this directory:

```
install_dir/common/
```

For each language installed, two locale directories are installed (except for English). For a language other than English, there is a root locale name plus a locale name ending in “x”. The contents of the root locale name and the locale name ending in “X” are identical.

For example, if you installed the Portuguese locale, there will be two directories in `install_dir/common`:

```
portug  
portugx
```

If you are installing the LinguistX-based locale for English, note that the installation program only installs one locale directory called “englishx”. The “english” directory contains the default locale directory included with the product.

**Important!** Do not delete the default “english” directory, even if your application will not use the default english locale. The Verity search engine uses the default english directory and it must be present to ensure proper functioning of your application.

The following table lists the locale directory names with the files contained in each.

	xlt.clg	xlt.da	xlt.dct	xlt.ds	xlt.hmm	xlt.ia	xlt.is	xlt.npr	xlt.tok	html.dct
<b>bokmalx</b>	x		x		x	x	x	x	x	x
<b>danishx</b>	x		x		x	x	x	x	x	x
<b>dutchx</b>	x		x		x	x	x	x	x	x
<b>englishx</b>	x		x	x	x	x	x	x	x	x
<b>finnishx</b>	x	x	x	x	x			x	x	x
<b>frenchx</b>	x		x		x	x	x	x	x	x
<b>germanx</b>	x		x		x	x	x	x	x	x
<b>italianx</b>	x		x		x	x	x	x	x	x
<b>nynorskx</b>	x		x		x	x	x	x	x	x
<b>portugx</b>	x		x		x	x	x	x	x	x

## Verity Locales

Installing Predefined Locales

	<b>xlt.clg</b>	<b>xlt.da</b>	<b>xlt.dct</b>	<b>xlt.ds</b>	<b>xlt.hmm</b>	<b>xlt.ia</b>	<b>xlt.is</b>	<b>xlt.npr</b>	<b>xlt.tok</b>	<b>html.dct</b>
<b>spanishx</b>	x		x		x	x	x	x	x	x
<b>swedishx</b>	x		x		x	x	x	x	x	x

# Localized Query Language

The Verity locales include translated query language operators and modifiers, as shown in the tables below. The Verity search engine allows the English representation of operators and modifiers when enclosed in angle brackets. This section covers:

- Operator Names in Danish, Dutch, Finnish, French, German
- Operator Names in Italian, Norwegian, Portuguese, Spanish, Swedish
- Using English Query Language for Locales other than English

## Operator Names in Danish, Dutch, Finnish, French, German

English	Danish	Dutch	Finnish	French	German
Not	Ikke	Not	Ei	Sauf	Nicht
Case	Tilfælde	Case	Tapaus	Casse	Fall
Many	Mange	Many	Monta	Plusieurs	Viele
Accrue	Påløbe	Accrue	Lisää	Cumul	Aufbau
Or	Eller	Or	Tai	Ou	Oder
And	Og	And	Ja	Et	Und
All	Alle	Alle	Kaikki	All	Alle
Paragraph	Paragraf	Paragraph	Kappale	Paragraphe	Absatz
Sentence	Sætning	Sentence	Lause	Phrase	Satz
Phrase	Frase	Phrase	Ilmaus	Expression	Phrase
Any	Enhver	Elk	MikäTahansa	Quelconque	Beliebig
Topic	Emne	Topic	Aihe	Concept	Topic
Field	Felt	Veld	Kenttä	Champ	Feld
Word	Ord	Word	Sana	Mot	Wort
Stem	Stilk	Stem	Runko	Racine	Stem
Soundex	Soundex	Soundex	Soundex	Consonnance	Soundex
Thesaurus	Thesaurus	Thesaurus	Tesaurus	Synonyme	Thesaurus
Wildcard	Wildcard	Wildcard	Yleismerkki	Troncature	Wildcard
Substring	Understring	Substring	Alamerkkijono	SousChaîne	Substring
Starts	Starter	Starts	Alkaa	Début	Start
Ends	Slutter	Ends	Loppuu	Fin	Ende
Matches	Matches	Matches	Täsmää	Correspon- dance	Übereinstim- mungen
Through	Igennem	Through	Läpi	Intervalle	Durch

**Verity Locales**  
Localized Query Language

English	Danish	Dutch	Finnish	French	German
Contains	Indeholder	Contains	Sisältää	Contenu	Enthält
Near	Nær	Near	Lähellä	Proche	Nahe
In	I	In	Kohteessa	Dans	In
Order	Bestilling	Order	Pyyntö	Ordre	Bestellung
When	Når	Wanneer	Kun	Quand	Wenn

## Operator Names in Italian, Norwegian, Portuguese, Spanish, Swedish

English	Italian	Norwegian	Portuguese	Spanish	Swedish
Not	Non	Ikke	Salvo	Excepto	Inte
Case	Cassa	Case	Cas	Caso	Fall
Many	Molti	Mange	Muitos	Muchos	Många
Accrue	Cumulo	Akkumulering	Acumulação	Cúmulo	Addera
Or	Opp	Eller	Ou	O	Eller
And	E	Og	E	Y	Och
All	Tutti	Alle	Tudo	Todo	Alla
Paragraph	Paragrafo	Avsnitt	ParGrafo	P	Paragraf
Sentence	Frase	Setning	Frase	Frase	Mening
Phrase	Espressione	Frase	Expressão	Expresión	Fras
Any	Qualsiasi	Vilkårlig	Um	Cualquiera	Någon
Topic	Tema	Emne	Tópico	Concepto	Ämne
Field	Campo	Felt	Campo	Campo	Fält
Word	Parola	Ord	Palavra	Palabra	Ord
Stem	Radice	Stamme	Raiz	Raíz	Huvud
Soundex	Consonanza	Soundex	Consonância	Consonancia	Soundex
Thesaurus	Thesaurus	Thesaurus	Sinónimos	Sinónimo	Lexikon
Wildcard	CarattereJolly	Jokertegn	Car	Comodín	Joker
Substring	Sottostringa	Delstreng	Subcadeia	Subcadena	Delsträng
Starts	Inizio	Starter	Início	Inicio	Startar
Ends	Fine	Ender	Fim	Fin	Avslutar
Matches	Concordanza	Samsvar	Corre- spondência	Correspon- dencia	Matchar
Through	Intervallo	t.o.m.	Intervalo	Intervalo	Genom
Contains	Contenuto	Inneholder	Conteúdo	Contenido	Innehåller

<b>English</b>	<b>Italian</b>	<b>Norwegian</b>	<b>Portuguese</b>	<b>Spanish</b>	<b>Swedish</b>
Near	Vicino	Nær	Próximo	Cercano	Nära
In	In	In	Em	En	I
Order	Ordine	Orden	Ordem	Orden	Ordning
When	Quando	Når	Quando	Cuando	När

## Using English Query Language for Locales other than English

If you are using a locale other than English, and your application has defined locale-specific names for the query language in the message database, users can use English query language to compose their queries. To specify English query language, you need to:

- Enclose the English operator/modifier name in braces
- Prefix the operator/modifier name with a pound sign (#)
- If multiple query language elements are used, prefix each element with a pound sign (#)

For example, to search for the phrase “Verity Inc” when using a non-English locale, you enter the following query:

```
<#Phrase> Verity Inc
```

Using explicit syntax you can specify the above query, as follows:

```
<#Phrase> (Verity, Inc)
```

If you want to search for stemmed variations of the word “computer,” you can enter the following query:

```
<#Many><#Stem> computer
```

English operator/modifier names can be used by putting a pound sign in front of each operator or modifier name, regardless of which localization is running. For some languages, using the pound sign will resolve ambiguities. For example, the word “Phrase” means “Sentence” in French.

# A

## Reference for mkvdk

This appendix provides syntax reference for **mkvdk**, a command-line tool to build and maintain collections for use with Verity software. This appendix is divided into the following sections:

- Overview
- Basic Operations
- Optimization, Modes, and Service Options
- Advanced Features

# Overview

## Default Behavior

The default behavior of `mkvdk` is to submit and index documents specified in the command, and to service the specified collection. `mkvdk` automatically sets the service level to support the work you request. Indexing work is queued to happen in the background as resources become available. This includes inserts, updates, deletes, building the word list, and service. You can cause indexing work to happen immediately by using the `-synch` option.

## Document Path Names in Collections

A Verity collection of files stores the paths to those files in one of two ways:

- The *relative path* tells how to get to the file relative to the directory that contains the collection
- The *absolute path* indicates a global file name

For a document, either a relative or absolute path can be stored in the collection. Verity collections can contain a mixture of relative and absolute path names. When indexing a document, the Verity search engine uses the type of path specified in the command line or in the file list. The concept of “current working directory” has no significance to the engine.

In general, relative paths are the most versatile and portable. We recommend their use whenever possible, particularly with the Verity search engine. An easy way to manage at once a set of documents and its associated collection is to set up a parent directory. The collection and document text can then easily be moved as a unit by using the parent directory as a handle.

However, there are two situations in which absolute paths are preferable:

- Collections on Windows require absolute paths unless the search is being conducted from the same drive that contains the collection and the documents. This is because these systems use drive letters. It is not possible to create a relative path that crosses from one drive letter to another. Thus, relative paths cannot be used if a document exists on a different drive from the collection.
- Absolute paths are also required when the collection and document text will not be moved together as a unit. This might happen when the two are stored in directories far removed from each other. This would also be the case when the data are owned by another application.



In general, UNIX systems are the most flexible. Their symbolic linking facilities can be used to work around tricky situations. When collections are built on Windows with absolute paths (starting with drive letters), symbolic links with names like E: can be created on UNIX in order to use the collections.

## Basic Syntax

The following syntax is valid for `mkvdk`:

```
mkvdk -collection path [option] [...] [dockey] [...]
```

Brackets ([]) indicate optional items. An ellipsis (...) indicates repetition of the previous item. Thus, [*dockey*] [...] indicates an optional series of *dockey* items. *dockey* can be a document file name, or a list of document file names. If *dockey* is a list of files, it should consist of an at-sign (@) followed by the file name containing the list, as in @filelist.

You must provide the path of the collection to work on. Additional options are described in the tables that follow. All options must precede the first *dockey*.

# Basic Operations

This section describes the options used for the following collection building operations:

- General Options
- Creating New Collections
- Updating Existing Collections
- Managing Collection Disk Space
- System Messages

## General Options

The following table summarizes required and options and options that can be used for a variety of operations.

Option	Description
-collection <i>path</i>	This option specifies the path of the collection to create or open. This is required to execute <b>mkvdk</b> .
-datapath <i>path</i>	This option specifies the datapath to use to find documents being added to the specified collection. All relative document paths will be relative to this setting. If you do not set this option, <b>mkvdk</b> looks for documents next to the collection directory.
-common	This option specifies the path of the Verity <code>common</code> directory. If you do not use this option, the Topic engine looks for the <code>common</code> directory in the directory containing the <b>mkvdk</b> executable, and then along the executable search path, determined by your operating system environment settings.
-help	This option displays <b>mkvdk</b> syntax options.
-debug	This option runs <b>mkvdk</b> in debugging mode.
-about	This option shows information about the collection, such as its description and the date when it was last modified.
-nolock	This option turns off file locking. Locking is on by default.
-synch	This option performs work immediately. If this option is not used, indexing work is done in the background, as time permits.
-noexit	Windows and Mac only. This option causes the I/O window to remain after the program is finished. By default, the window closes and the program exits so that scripts calling <b>mkvdk</b> will not hang.

## Creating New Collections

The following table summarizes the options that are used when creating a new collection.

Option	Description
-create	This option creates a collection in the specified <code>-collection</code> directory. It creates the directory structure, determines the index contents and sets up the documents table schema according to the style files used. If the specified collection already exists, <code>mkvdk</code> exits rather than overwriting the existing collection.
-style <i>dir</i>	This option specifies the style directory that contains the style files to use in creating a collection. This option can only be used with the <code>-create</code> option. If you do not specify this option when you use <code>mkvdk</code> to create a collection, <code>mkvdk</code> uses the style files in the <code>common/style</code> directory.
-description <i>desc</i>	This option sets the collection's description. Enter any alphanumeric text you like, surrounded by quotes (such as "This collection contains electronic mail from ABC Company.")
-words	This option builds the word list for each of the individual partitions in the collection.
-topicset <i>path</i>	This option creates a topic index for the collection based on the specified topic set and stores it in the collection directory. This facilitates efficient searches over the collection when using topics.

## Updating Existing Collections

The following table summarizes the options used to update existing collections.

Option	Description
-insert	This option adds documents to the collection. This is the default option for <code>mkvdk</code> .
-update	This option adds documents to the collection, replacing all previous information about the specified documents.
-delete	This option marks the specified documents as deleted and makes them unavailable for searches. To actually remove deleted documents from the collection's internal documents table and word indexes, use the <code>squeeze</code> keyword.
-persist	This option services the collection repeatedly, at default intervals of 30 seconds. Use the <code>-sleeptime</code> option to set a different interval.
-sleeptime <i>sec</i>	This option specifies the interval between service calls when <code>mkvdk</code> is run with the <code>-persist</code> option.

## Managing Collection Disk Space

The following table summarizes the options for managing the disk space used by a collection.

Option	Description
-backup <i>dir</i>	This option backs up the collection into the specified directory.
-purge	This option waits the amount specified by the <code>purgewait</code> option and then deletes all documents in the collection, but not the collection itself; it leaves the collection directory structure intact. To specify a different wait period, use the <code>-purgewait</code> option instead of <code>-purge</code> . If you do not use <code>purgewait</code> , the default is 600 seconds. Note that <code>-purge</code> deletes all documents in a collection, but does not delete the collection itself. To delete a collection, use operating system commands such as the <code>rm</code> command on UNIX to remove the collection directory structure and control files.
-purgeback	This option, used with the <code>-purge</code> option, performs a purge in the background.
-purgewait <i>sec</i>	This option specifies to the <code>-purge</code> option how many seconds to wait. If you do not specify <i>sec</i> , the default is 600.

## Managing System Messages

The following table summarizes the options for controlling the messages generated when you run `mkvdk`.

Option	Description
-quiet	This option displays only fatal and error messages to the console. It overrides the <code>-outlevel</code> setting.
-outlevel ( <i>num</i> )	This option indicates which message types to display to the console. Valid values are determined by adding numbers together that correspond to the desired message types. Fatal=1, Error=2, Warning=4, Status=8, Info=16, Verbose=32, Debug=64, The default value is 15.
-logfile <i>file name</i>	This option saves messages in the specified file.
-loglevel ( <i>num</i> )	This option indicates which message types to route to the optional log file. Valid values are determined by adding numbers together that correspond to the desired message types. The default value is 15.

# Optimization, Modes, and Service Options

This section describes options for optimizing collections, setting indexing modes, and determining service levels provided when running **mkvdk**.

## Optimizing Collections

The following option performs various optimizations on the collection, depending on the value of *spec*:

```
-optimize spec
```

The specifier, *spec*, is a string consisting of keywords separated by hyphens, such as `maxmerge-squeeze-readonly`. The following table summarizes the keywords for the `-optimize` option.

Keyword	Description
maxclean	This keyword performs the most comprehensive housekeeping possible, and removes out-of-date collection files. This optimization is recommended only when you are preparing an isolated collection for publication. Note that when using this type, if the collection is being searched, sometimes files get deleted too early and this affects search results.
maxmerge	This keyword performs maximal merging on the partitions to create partitions that are as large as possible. This creates partitions that can have up to 64000 documents in them.
readonly	This keyword makes the collection read only. When used, <b>mkvdk</b> marks the collection as read-only and unchanging after the function call is done. This is appropriate for CD-ROM collections.
spanword	This keyword creates a spanning word list across all the collection's partitions. A collection consists of numerous smaller units called partitions each of which includes a word list.
squeeze	This keyword squeezes deleted documents from the collection. Squeezing deleted documents recovers space in a collection, and improves search performance. Using this option invalidates the search results.

<b>Keyword</b>	<b>Description</b>
vdbopt	Each collection consists of smaller units called Verity databases (VDBs). The <code>vdbopt</code> keyword configures the collection's VDBs. This keyword has the effect of linearizing the data in a VDB, and making the collection metadata contained in the VDB more streamlined. It also allows the VDB to grow to a much larger size.
tuneup	This keyword is a convenience keyword that includes <code>maxmerge</code> , <code>vdbopt</code> , and <code>spanword</code> .
publish	This keyword is a convenience keyword that includes all of the optimization types. Use this keyword to optimize the collection for the best possible retrieval performance, such as for publication to a network on a server or on a CD-ROM.

## Indexing Modes

The following option controls the way the collection is built, depending on the value of `mode`:

`-mode mode`

If you do not set a mode, the default is `generic`. The following table summarizes the function of each mode.

<b>Mode Name</b>	<b>Description</b>
generic	The generic mode is the default if no mode is specified. It is the base mode from which all other modes inherit their metaparameters. It is optimized to give the best overall performance without assuming anything about the desired indexing rates of documents, how many searches are occurring simultaneously, and so on.
fastsearch	The fast search mode is optimized to index documents so that retrievals happen as quickly as possible. This mode causes the K2 Server engine to do more work at indexing time.
bulkload	The bulk load mode is for indexing large numbers of documents in large batches with bulk insert. It is primarily intended to create new collections from a large number of preexisting documents.
readonly	The read only mode is not an indexing mode, but is useful for accessing a collection on a read-only medium such as CD-ROM.

Mode Name	Description
newsfeedidx	The news feed indexing mode is optimized to accept a large number of documents in a short amount of time where the documents arrive in small batches. It is meant to be able to keep up with the high arrival rates of news feeds without falling behind in the indexing. This mode is different from the bulk load mode in that news feed indexing processes small frequent batches of documents, while the bulk load mode indexes large infrequent batches of documents.
newsfeedopt	The news feed optimizing mode merges partitions created by the news feed indexing mode into large, optimized partitions. This action optimizes an existing collection for fast searches.

## Service Options

The following table summarizes the options that you can use to control the kind of service provided by `mkvdk`.

Option	Description
-repair	This option repairs the collection, performed by an API call.
-noservice	This option prevents collection servicing (servicing includes indexing) by this instance of <code>mkvdk</code> , performed by an API call.
-nooptimize	This option prevents optimization by this instance of <code>mkvdk</code> . Using this option turns off the service level <code>VdkServiceType_Optimize</code> . The service types determine what type of work the K2 Server engine and its self-administration features will execute on a collection.
-nohousekeep	This option prevents housekeeping by this instance of <code>mkvdk</code> . Housekeeping includes deleting files that are no longer needed. Using this option turns off the service level <code>VdkServiceType</code>
-noindex	This option prevents indexing by this instance of <code>mkvdk</code> . Documents will not be inserted or deleted. Using this option turns off the service level <code>VdkServiceType_Index</code> .
-servlev <i>level</i>	Service level. The specifier, <i>level</i> , is a string consisting of keywords separated by hyphens, such as <code>search-index-optimize</code> .

## Service Level Keywords

The following table summarizes the keywords for the `-servlev` option:

<b>Keyword</b>	<b>Description</b>
search	Enable search and retrieval
insert	Enable adding and updating documents
optimize	Enable opportunistic collection optimization
assist	Enable building of word list
housekeep	Enable housekeeping of unneeded files
delete	Enable document deletion
backup	Enable backup
purge	Enable background purging
repair	Enable collection repair
dataprep	Same as <code>search-index-optimize-assist-housekeep</code>
index	Same as <code>insert-delete</code>



# Advanced Features

This section describes advanced options that can be used with **mkvdk**, including the following

- Bulk Submit
- Field Extraction
- Formats, Locales, and Characters Sets

## Bulk Submit

The following table summarizes the options to use with the bulk submit feature for populating field values in the attribute table for a collection.

Option	Description
-bulk	This option tells <b>mkvdk</b> to interpret <i>dockey</i> as a bulk submit file. The option can be used with <code>-insert</code> , <code>-update</code> , and <code>-delete</code> .
-offset <i>num</i>	This option specifies the offset into a bulk submit file or files. Note that if you specify multiple bulk submit files and use the <code>-offset</code> option, the offset is applied to all of the bulk submit files.
-numdocs <i>num</i>	This option specifies the number of documents to insert or delete from the bulk insert file or files. Note that if you specify multiple bulk insert or delete files and use the <code>-numdocs</code> option, the <code>-numdocs</code> setting is applied to all of the bulk insert or delete files.
-autodel	This option deletes the bulk submit file or files when the bulk submission work is finished.

## Field Extraction

The following table summarizes the options to use with the field extraction feature for populating field values in the attribute table for a collection.

Option	Description
-extract	This option extracts field values from documents, using the field extraction rules specified in the <code>style.tde</code> file.

Option	Description
-nosave	Specifies that a work list, which is generated by <b>mkvdk</b> automatically when the <code>-extract</code> option is used, will not be saved in the collection directory in a file called <code>worklist</code> (in the Verity bulk submit file format). By default, <b>mkvdk</b> saves the work list in the <code>worklist</code> file.
-nosubmit	Specifies that a work list, which is generated by <b>mkvdk</b> automatically when the <code>-extract</code> option is used, will not be submitted to the indexing engine and will be saved in the collection directory in a file called <code>worklist</code> (in the Verity bulk submit file format). This option allows <b>mkvdk</b> to process field extraction separately from other indexing tasks.

## Formats, Locales, and Character Sets

The following table summarizes the options.

Option	Description
-charmap <i>name</i>	Names the character set that you would like all strings mapped to for your application. You should set this to name a character set that your system can display properly. In the English version of the search engine, the character set that any version of Windows displays is 8859, the character set that a Mac would display is mac or mac1. Note that this is NOT the name of the character set of documents being indexed, it is only the name of the character set that your display can handle properly. Valid options are 850, 8859, mac. The default is no mapping.
-locale <i>name</i>	Names the language in which you would like to perform searches. The language name corresponds to the name of a directory underneath common that contains language configuration files. The search engine uses these configuration files to know how to perform searches in different languages. Valid options are english, deutsch, and francais. The default is english.
-datefmt <i>format</i>	This option is used to convert a date field value into Verity's internal data representation, and can be used in conjunction with the <b>mkvdk</b> options <code>-extract</code> (for the field extraction feature) and <code>-bulk</code> (for the bulk submit feature). The named format string identifies to the date parsing routines as to what order dates are written in when the date string only consists of a sequence of numbers (for example, 03/03/96). The default is MDY.

## Date Format Keywords

The following table summarizes the keywords supported by the `-datefmt` option.

<b>Format Variable</b>	<b>Description</b>
MDY	Dates written as month-day- year (US format, the default)
DMY	Dates written as day-month-year (European formats)
YMD	Dates written as year-month-day (ISO international format)
YDM	Dates written as year-day-month (Swedish format)
USA	Dates written in US format (the same as MDY)
EUR	Dates written in European format (the same as DMY)



# B

## Using Verity Topics

This appendix describes how to build and use pre-defined queries called topics to create virtual collections. These subjects are covered:

- Using mktopics to Create Virtual Collections
- mktopics Syntax Reference
- Topic Set Limits

# Using mktopics to Create Virtual Collections

The previous sections described how to build collections to ensure consistent and responsive search performance. As explained earlier, to ensure optimum response, collections may be designed to optimize searches over the entire collection system. In cases where users typically search a subset of the document set, you can build topics to create virtual collection, which allow limiting the search to documents from specific sources.

For further information about making topics refer to Appendix B of this guide.

The command-line syntax for the **mktopics** utility is shown below.

```
mktopics -topicset topicset_dir -outline file.otl
To update topic indexes: [ -collection collection|@list]
To indicate topic indexing mode: [-indexType normal|namedOnly]
To reset topics: [-reset]
To run in quiet mode: [-quiet]
To import topic definitions: [-outline file.otl options]
  Import Options:
    To show warnings for undefined topics: [-warnundef]
    To not show warnings for undefined topics: [-nowarnundef]
    To do precedence resolution: [-precres]
    To not do precedence resolution: [-noprecres]
To export topic definitions:
  [-fullotl file.otl [-topic name] [-options]]
  Export Options:
    To create a deep dump: [-deep]
    To create a shallow dump: [-shallow]
To copy output into a log file: [-logfile logfile]
```

# mktopics Syntax Reference

Element	Description
mktopics	The <code>mktopics</code> utility name.
-topicset <i>topicset_dir</i>	The required <code>-topicset</code> argument specifies the name of a new or existing topic set directory, depending on the other mktopics syntax supplied.
-collection <i>collection @list</i>	The optional <code>-collection</code> argument specifies a collection directory or an ASCII file containing a list of collection directories (each on a separate line); the name of such a list file must be preceded by an at-sign. This argument specifies which collection(s) the topic set will be indexed against. When specified, the topic set index is updated in the specified collection directories. Maintaining a topic index in a collection facilitates quick and efficient searches over the collection data when using topics.
-indexType <i>normal namedOnly</i>	The optional <code>-indexType</code> argument specifies the type of topic set index to be built when the topic set is indexed against a collection. Valid values are: <code>normal</code> for indexing topics in the outline file with a precedence rating of incremental or lowest, and <code>namedOnly</code> for indexing only named topics in the outline file. The default is <code>normal</code> . For information about topic precedence ratings, refer to the <i>Verity Introduction to Topics Guide</i> , Chapter 3.
-reset	The optional <code>-reset</code> argument deletes and replaces an existing topic set, rather than updating it.
-quiet	The optional <code>-quiet</code> argument suppresses status messages. By default, <code>mktopics</code> runs in verbose mode.
-outline <i>file.otl</i>	The full or relative path and name of the outline file from which the new topic set will be built. Use <code>.otl</code> as the file name extension.
-fullotl <i>file.otl</i>	The optional <code>-fullotl</code> argument is followed by the full or relative path and name of the file to which you want to export a copy of the topic set. Use <code>.otl</code> as the file name extension.
-topic <i>name</i>	The optional <code>-topic</code> argument is followed by the name of the topic in the specified topic set that you want to export to a topic outline file. This argument must be specified with <code>-fullotl file.otl</code> described above.

Element	Description
<code>-warnundef</code>	The optional <code>-warnundef</code> argument specifies that you will be warned if there are any undefined topics when importing topic definitions from an outline file. Only meaningful when used with <code>-outline</code> . This is the default.
<code>-nowarnundef</code>	The optional <code>-nowarnundef</code> argument specifies that you will not be warned if there are any undefined topics when importing topic definitions from an outline file. Only meaningful when used with <code>-outline</code> . The default is <code>-warnundef</code> .
<code>-precrs</code>	The optional <code>-precrs</code> argument specifies that topic precedence checking will occur when the topic set is built or updated. This argument is the default. Only meaningful when used with <code>-outline</code> .
<code>-noprecrs</code>	The optional <code>-noprecrs</code> argument specifies that topic precedence checking will not occur when the topic set is built. If this argument is set, then topics with precedence errors are rewritten at query time, making the performance of topic searching slow. The default is <code>-precrs</code> . Only meaningful when used with <code>-outline</code> .
<code>-deep</code>	The optional <code>-deep</code> argument specifies that a dump of a topic set to an outline file will dump each top level topic as far down as possible. Only meaningful when used with <code>-fullotl</code> . This is the default.
<code>-shallow</code>	The optional <code>-shallow</code> argument specifies that a dump of a topic set to an outline file will dump each topic down to the next named topic. Only meaningful when used with <code>-fullotl</code> . The default is <code>-deep</code> .
<code>-logfile <i>filename</i></code>	The optional <code>-logfile</code> argument followed by a log file name indicates the a log file will be generated. For the log file name, you can specify the file name and path. If a path is not specified with the file name, the log file is put in the current working directory.



# Topic Set Limits

Although the overall limits on the size of a topic set have been raised to 500,000 nodes and 800,000 links, there are some search time limitations on the size of a single topic. These limits apply to the topic which is built from the query you type in, which may be a combination of query terms and pre-defined topics from a topic set.

Unfortunately, the search time limitations are not simple to describe accurately since they are implementation limitations of various portions of the search engine, rather than a straightforward, uniformly-enforced limit on the physical number of nodes/links.

By way of background, the Verity search engine is built on the notion that topics represent search concepts. Queries that go beyond a single word or phrase typically involve the ACCRUE-class operators (ACCRUE, AND, OR) to combine several branches of evidence in a topic tree. At search time, the combined evidence is evaluated by a stack-based engine.

The stack engine imposes some restrictions for ACCRUE-class topics. Its limited stack space imposes the restriction of 1024 children for any single ACCRUE-class node and about 5,300 total nodes (16000/3 to be precise), in a topic. These limits are detected gracefully while building the query (before running the search), and result in an error. The second limit can be worked around by having named sub-nodes in a large topic and building the topic set with this `mktopics` option:

```
-indextype namedonly
```

The above option causes separate queries to be built for each named sub-topic, leaving more space for each query. Carrying this process to the extreme, however, will reduce the effectiveness of the topic index for the top-level topic.

There are some limits with regards to the use of operators:

- There can be a maximum of 8K children for the ANY operator. If a topic exceeds this limit, an error message is not always reported.
- The NEAR operator will evaluate only 64 children. If a topic exceeds this limit, no message is reported.

Say you have a large topic that uses the ACCRUE operator with 8365 children. This topic exceeds the 1024 limit for any ACCRUE-class topic and the 16000/3 limit for the total number of nodes. In this case, you could not substitute ANY for ACCRUE since that would cause the topic to exceed the 8K limit for the maximum number of children for the ANY operator. To get this large topic functional, you could build a bushier/deeper tree structure, by grouping topics, with some named sub-nodes.



# C

## Date Formats

You can specify both import and export date formats using K2 Toolkit. This appendix lists the variables available for both. These sections are included:

- Export Date Format
- Import Date Format

# Export Date Format

A date export format can be specified per search using the `dateOutputFormat` in `K2SearchNewArgRec`. Date export format constructs can be used to output date field values when dates are formatted for output. Typically, date fields are output to results lists.

Valid values are listed in the following table.

Variable	Description
<code>\${yyyy}</code>	Represents a year as a four-character variable, as in 1996.
<code>\${q}</code>	Represents a quarter as a one-character variable, as in 3. Note that the <code>sq</code> variable assumes that January through March is 1, April through June is 2, July through September is 3, and October through December is 4.
<code>\${mm}</code>	Represents a month as a two-character variable, as in 09.
<code>\${ddd}</code>	Represents a specific day of the year as a three-character variable between 001 and 366, as in 225.
<code>\${dd}</code>	Represents a day of the month as a two-character variable between 01 and 31, as in 29.
<code>\${hh24}</code>	Represents an hour in 24-hour time format as a two-character variable between 00 and 23, as in 15.
<code>\${hh12}</code>	Represents an hour in 12-hour time format as a two-character variable between 01 and 12, as in 08. <code>hh12</code> and <code>hh</code> are interchangeable.
<code>\${hh}</code>	Same as <code>hh12</code> above.
<code>\${mi}</code>	Represents a minute as a two-character variable between 01 and 59, as in 45.
<code>\${ss}</code>	Represents a second as a two-character variable between 01 and 59, as in 23.
<code>\${w}</code>	Represents a day since the previous Sunday as a two-character variable between 00 and 06, as in 01.
<code>\${month}</code>	Represents a month using the full month name, as in september.
<code>\${mon}</code>	Represents a month using a three-character abbreviation, as in aug.
<code>\${day}</code>	Represents a day using the full day name, as in tuesday.
<code>\${dy}</code>	Represents a day using a three-character abbreviation, as in wed.
<code>\${am}</code>	Represents AM or PM, as appropriate.
<code>\${ap}</code>	Represents AM or PM, as appropriate, using a single-character variable, such as a.

## Choosing Case

You can choose to display dates in lower case, with initial capitals, or in upper case, by entering the variable in the desired style, as follows.

<b>Example Variable Case</b>	<b>Example Display Case</b>
<code>\${month}</code>	may
<code>\${Month}</code>	May
<code>\${MONTH}</code>	MAY

## Date Punctuation and Spaces

You can specify date display punctuation and spaces in date format strings, as desired. Date punctuation typically includes commas (,), slashes (/), and dashes (-).

# Import Date Format

The Verity engine can parse a variety of date formats. A single date format string can include a calendar format plus an optional time format. The import date format specifies the date format users must enter in field searches on a search form. This format is specified in the `inputDateFormat` configuration parameter in the `k2server.ini` file. For example:

```
inputDateFormat = DMY
```

If there is no specified value for `inputDateFormat`, the K2 engine interprets the numbers in MDY format, where M represents a two-digit month, D represents a two-digit day, and Y represents a two-digit year.

## Elements

The following key tables list the valid calendar and time elements which can be used in the supported date formats.

### Calendar Elements

Calendar Format Element	Description
MM	Represents a one or two-digit numeric month.
Mon	Represents an alphabetic month, 3 or more characters in length.
DD	Represents a one or two-digit numeric day of the month.
YY	Represents a two or four-digit numeric year.
<i>time</i>	Represents a time format, as described in the following section.

### Time Elements

K2 Server understands time formats that have one of the following structures:

```
HH:MI
```

```
HH:MI:SS
```

```
HH:MI:SS AM|PM
```

```
HH:MI:SS AM|PM TIMEZONE
```

A definition for each of the time format elements in the above structures is provided below.

Time Format Element	Description
HH	This element represents the hours.
MI	This element represents the minutes.
SS	This optional element represents the seconds.
AM	This optional element represents the AM hours.
PM	This optional element represents PM hours.
TIMEZONE	This optional element represents a time zone.

## Numeric-only Date Formats

Supported date formats composed entirely of numeric elements are listed in the following table.

The date format constructs described below resolve ambiguities in numerical-only date expressions. By default, dates input into the documents table are assumed to be in American numeric date format, MM-DD-YY. This means that if a user enters a date for a field search query in the same format, the Verity engine can interpret the date and perform the appropriate retrieval. Numeric date formats can be delimited by spaces or slashes in addition to dashes.

If users want to enter date field search criteria in a different format, such as English or European numeric date format, then you must specify which date format to use when interpreting date formats of the form XX-YY-ZZ. Specify one of the following date formats for the `importDateFormat` parameter in the `k2server.ini` configuration file.

Date Format	Examples	Description
MDY	5 17 96 05-17-96 05/17/1996	Month-day-year, American numeric date format. This is the default.
DMY	17 05 96 17-5-96 17/05/1996	Day-month-year, English numeric date format
YMD	96 05 17 96-5-17 1996/5/17	Year-month-day, European numeric date format
YDM	96 17 05 96-17-5 1996/17/5	Year-day-month, Swedish numeric date format

## Alphanumeric Date Formats

Supported date formats composed of both numeric and alphanumeric elements are listed in the following table.

Date Format	Examples	Description
DD Mon YY	17 Feb 96 17 February 1996	Day (numeric), month (alphabetic), year (numeric)
DD Mon YY <i>time</i>	17 Feb 96 23:59 17 February 96 01:50	Day (numeric), month (alphabetic), year (numeric), <i>time</i> . See “Time Elements” above for valid time element formats.
Mon DD YY	Feb 17 96 February 17 1996	Month (alphabetic), day (numeric), year (numeric)
Mon YY	Mar 96 Jan 97	Month (alphabetic), year (numeric)
MM YY	02 96 12 97	Month (numeric), year (numeric)
YYDDDD	96364 20001	Julian Date format
HH Min DD MM YY	23 59 25 12 91 00 00 01 01 32	Dow Jones date format
DDHHMMZ Mon YY	252312Z JAN 94 310101Z JAN 94	Zulu date format

## Special Date Formats

Supported date formats which are special in format, applicable situations or both are listed in the following table.

Date Format	Examples	Description
YYDDDD	96364 20001	Julian date format
HH Min DD MM YY	23 59 25 12 91 00 00 01 01 32	Dow Jones date format
DDHHMinZ Mon YY	252312Z JAN 94 310101Z JAN 94	Zulu date format



## **Time Formats with the Zulu Date Format**

K2 Server assumes that the time in Zulu date format is in Greenwich Mean Time (GMT). If you use a different time format when you enter search criteria, local time is assumed. Local time depends on the time and time zone settings of your operating system.

Thus, if you enter the following date as the DATE field value for a document:

```
252312Z JAN 94
```

and your computer is set to Pacific Standard Time (PST), a Verity client finds this document if you query the following DATE field value:

```
Jan 25 94 15:12
```

This is because PST is 8 hours behind GMT.

## Import Date Format

# Index

## A

- about A-4
- absolute path
  - mkvdk** A-2
- accessProfile 3-5
- assist A-10
- attribute table 4-3
- attributes
  - collections 4-3
- autodel A-11

## B

- background information vii
- backup A-6
- backup A-10
- basic operations
  - mkvdk** A-4
- broker() 3-4
- browse.exe** 4-3
- bulk A-11
- bulk insert and delete options
  - mkvdk** A-11
- bulkload A-8

## C

- character sets **mkvdk** A-12
- charMap 3-5, 3-7
- charmap A-12
- client/server overview 1-1
- collAlias 3-6

- collection A-4
  - B-3
- collection
  - disk space, managing A-6
  - partitions 4-2
- collection disk space
  - mkvdk** A-6
- collection maintenance A-1
- collection management 2-5
- collection redundancy 1-10
- collection sections
  - K2 Server 3-6
- collection state 2-5
- collections
  - attributes 4-3
  - distributing 2-6
  - document path names in A-2
  - fields 4-3
  - how they work 4-2
  - indexes for documents from different sources, illustration 2-7
  - merging 4-9
  - new creating with **mkvdk** A-5
  - optimizing, **mkvdk** A-7
  - segmented 2-8
  - splitting 4-9
  - using with K2 Server 2-6
  - virtual, creating with **mktopics** B-2
- collPath 3-6
- command line
  - mktopics** utility B-2
- common A-4
- configuration file
  - K2 Server 3-3
- configuration overview
  - K2 Server 2-2
- connTimeout 3-12, 3-13
- content summary viii
- conventions
  - typeface x
- create A-5

**D**

- datapath A-4
- dataprep A-10
- date formats 13
  - export 8
  - import 10
  - mkvdk** A-13
  - Zulu date format 13
- datefmt A-12
- Debug
  - log level 4-8
- debug A-4
- deep B-4
- delete A-5
- delete A-10
- description A-5
- didump.exe** 4-4
- directory structure
  - illustration 4-2
- distributing collections 2-6
- document key 2-5
- document management 2-5
- document path names
  - in collections A-2

**E**

- Error
  - log level 4-8
- extract A-11

**F**

- fastsearch A-8
- Fatal
  - log level 4-8
- features
  - K2 Server and K2 Client API 1-1
- field extraction
  - mkvdk** A-11

**fields**

- collections 4-3
- permanent 4-4
- transitory 4-4

**formats**

- mkvdk** A-12
- fullopt1 B-3

**G**

- generic A-8

**H**

- help A-4
- housekeep A-10

**I****illustration**

- brokered system 1-8
- collections with indexes for documents
  - from different sources 2-7
- configuration overview, K2 Server 2-2
- directory structure of typical collection 4-2
- indexing application interaction with
  - Verity search engine 4-5
- K2 Broker 1-6, 3-9
- K2 Server configuration overview 2-2
- K2 Server TCP acceptor 2-2
- K2 servers and brokers 1-8
- load balancing 2-6
- log file format 4-7
- segmenting indexes 2-8
- incremental squeeze 4-11
- index A-10
- indexes 1-4
- indexing application
  - interaction with Verity search engine,
    - illustration 4-5
  - mkvdk** 4-7

## indexing modes

**mkvdk** A-8

## indexing process 4-5

`-indexType` normal B-3

## Info

log level 4-8

`-iniEmit` 3-3, 3-9`-iniFile` 3-2, 3-9`inputDateFormat` 3-8`-insert` A-5

insert A-10

**K**

## K2

client/server overview 1-1

Server reference 2-1

## K2 Broker 2-5

illustration 2-5, 3-9

illustration, concurrent client

connections 3-10

node section 3-13

reference 3-1

starting 3-9

## K2 collection state 2-5

## K2 document key 2-5

## K2 Server

collection sections 3-6

configuration file 3-3

configuration keywords 3-3

configuration overview 2-2

keywords 3-3

reference 2-1

starting 3-2

TCP acceptor, illustration 2-2

using collections with 2-6

## keywords

Server 3-3

knowledgeBase 3-5, 3-7

**L**

listeners 2-3

## illustration

K2 Broker, concurrent client

connections 3-10

## load balancing

illustration 2-6

`-locale` A-12

locale 3-5, 3-8

## locales

components 7-3

custom 7-2

from Verity partners 7-2

installation 7-6

language support 7-4

**mkvdk** A-12

predefined 7-2

upgrading 7-4

using `THESAURUS` operator 7-5

## log file format

illustration 4-7

`-logfile` A-6, B-4`-loglevel` A-6**M**`maxclean` A-7`maxColSize` 3-4, 3-7`maxFiles` 3-4, 3-7`maxmerge` A-7**merge** utility 4-9

merging collections 4-9

**mkbulk** 2-8**mktopics**

creating virtual collections B-2

**mktopics utility**

- command line B-2
- suppressing status messages B-3
- syntax B-2
- syntax defined, -fullotl B-3
- syntax defined, -outline B-3
- syntax defined, -quiet B-3
- syntax defined, -reset B-3
- syntax defined, -topic B-3
- syntax usage, -topicset B-2

**mkvdk utility 4-7**

- advanced features A-11
- basic operations A-4
- basic syntax A-3
- bulk insert and delete options A-11
- creating new collections A-5
- date format options A-13
- default behavior A-2
- field extraction A-11
- formats, locales, character sets A-12
- general options A-4
- indexing modes A-8
- managing collection disk space A-6
- managing system messages A-6
- optimization keywords options A-7
- optimizing collections A-7
- reference A-1
- service level keywords A-10
- service options A-9
- servlev A-10

multitiered, brokered system 1-8

**N**

- namedOnly B-3
- new collections, creating with **mkvdk** A-5
- newsfeedidx A-9
- newsfeedopt A-9
- node section
  - K2 Broker 3-13
- nodeSpec 3-13
- noexit A-4

- nohousekeep A-9
- noindex A-9
- nolock A-4
- nooptimize A-9
- noprecres B-4
- nosave A-12
- noservice A-9
- nosubmit A-12
- nowarnundef B-4
- ntService 3-2, 3-3, 3-9
- numdocs A-11
- numListeners 3-4, 3-12
- numThreads 3-3, 3-7, 3-12, 3-13

**O**

- offset A-11
- onLine 3-7, 3-13
- optimization keywords options
  - mkvdk** A-7
- optimize A-10
- options
  - mkvdk** A-4
- outlevel A-6
- outline B-3

**P**

- partitions 4-2
- permanent fields 4-4
- persist A-5
- ping communications 1-10
- port 3-2, 3-9
- portNo 3-4, 3-11
- precres B-4
- publish A-8
- purge A-6
- purge A-10
- purgeback A-6
- purgewait A-6

**Q**

-quiet A-6  
Font>-quiet B-3

**R**

**rck2** Utility 3-16  
readonly A-7, A-8  
relative path  
    **mkvdk** A-2  
-repair A-9  
repair A-10  
-reset B-3  
resultCacheEnabled 3-6, 3-12  
resultCacheMaxInBytes 3-6, 3-12  
resultCacheQuota 3-6, 3-12  
resultCacheTimeout 3-5, 3-12

**S**

search A-10  
search engine 1-2, 4-5  
search performance  
    optimizing 2-3  
segmented collections 2-8  
segmenting indexes  
    illustration 2-8  
Server  
    reference 2-1  
Server reference 2-1  
Server section 3-3  
serverAlias 3-3  
service level keywords  
    **mkvdk** A-10  
service options  
    **mkvdk** A-9  
-servlev A-9  
servlev  
    **mkvdk** A-10  
Setting Up K2service.ini file  
    K2 Server 3-3

-shallow B-4  
-sleeptime A-5  
sortTruncDocs 3-5  
spanword A-7  
splitting collections 4-9  
squeeze A-7  
Status  
    log level 4-8  
style A-5  
style.ddd 4-3  
style.sfl 4-3  
style.ufl 4-3  
-synch A-4  
syntax  
    **mktopics** utility B-2  
    **mkvdk** A-3  
system messages  
    **mkvdk** A-6  
system messages, managing A-6

**T**

time formats 10  
-topic B-3  
topic set  
    exporting to outline file B-3  
    size limits B-5  
topics  
    export definition to outline file B-3  
    size limits B-5  
topics, support for 1-11  
topicSet 3-5, 3-7  
-topicset  
    B-3  
topicset A-5  
transitory fields 4-4  
typeface conventions x

**U**

-update A-5  
URL, Verity's Web site vii

## V

- vdkHome 3-5, 3-11
- vdkSortingFlag 3-5, 3-12, 3-14
- Verbose
  - log level 4-8
  - verbose 3-9
- Verity
  - Web site vii
- Verity collections 1-4
- Verity Locales 1-4, 7-1
- Verity Locales Using LinguistX
  - installing 7-6
  - languages 7-4
- Verity Query Language
  - using English for locales other than English 7-10
- Verity query language
  - localized 7-8
- Verity search engine 1-2
- virtual collections
  - creating with **mktopics** B-2

## W

- Warning
  - log level 4-8
  - warnundef B-4
- Web site, Verity's vii
- word index 4-4
- words A-5

## X

- XML documents 6-1
- XML filter 6-1
- XML support 6-1