

eVision

eVe 3 Professional Getting Started Guide



eVision eVe 3 Getting Started Guide

Printed: 2001

Publication Number: EVE-300-APIGS-00

Information in this guide is subject to change without notice and does not constitute a commitment on the part of eVision LLC. It is supplied on an "as is" basis without any warranty of any kind, either explicit or implied. Information may be changed or updated in this guide at any time.

Mailing Address

eVision LLC
1 South 450 Summit Ave., Suite 210
Oakbrook Terrace, IL
60181

Table of Contents



Preface

1 • Introduction

Overview	1-2
What is a Visual Search Engine?	1-2
About the eVision Technology	1-3
What is eVe?	1-4
How do I use eVe with an Application?	1-5
The Basics	1-6
Digital Images	1-6
Processing and Using Images with eVe	1-7
Preparing Images for Analysis	1-8
Analyzing and Comparing Images	1-8
Storing Image Information	1-10
Searching for and Retrieving Images	1-11
Using the Sample Application	1-13
Overview	1-13
Getting Started	1-13
Initiating Searches	1-14
Performing Searches	1-15
Adding Categories and Images	1-19
Creating a Visual Vocabulary	1-22
Window Description	1-24

2 • Integrating eVe into an Application

Overview	2-2
The Basics	2-2
Determine Requirements	2-2
Analyze and Design	2-4
Implement Design	2-6
eVe Demo Implementation	2-7
Overview	2-7
eVe Interfaces Used in eVe Demo	2-9
How the eVe Demo Works	2-10

3 • Code Samples

Analyzing a File3-2

Creating a MediaCollection3-3

Performing a Search3-4

Refining a Search3-6

Using EveContext3-7

Searching Multiple Collections3-8

Searching with Metadata and Images3-9

Performing Partial Image Selection3-10

 Overview3-10

 Code Samples3-11

Determining the Distance Between Images3-13

EveExceptions3-14

Using Before and After Command Methods3-15

Index



Preface

Company Positioning

Locating images is not an easy task, and when there are many files to search through, sometimes the only way to search through them is visually. Now the remarkable eVision technology can do just that — match images visually.

eVision's solution revolutionizes the visual search experience by giving customers direct access to the information within images. eVe (eVision Visual engine) is an advanced Visual Search engine that includes analysis, storage, indexing, and search/retrieval of images. Unlike a classical keyword-based search, eVision software retrieves images by analyzing their perceptual content. *Images do not need to be viewed or interpreted and keyworded by people beforehand.*

Contacting eVision

Corporate Headquarters

eVision
1 South 450 Summit Ave
Suite 210
Oakbrook Terrace, IL 60181

- Tel: 630.932.8920
- Fax: 630.932.8936
- Email: info@evisionglobal.com

For technical support

eVision discussion boards or eVision newsgroup

For press and media relations

Email: media@evisionglobal.com

For business development

Email: bizdev@evisionglobal.com

For investor relations

Email: invest@evisionglobal.com

For careers opportunities

Email: careers@evisionglobal.com

About This Guide

This guide assumes that the appropriate eVe 3 Professional components have been installed at your site. The instructions for installing the product are in the Installation Guide.

Ch. No.	Chapter Name	Content Description
1	<i>Introduction</i>	Introduces you to conceptual information about digital images, as well as about content-based retrieval. You should be familiar with all the concepts in this chapter before you start programming with eVe.
2	<i>Integrating eVe into an Application</i>	Discusses how to integrate eVe within an application.
3	<i>Code Samples</i>	Contains code samples and reference code designed to help you get started using eVe as quickly as possible.

What's New in eVe 3 Professional?

The following features are new to eVe 3 Professional:

New Feature	Description
Video Key Frame Extraction	<p>eVe 3 Professional lets you perform visual search on analog and digital video as well as images and graphics. The engine is able to "watch" video and index a keyframe just as if it were any other still image. You can then search for videos with keyframes that match a query image. You can also use this to create storyboard representations or automatically annotate the video for later navigation.</p> <p>See the <i>eVe Low-Level API</i> chapter in the <i>API Reference Guide</i> for more information about the new FrameGrabber interface and video key frame extraction.</p>
Support for IRIX OS	<p>eVe 3 Professional supports eVe installations under the following operating systems: Win32, Linux, Solaris, Mac OS X, and IRIX.</p> <p>See the <i>System Requirements</i> section in the <i>Installation Guide</i> for more information.</p>
Enhanced access to the metadata fields within a MediaCollection	<p>eVe 3 Professional will give you the ability to define and query the names of the Keys (categories) that contain metadata. You can also query the values associated with those Keys, and all of the images that have those values.</p> <p>For example, you can find that one Key is "Color", and ask for the values under "Color". You might get back "Blue", "Red", and "Brown". Then you can get all of the names of the images that have a value of "Red".</p> <p>See the <i>Low-Level API</i> chapter in the <i>API Reference Guide</i> for more information about the new <code>getMetaDataValues</code> method.</p>
Support for XML	<p>eVe 3 Professional now accepts commands written in XML to perform visual search functions.</p>
Support for Oracle and SQLserver for indexing	<p>eVe 3 Professional allows the indexes of the MediaCollection to be stored in Oracle and SQL Server, as well as in Object Store and flat files.</p>

Conventions

Some or all of the following conventions appear in this guide:

Symbol or Type Style	Represents	Example
Bold	what a user presses (either a key on the keyboard or a button on the screen)	...press EnterClick Modify .
	what a user types	type RUN APP.EXE in the Application field
<i>Alternate color</i>	hotlinked cross-references to other sections in this guide; if you are viewing this guide online in PDF format, you can click the cross-reference to jump directly to its location	...see <i>Chapter 3, eVe High-Level API</i> .
<i>Italic</i>	words that are emphasized	...the entry <i>after</i> the current entry...
	the titles of other documents	<i>eVe Installation and Getting Started Guide</i>
	syntax variables	COPY <i>filename</i>
Monospace	directories, file names, syntax, SQL	&HIGHLVL.SRCLIB
	screen text, system responses, command line commands	Copy file? Y/N
▶	choosing a command from a cascading menu	File ▶ Import ▶ Object

Related Publications

As you use this *eVe 3 Professional Getting Started Guide*, you might find it helpful to have these additional books available for reference:

- *eVe 3 Professional Installation Guide*
- *eVe 3 Professional API Reference Guide*
- *eVe General FAQ*
<http://www.evisionglobal.com/tech/faq.html>
- *eVe Getting Started as a Developer FAQ*
http://www.evisionglobal.com/developers/faq/developer_faq.html
- *eVe Technical FAQ and TroubleShooting Guide*
http://www.evisionglobal.com/developers/faq/technical_faq.html



1

Introduction

This chapter introduces you to conceptual information about digital images and content-based retrieval, as well as how to use the eVe sample application to perform visual searches. You should be familiar with all the concepts in this chapter before you start programming with eVe.

Overview	1-2
What is a Visual Search Engine?	1-2
About the eVision Technology	1-3
What is eVe?	1-4
How do I use eVe with an Application?	1-5
The Basics	1-6
Digital Images	1-6
Processing and Using Images with eVe	1-7
Preparing Images for Analysis	1-8
Analyzing and Comparing Images	1-8
Storing Image Information	1-10
Searching for and Retrieving Images	1-11
Using the Sample Application	1-13
Overview	1-13
Getting Started	1-13
Initiating Searches	1-14
Performing Searches	1-15
Adding Categories and Images	1-19
Creating a Visual Vocabulary	1-22
Window Description	1-24

Overview

Locating specific images within a database is not an easy task. Especially when those images have non-standard file names or inadequate text tags. Often, the best and most natural way to locate these images is visually.

What is a Visual Search Engine?

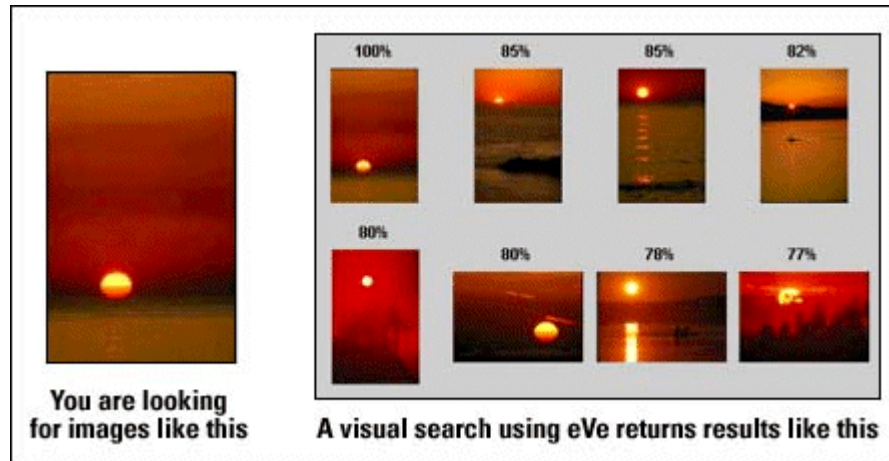
A visual search engine enables users to search and find visual information contained in images, graphics, videos, etc. There are two approaches to retrieving visual information: text-based and content-based. Text-based search engines rely on manual annotation of the images and videos. Content-based or “Visual Search” engines rely on visual properties such as color, texture, shape, etc. These properties are automatically extracted from the images and videos based on image processing and pattern recognition techniques.

Using a text-based search engine to search the Internet or a database often produces a large number of inaccurate or irrelevant results. For example, a recent search on Altavista.com for "Angels" turned up links to Web sites for Charlie's Angels, the Blue Angels, Angel perfume, the Guardian Angels, the neurogenetic disorder Angelman's syndrome, Vanessa Angel, a schedule for the Anaheim Angels baseball team Angel fish, and references to the biblical Angel.



Text-based search engines or Digital Asset Management (DAM) software offer image searches that comb the data based on the text file names attached to images. If the file name for an image is not representative of that image, or if a user does not know what words to input, a text-based search is inadequate and can be frustrating.

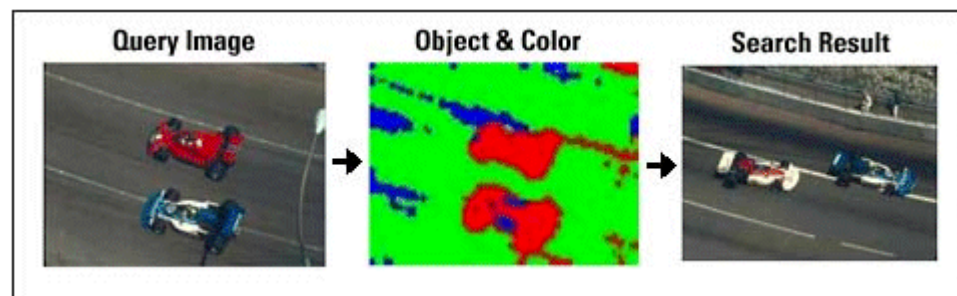
In contrast to text-based searching, a visual search looks inside the actual image and uses mathematical algorithms that analyze color, shape, and texture to help produce more accurate results. Users can select a query image, such as the image of a sunset, and then "Zero In" their search to find similar pictures, either by color, shape, texture or object.



About the eVision Technology

eVision's visual object recognition and extraction technology breaks images down into their key visual elements to generate and index visual signatures.

This technology distills images into their representative visual characteristics based on colors, shapes, textures, and object regions. The algorithms employed segment an image into distinct object regions, then generate scale-independent descriptions of those regions, known as *visual signatures*. These visual signatures are organized in a proprietary indexing scheme for extremely fast retrievals.



What is eVe?



Why is eVe unique?

eVe's uniqueness comes from its ability to automatically segment an image into relevant object regions and generate signatures that capture the color, texture, shape, and object patterns. The ability to segment enables whole and partial image searches with unparalleled search accuracy.

eVe™ (eVision Visual Engine) is a content-based visual information retrieval engine that can be seamlessly integrated into and extend the functionality of text-based visual search engines. It is a true content-based search product that adds visual dimension to text. eVe combines an intelligent scalable infrastructure to acquire, process, and manage visual assets with the ability to develop intuitive user interfaces that work the way people think.

Using eVe, creators and users of media can enable their customers to rapidly find the desired target products (for an e-Business) or assets (for media asset management) using visual media and fewer clicks. When eVe is integrated within an application, a user of that application simply chooses an example query image and starts searching your databases. eVe can search all your media databases and return a set of sorted images and videos based on visual similarity and relevancy to the original query image.

What is the eVe SDK?

The eVe Software Developers Kit (SDK) is a programming toolkit for building image analysis and visual search applications for the recognition and retrieval of images. The components include Java, C++, and COM class libraries, sample programs with source code, sample images, and reference documentation.

Developers and software engineers can use eVe as a building block for developing a visual search application—it is not an end-user application in itself. eVe provides an intelligent infrastructure to acquire, process, and manage visual assets while assisting the development of natural and intuitive user interfaces that perform searches in the same way that people think.

Some example applications using eVe are:

- **Online shopping.** Locating a specific product within an e-Commerce site can sometimes be a difficult task. For example, to locate a particular type of shirt on a clothing site, a user must first specify the type, purpose, color, and material of that shirt using text-based prompts and menu listings. After clicking through and choosing options from these numerous prompts and menus, the user is then presented with a number of unsorted shirt images from which to choose. Utilizing eVe's Visual Search technology, you can enable a user to find the desired shirt using a more visually intuitive interface that requires fewer clicks and produces faster results.
- **Medical imaging.** Medical imaging is critical to patient diagnosis and care across a broad range of health care procedures and disease states. For example, to locate radiology images from different patients who share common symptoms is difficult, if not impossible. Utilizing eVe's Visual Search technology, you can enable a medical professional to quickly find visually similar radiology images from different patients to help with diagnosis.
- **Stock photograph collections** (for example, for fashion designers or architects). Using eVe, you can enable:
 - fashion designers to quickly locate similar clothing designs, color usage, etc.
 - architects to locate matching structures, blueprints, schematics, etc.

How do I use eVe with an Application?

See [Chapter 2, Integrating eVe into an Application](#) for detailed information about how to integrate eVe within an application.

eVe analyzes, stores, and indexes digital media, and enables your application to quickly and accurately search and retrieve that media. The eVe media engine provides, or provides access to: image storage, content-based retrieval, and format-conversion capabilities. The engine itself does not store images; rather, it allows you to access images stored in databases and flat files.

The following illustration shows how eVe fits within an application's framework.

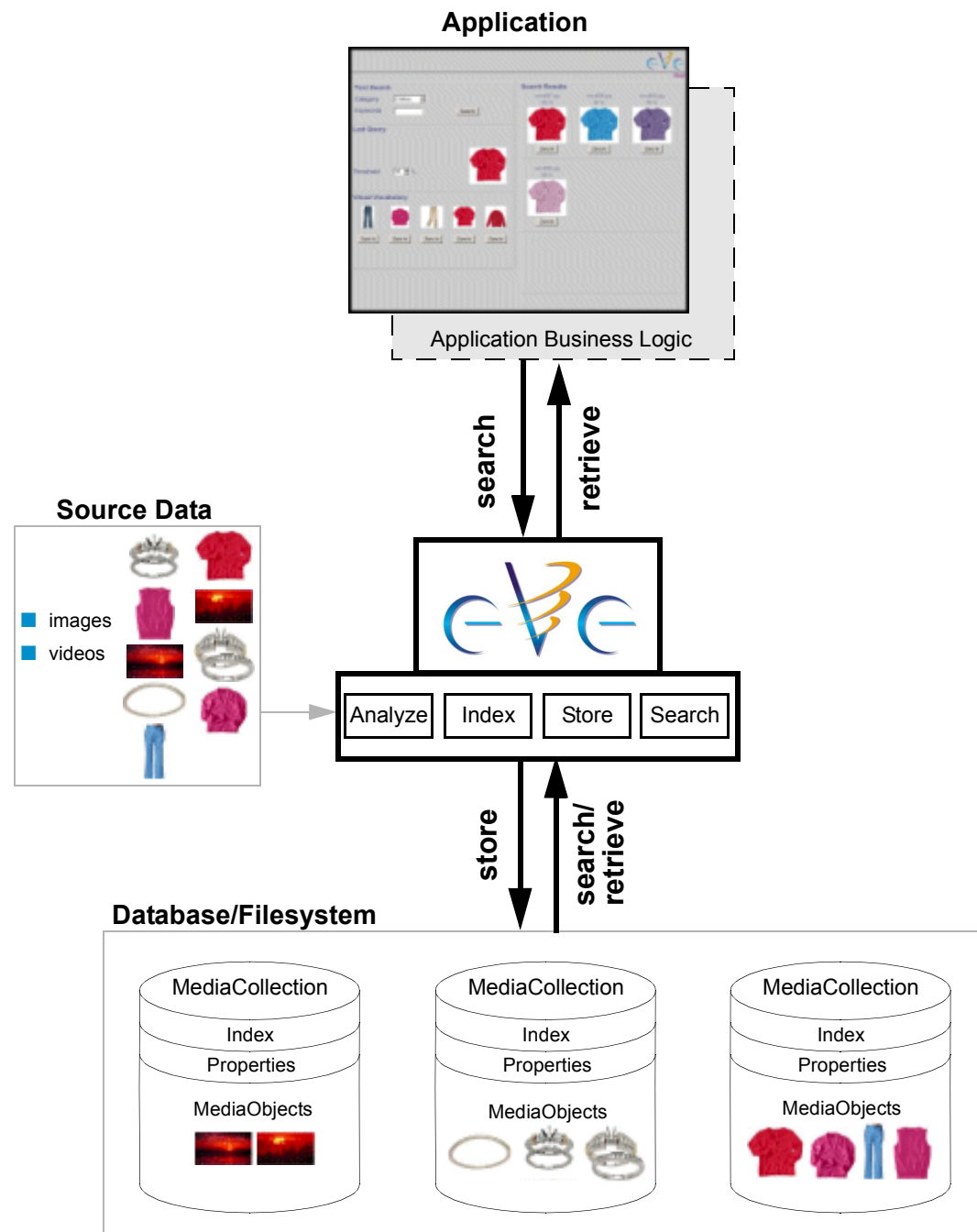


Figure 1-1 • How eVe fits within an Application Framework

The Basics

This section contains conceptual information about digital images, content-based retrieval, and how eVe works. You should be familiar with all the concepts in this section before you start programming with eVe.

Digital Images

eVe's technology implementation supports two-dimensional, static, digital images. These images can be still photographs or video keyframes. They can also be bitonal (black and white) images, grayscale photographs, or color photographic images.

The source of the images is unimportant: they might be produced by a scanner, a digital camera, a video recorder connected to a digitizer or frame grabber, or even generated by a program. A digital image capture device converts an analog signal, such as light in a traditional camera, into discrete digital values on a two-dimensional grid of pixels.

eVe supports the following file formats: JPEG, GIF, and PNG. If the images you want to analyze with eVe are a different file type than these supported types, you must convert them to one of these types. For example, if your image library consists of BMP files, you might convert them to the GIF format before you run them through analysis.

To help you with any image conversions, the eVe SDK includes a third-party program called ImageMagick™. This program allows you to resize, rotate, sharpen, color, reduce, and convert images. ImageMagick is available on the eVe installation CD in the following locations:

- **Windows** — \extras\im\ImageMagick-win2k.zip
- **Solaris 2.7** — \extras\im\ImageMagick-sparc-sun-solaris2.7.tar.gz
- **Solaris 2.8** — \extras\im\ImageMagick-sparc-sun-solaris2.8.tar.gz
- **IRIX** — \extras\im\ImageMagick-mips-sgi-irix6.5.tar.gz
- **Linux** — \extras\im\ImageMagick-i686-pc-linux-gnu.tar.gz

Processing and Using Images with eVe

Processing and using digital images within eVe consists of the following four steps:

- 1 *Preparing Images for Analysis*. Before starting and using eVe, you must first prepare the images upon which you want to perform visual searches.
- 2 *Analyzing and Comparing Images*. Before you can use eVe to search and retrieve an image from a database or flat file, you must analyze that source image. During analysis, eVe segments an image into distinct object regions, then generates scale-independent descriptions of those regions, known as visual signatures.
- 3 *Storing Image Information*. eVe stores information about images in MediaObjects and MediaCollections. A MediaObject contains a single image and all the information related to that image. A MediaCollection is a container for multiple MediaObjects.
- 4 *Indexing Images*. After visual signatures for images are generated, they are organized in a proprietary indexing scheme for extremely fast retrievals.
- 5 *Searching for and Retrieving Images*. After analyzing, storing, and indexing images, use eVe's comprehensive visual search types to find images quickly and intuitively.

Using the eVe API to Work with Images

The following table lists the high- and low-level interface parts involved in each step.:

Function	High-Level API	Low-level API
Analyze	MediaCollectionHL.addFolder()	Analyze.analyze()
	MediaCollectionHL.addFolder(MediaCollection)	
	MediaCollectionHL.addImage()	
	MediaCollectionHL.addImage(MediaCollection)	
Store	MediaCollectionHL.addFolder()	MediaCollection.add()
	MediaCollectionHL.addFolder(MediaCollection)	MediaObject.loadImage()
	MediaCollectionHL.addImage()	
	MediaCollectionHL.addImage(MediaCollection)	
Index	MediaCollectionHL.addFolder()	MediaCollection.reorganize()
	MediaCollectionHL.addImage()	
Search and Retrieve	MediaCollectionHL.search()	MediaCollection.metadataFind()
	SearchResultsHL	MediaCollection.search()

See the *API Reference Guide* for more information about the eVe SDK's low- and high-level APIs.

Preparing Images for Analysis

Before starting and using eVe, you must first prepare the images upon which you want to perform visual searches. Preparing images consists of the following steps:

- 1 Convert images to GIF, JPEG, or PNG format.** The eVe analysis engine supports the following image types: GIF, JPEG, and PNG. If the images you want to analyze with eVe are stored as a different file type than these supported types, you must convert them to one of these types. To help you with any image conversions, the eVe SDK includes a third-party program called ImageMagick™. This program allows you to resize, rotate, sharpen, color, reduce, and convert images.
- 2 Resize images (optional).** Before an image is run through the eVe analysis engine, that image will be automatically resized (if needed) to match the value set in the `maxResolution` parameter of the `eve.properties` file. However, if you first used ImageMagick to convert an image to a GIF, JPEG, or PNG format (see the previous section), we recommend you use ImageMagick to perform the resize, rather than allowing the automatic resize mechanism to perform the resize.

Performing these steps before analysis helps ensure a successful analysis and improved visual search results. For more information about how to prepare images for analysis, see the *Additional Configuration* section in the *eVe Installation Guide* for more information.

Analyzing and Comparing Images

When you search for images, you can compare them based on any combination of their properties: color, shape, texture, and object. For example, if you had a collection of handbag images, you could retrieve all the images that contain red bags, bags of a particular shape, plaid-patterned bags of a certain shape, etc.

Image Components

In the eVe system, images have four distinct attributes: *color*, *shape*, *texture*, and *object region*. Each of these attributes is used in the corresponding search type. Note that image size does not factor into the attributes. Images in eVe are scale-independent to ensure more accurate comparisons.

Visual Signatures

The eVe engine automatically segments an image into distinct object regions, then generates scale-independent descriptions of those regions, known as visual signatures. An image's visual signature is a double array vector composed of the calculated values for its four basic properties of color, shape, texture, and object.

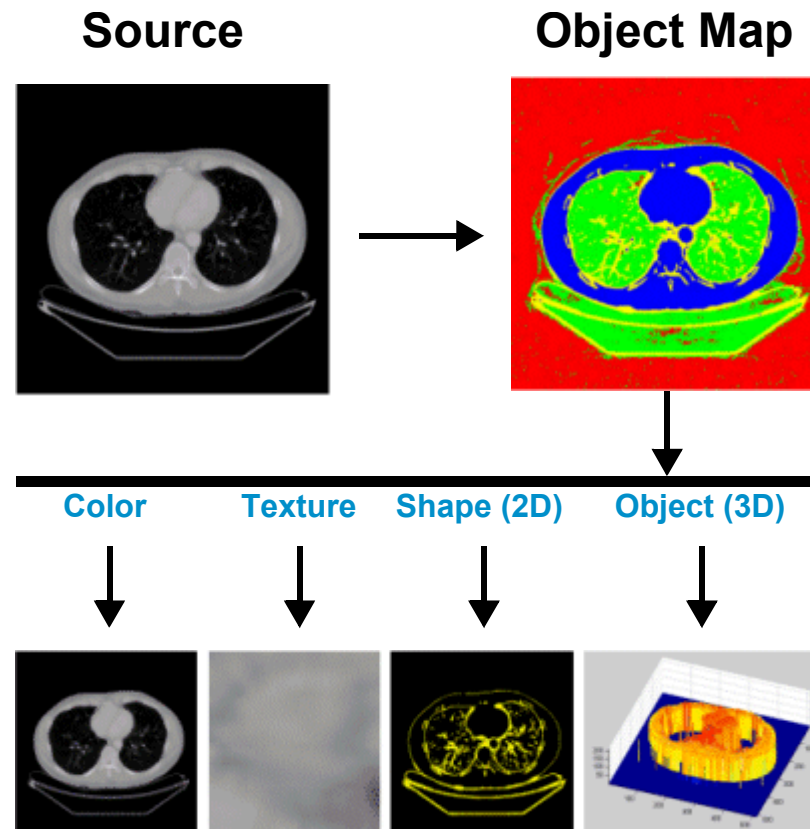


Figure 1-2 • Creating a Visual Signature

These visual signatures are organized in a proprietary indexing scheme for extremely fast retrievals. A comparison of images is done by comparing the visual signatures.

To generate a visual signature for an image using the eVe API

- 1 Prepare images for analysis (see the *Additional Configuration* section in the *eVe Installation Guide* for more information).
- 2 Instantiate a `MediaObject`.
- 3 Load an image into the `MediaObject`.
- 4 Pass the `MediaObject` to the `analyze ()` method within the `analyze` class.

See the *Analyzing a File* section in the *Code Samples* chapter for an example of how to analyze an image. See the *eVe SDK API Reference Guide* for more information about the `analyze ()` method.

Attribute Weighting and Ranking

When you perform a search, you can specify the relative importance of each of the four basic properties. You can also specify the order in which you want eVe to return the results of your search. For example, if you were searching for pictures of a particular car, you could weight the shape and region properties heavily, and perhaps not include color at all.

Storing Image Information

eVe stores two types of information about an image: its visual signature and any textual metadata associated with the image. Together, this information constitutes a *MediaObject*.

- **MediaObjects.** A *MediaObject* contains a single image and all the information related to that image. This information includes:

- image properties such as height, width, size, etc.
- signatures that capture the visual content (color, texture, shape, object)
- segmentation map
- thumbnail of the original image (original image itself if it is smaller than 128x128)
- path to the location of the original image
- date and time when the image was analyzed
- statistics related to the analysis process
- Metadata

MediaObjects can be stored as EDF (Eve Data Format: eVe's representation of an image, already broken down into signatures) files on disk or in a database.

- **MediaCollections.** The sample UI included with the eVe SDK refers to *MediaCollections* as *Categories*. A *MediaCollection* is a container for multiple *MediaObjects*. Searching operates on individual *MediaCollections*. *MediaCollections* are stored in either a database or in a filesystem. *MediaCollections* contain four types of information:

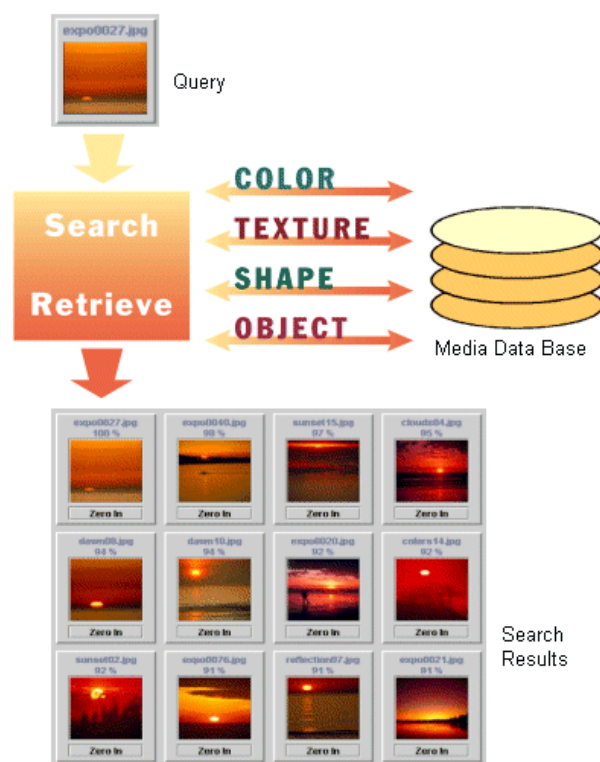
- *Indexes* are always stored on disk.
- *MediaCollection properties* are always stored on disk. Currently, the only property is the collection name, but a property can be any serializable object.
- *MediaObjects* can be stored on disk or in the database, but you cannot “mix and match” the locations: all *MediaObjects* in the *MediaCollection* must be either in the database or on disk.
- *MediaObject metadata* can be file-based or contained in the database, but as with *MediaObjects*, you cannot “mix and match” the locations.

Searching for and Retrieving Images

You can perform visual searches, metadata searches, or a combination of the two. For example, you might start with a text (metadata) search to find images in a certain category, followed by visual queries to refine the search once you have retrieved an initial set of images. Or, if you have established a Visual Vocabulary™, you may wish to start right away with a visual search.

Visual Search Types

Search types are known as *indexes* in the eVe APIs. The four indexes are `Eve.COLOR`, `Eve.SHAPE`, `Eve.TEXTURE`, and `Eve.REGION`.



- **Color.** This type of search matches the predominant colors in the source and target images. The more colors the images have in common, and the more similar the proportions of those colors, the higher the target image's score.

- **Shape.** This type of search uses the two-dimensional outlines of objects within the image as patterns to match in the target image. The more similar the shape, the higher the target image's score.

- **Texture.** A texture search identifies unique textures within the image, and then ranks the target images based on their similarity to that texture.

- **Object Region.** Region searches are the most advanced search type. In an object region search, eVe identifies statistically similar parts of the image and groups them into a certain number of regions, or classes. You can then apply color, shape, or texture searches to the individual object regions.






























What is a Visual Vocabulary?

Using the eVe SDK API, you can enable a user to search through a library of images and videos using a Visual Vocabulary. A Visual Vocabulary is a representative list of images that fit a certain set of criteria. In other words, hundreds of thousands of images can automatically be distilled down to a representative set of images that can be used to quickly search the entire inventory of images. This set of representative images is referred to as a Visual Vocabulary.

A Visual Vocabulary allows a user to base an entire search on an image, rather than on a text entry. After a search, eVe will return a sortable set of images and videos based on visual similarity to the original image.

A simple Visual Vocabulary can be built using a method such as the following: given a pre-existing list of categories (such as jewelry, clothing, and sunsets), select an image from each category and add it to a vocabulary list. Do this until at least one image for each category has been added. Then, use these images as the basis for visual searches within their corresponding categories.

The following illustrates how you can use a visual vocabulary to perform different searches.

Visual Vocabulary images (base your search on one of these images)			
Category		Search Type	Results of Visual Search
Clothes		→ Shape →	 
		→ Shape →	   
		→ Shape →	   
		→ Shape →	  
Jewelry		→ Object →	  
		→ Object →	  
		→ Object →	  

Using the Sample Application

Overview

After you successfully install eVe, you can use the included sample application to view and interact with an implementation of its API. This application lets you try out the eVe Visual Search technology using a sample set of real data. Within the application, you can visually search on five categories of images: Clothes, Jewelry, Medical, Sunsets, and Textures.

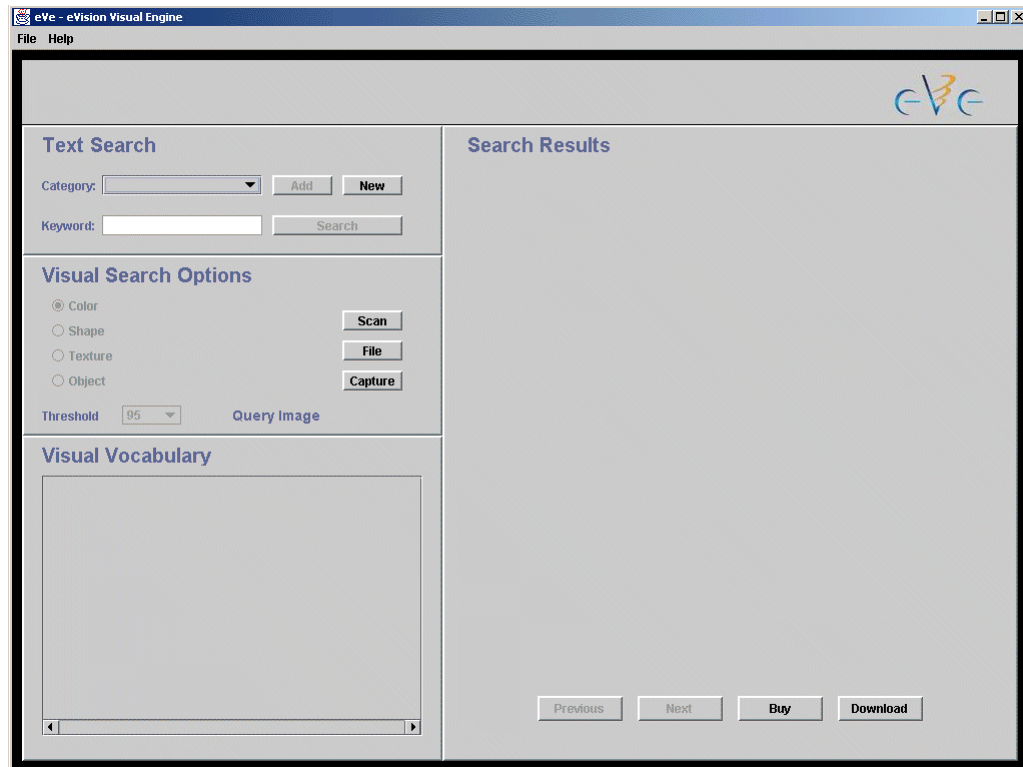
Note • The sample application is not a supported product. It is intended to provide you with a working model from which you can build an actual application for your organization. For example, not all buttons work in the sample application—they are included simply to give you ideas of the possible ways you can implement and initiate searches within eVe’s framework.

Getting Started

To start the sample application

- 1 Install the eVe SDK from the eVision CD or from the eVision web site (<http://www.evisionglobal.com/developers/sdk/>). See the *eVe 3 Professional SDK Installation Guide* for information on how to install eVe.
- 2 Select **Start ▶ Programs ▶ eVe 3 Professional ▶ SampleApp.bat**. The eVe sample application starts.

Note • The name of the eVe folder in the Start menu might be different if you changed the default settings upon installation.



Initiating Searches

The sample application shows how you can use eVe to initiate visual searches in a number of different ways:

- Use text
- Scan your own image (not enabled in sample application)
- Use an image file on your hard drive (not enabled in sample application)
- Capture an image from the web or other source (not enabled in sample application)
- Use an image in the Visual Vocabulary
- Pick an image from a result set obtained through text or visual search

The following sections describe how to use the sample application to initiate and perform the different searches supported by eVe.

Performing Searches

Before you can initiate a search, you must choose a category. A category represents a collection of related images (MediaObjects). This collection of images is also called a MediaCollection.

- ▶ Select a MediaCollection from the **Category** drop-down list. The MediaCollections available in the sample application are clothes, textures, jewelry, sunsets, and medical.

After you make a selection, the images in that MediaCollection appear in the **Search Results** section of the main window.

Text Searches

You can use eVe to enable metadata searching of the images in a MediaCollection. Metadata represents information that is associated with an image, such as a file name or related keywords.

To perform a standard text search:

- 1 Enter the text upon which you want to search in the **Keyword** field. For the sample application, keywords are limited to the text that appears in the file names of the images (you can enter partial file names as search criteria).
- 2 Click **Search**. The images whose file names contain the text you entered appear in the **Search Results** section.

Visual Searches

When searching a database for visual images, a visual search system bases its retrievals on the content of an image, and not on external tags, such as file name, captions, headings, keywords attached as metatags, etc. Its focus on the content of an image, rather than on manually defined external tags, makes applications based on eVe technology qualitatively more effective for image searches than any other type of search.

Using the sample application, you can perform visual searches based on an image from a:

- search result set
- visual vocabulary

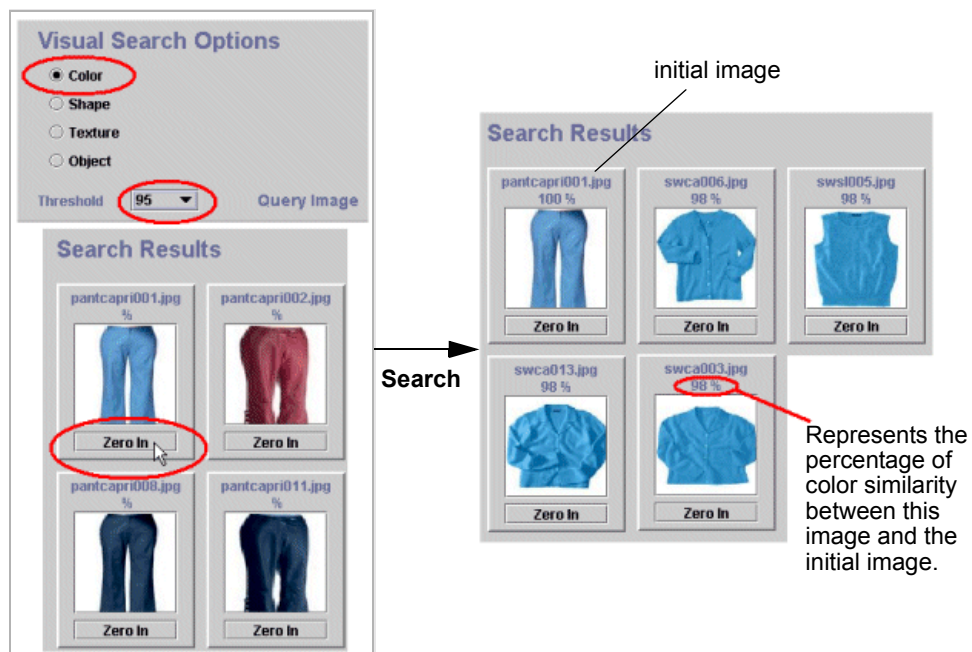
To perform a visual search on an image from a search result:

- 1 After selecting a category, select the image attribute upon which you want to search. The four attributes upon which you can search include color, shape, texture, and object:
 - A color search is based upon the visual similarity of the predominant colors within images.
 - A shape search is based on the two-dimensional outlines of objects within an image that are used as patterns to match against other images.
 - A texture search is based on unique textures within an image that is used to rank other images based on their similarity to that texture.

- Within an object search, eVe groups statistically similar parts of an image into a certain number of regions, or classes. The classes in the source image are then compared against the classes within other images to determine visual similarity.

Note • In the sample application, you can select one attribute upon which to search only. When using the full eVe SDK to add visual search functionality to an application, you can enable users to define a combination of attributes upon which to search.

- 2 Select a threshold for the search from the **Threshold** drop-down list. The value you select for the threshold determines what degree of visual similarity that the search should use as criteria when retrieving/comparing images. A high threshold means images must be very similar, thus fewer images are retrieved from the search; a low threshold relaxes the criteria, thus more images are retrieved from the search.
- 3 Click **Zero In** under an image in the **Search Results** section to initiate the search. The results of the search are displayed.



To perform a visual search on an image from a Visual Vocabulary:

After you select a MediaCollection from the **Category** drop-down list, the images that make up that MediaCollection's "vocabulary" appear in the **Visual Vocabulary** section of the window. A Visual Vocabulary is a list of images that are representative of the other images within a MediaCollection. You can base your visual search on one of these images.

For example, if you select **Clothes** from the **Category** drop-down list, the following images appear as vocabulary items for the Clothes MediaCollection:

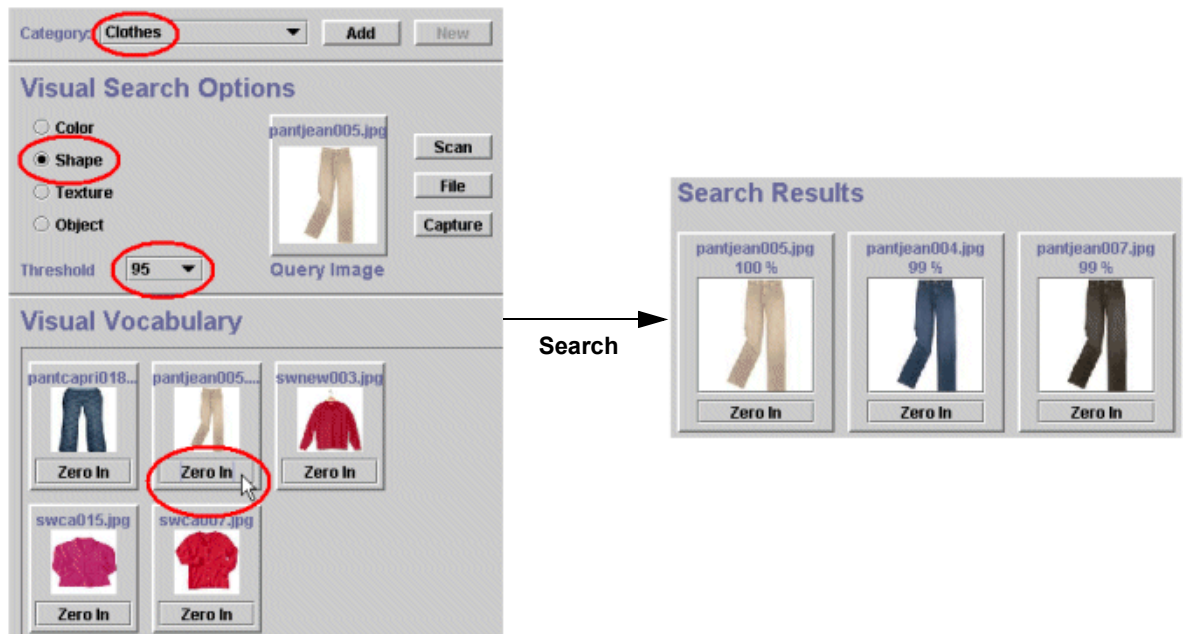


To perform a Visual Search using a Visual Vocabulary:

- 1 Select the image attribute upon which you want to search. The four attributes upon which you can search include color, shape, texture, and object.
 - A color search is based upon the visual similarity of the predominant colors within images.
 - A shape search is based on the two-dimensional outlines of objects within an image that are used as patterns to match against other images.
 - A texture search is based on unique textures within an image that is used to rank other images based on their similarity to that texture.
 - Within an object search, eVe groups statistically similar parts of an image into a certain number of regions, or classes. The classes in the source image are then compared against the classes within other images to determine visual similarity.

Note • In the sample application, you can select only one attribute at a time upon which to search. When using the full eVe SDK to add visual search functionality to an application, you can enable users to define a combination of attributes upon which to search.

- 2 Select a threshold for the search from the **Threshold** drop-down list. The value you select for the threshold determines what degree of visual similarity that the search should use as criteria when retrieving/comparing images. A high threshold means images must be very similar, thus fewer images are retrieved from the search; a low threshold relaxes the criteria, thus more images are retrieved from the search.
- 3 Click **Zero In** under an image in the **Visual Vocabulary** section to initiate the search. The results of the search are displayed.



To perform a visual search based on a file, scan, or capture:

These features are not active within the sample application, but can be implemented within the full version of the eVe SDK. The following describes each of these features:

- **Search Based on a Scanned Image.** Base a visual search on an image retrieved using a scanner.
- **Search Based on a File.** Base a visual search on an image file stored on your machine, corporate network, or on the Web.
- **Search Based on a Captured Image.** Base a visual search on an image captured by a digital camera or other image capturing device.

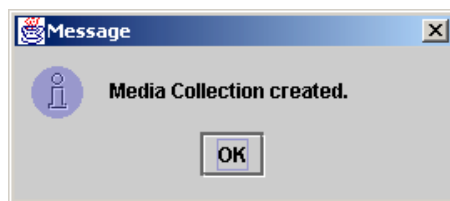
Adding Categories and Images

Adding a Category (MediaCollection)

If the predefined categories (Clothes, Jewelry, Medical, Sunsets, and Textures) in the sample application are not sufficient for your needs, you can add your own.

To add a category:

- 1 Click the **New** button. The Add Category dialog box appears.
- 2 Enter the name of the new category in the **New Category** field and click **OK**. A message window appears confirming that you have added a new category.



- 3 Click **OK**. The new category is now selectable from the **Category** drop-down list.

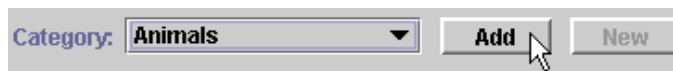
Once you have added a new category, you will need to add images to it. See the next section for information on how to do this.

Adding Images to a Category (MediaCollection)

You can add your own images to an existing category. When you add an image to a category in the sample application, the system automatically analyzes and indexes it for use with visual search.

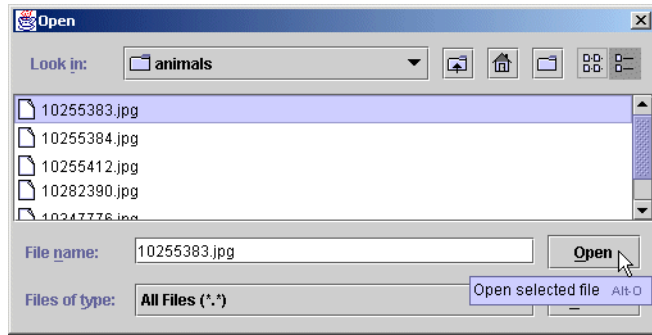
To add an image, or a folder of images:

- 1 Select the category to which you want to add images from the **Category** drop-down list.

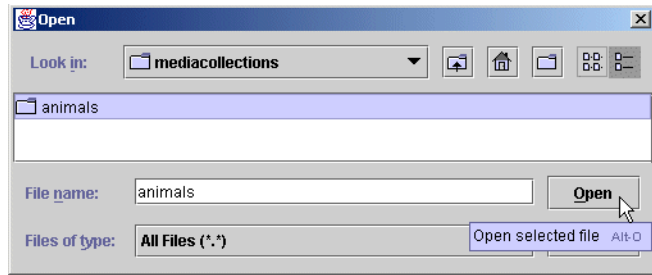


- 2 Click the **Add** button. The Open dialog box appears.
- 3 Use this window to navigate to the image or folder of images you want to add to the category.

- 4 Select the image or folder you want to add to the category.

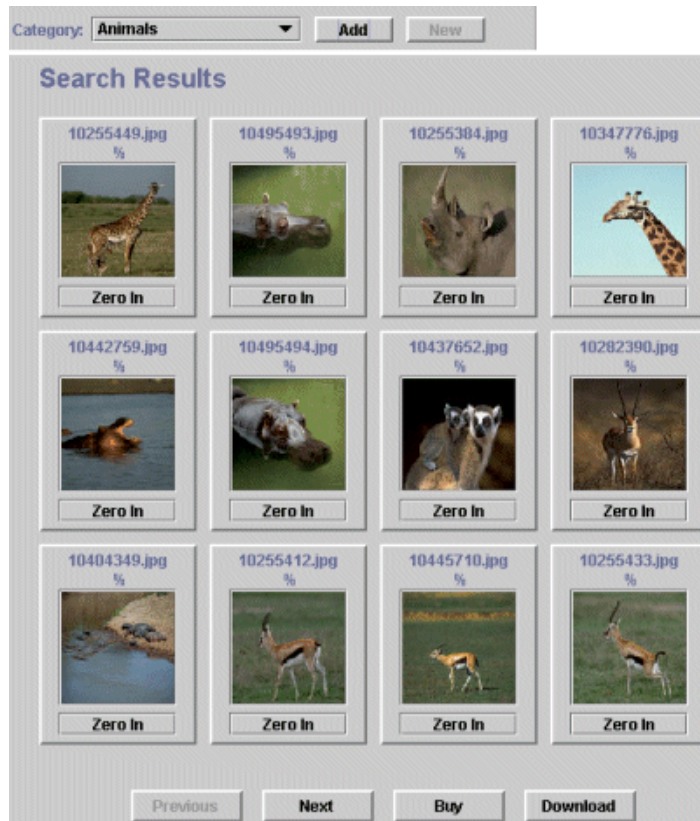


Adding an Image to a MediaCollection



Adding a Folder to a MediaCollection

- 5 Click **Open**. The system analyzes the image or folder of images and will make them available for searching.



Note • When you add an image, the system places a thumbnail copy of the image into the selected category. The original image remains untouched in the source directory.

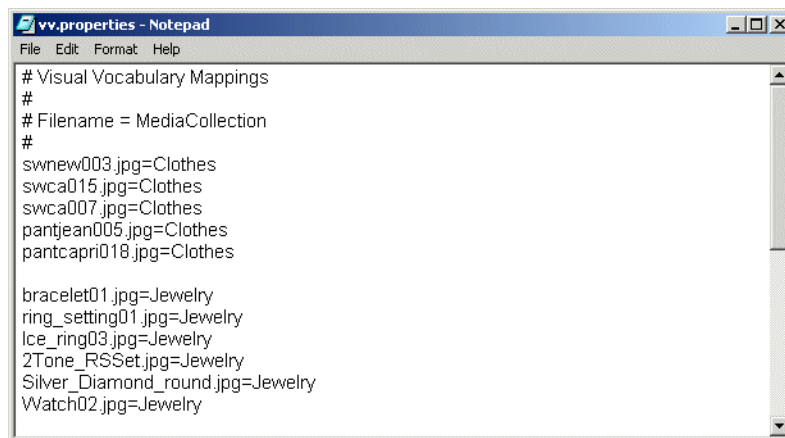
Creating a Visual Vocabulary

After you add a new MediaCollection to the sample application, you can define its Visual Vocabulary.

Note • The following procedure describes how to define Visual Vocabulary items for a category in the sample application. In the full version of the eVe SDK, you can provide multiple ways (both manual and automatic) for defining Visual Vocabularies.

To define a Visual Vocabulary

- 1 View the images in the new MediaCollection and determine which are representative of the collection as a whole. We recommend that you limit your selections for the Visual Vocabulary to six images or less.
- 2 Note the file names of the images you want to include in the Visual Vocabulary for the new MediaCollection.
- 3 Using a text editor, open the `vv.properties` file located at `\com\evision\global\eve\sample\`. This file contains the vocabulary definitions for the default categories included with the sample application.



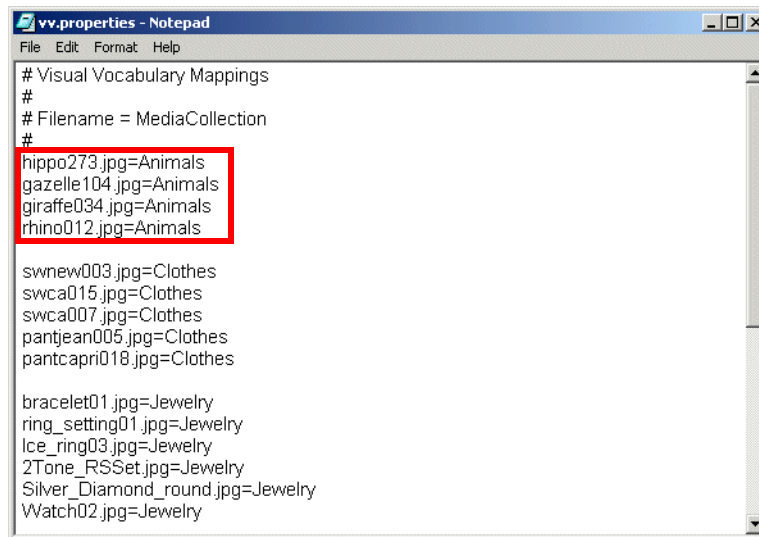
- 4 Add an entry for each of the images you want to include in the Visual Vocabulary for the new MediaCollection. Use the following format for each entry:

file_name=MediaCollection_name

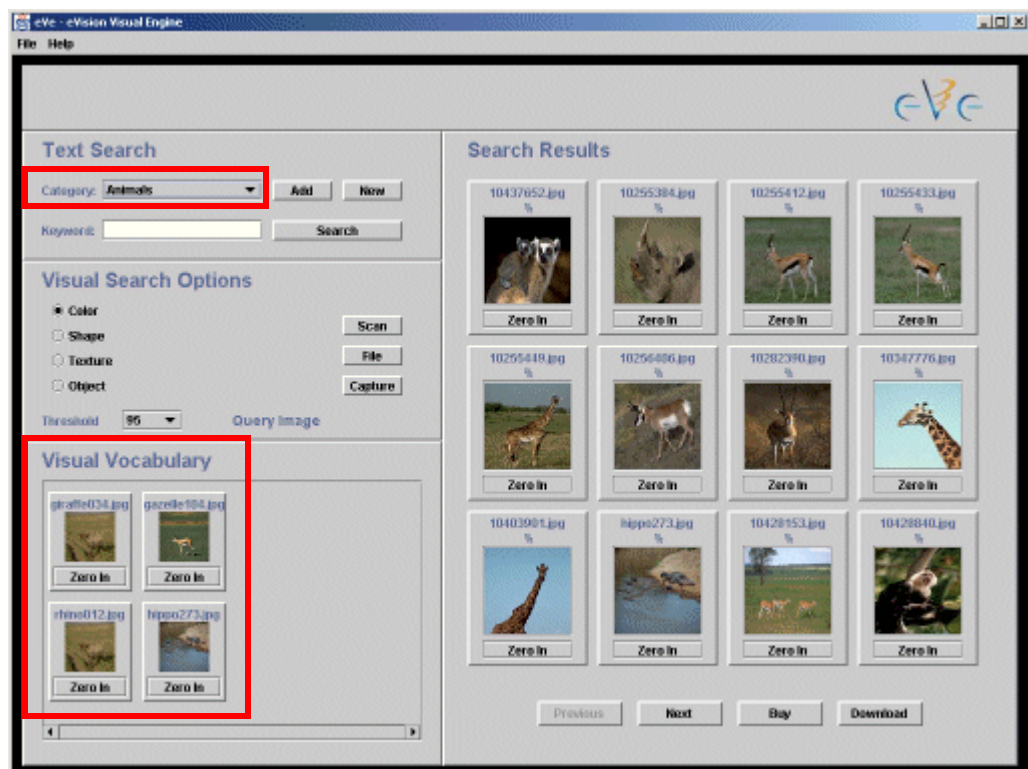
where:

- *file_name* represents the name of the image file that you want to add to the Visual Vocabulary.
- *MediaCollection_name* represents the name of the category for which you are defining the Visual Vocabulary item. This name must match the name used when the category was defined.

For example, entries for a MediaCollection named Animals might appear as:



- 5 Save the vv.properties file after you have defined the Visual Vocabulary items for the new category.
- 6 When you select the new category in the sample application, the vocabulary items will appear in the **Visual Vocabulary** section.




Window Description

This section describes the components of the sample application’s user interface.

- | | |
|------------------|--|
| File menu | <ul style="list-style-type: none"> ■ About. Displays information about the current version of the eVe SDK. ■ Exit. Closes the application. |
|------------------|--|


Text Search

Category	The sample application has its images organized into several categories. Internally, each category corresponds to a MediaCollection. The categories included with the sample application are: Clothes, Textures, Jewelry, Sunsets, and Medical.
-----------------	---

	Click this button to add your own images to a category (MediaCollection). When you add an image, the system automatically analyzes and indexes it.
---	--

Keyword	Enter text upon which you want to search. The text search is performed against the file names of the images within the current category.
----------------	--

In the full version of the eVe SDK, you can enable users to search for any meta data associated with an image, including file name.

	Click this button to search images based on the text you entered in the Keyword text box.
---	--






Visual Search Options

Color	A color search matches the predominant colors in the initial and target images. The more colors the images have in common, and the more similar the proportions of those colors, the higher the target image’s score.
--------------	---

Shape	Shape searches use the two-dimensional outlines of objects within the initial image as patterns to match in the target image. The more similar the shapes are, the higher the target image’s score.
--------------	---

Texture	A texture search identifies unique textures within the initial image, and then ranks the target images based on their similarity to that texture.
----------------	---

Object	Object region searches go beyond the other search types. In an object region search, eVe identifies statistically similar parts of images and groups them into a certain number of regions, or classes. The classes in the initial image are then compared against the classes in target images to determine visual similarity.
---------------	---

Threshold	Select the percentage threshold, 0 to 100, that determines what degree of visual similarity that the search should use as criteria when retrieving images. A high threshold means images must be very similar, thus fewer images are retrieved from the search; a low threshold relaxes the criteria, thus more images are retrieved from the search.
	While this button is inactive within the sample application, it demonstrates that you can use eVe to enable a visual search based on an image scanned using a scanner.
	While this button is inactive within the sample application, it demonstrates that you can use eVe to enable a visual search based on a file stored locally, on a corporate net, or on the Web.
	While this button is inactive within the sample application, it demonstrates that you can use eVe to enable a visual search based on a captured by a digital image or other image capturing device.
Visual Vocabulary	A Visual Vocabulary provides a representative sample of the images contained within a category. The Visual Vocabulary can take the place of a keyword search as a starting point for finding an image.
	Click this button to scroll backwards through the images in the search results.
	If there are more than 12 images in the search result, click this button to scroll through those results.



2

Integrating eVe into an Application

This chapter describes how to integrate the eVe engine within an application.

Overview	2-2
The Basics	2-2
Determine Requirements	2-2
Analyze and Design	2-4
Implement Design	2-6
eVe Demo Implementation	2-7
Overview	2-7
eVe Interfaces Used in eVe Demo	2-9
How the eVe Demo Works	2-10

Overview

The eVe application API allows a software developer to integrate the capabilities of eVision's visual search technology into a new or existing application or system. These capabilities include the analysis engine, the indexing component, and the visual search engine.

This chapter provides:

- basic information about how to integrate eVe with an application
- an example of the integration between eVe and an application

The Basics

Developing a plan for integrating eVe into an application is critical in making your visual search initiative successful. Typically, a software project plan is comprised of project phases, stages, methods, techniques, and practices that you can employ to best develop an application. Applying this same set of planning to implementing eVe can help you fully utilize its visual search capabilities.

Before attempting to integrate an eVe into an application, consider the following steps:

- 1 *Determine Requirements*
- 2 *Analyze and Design*
- 3 *Implement Design*

Determine Requirements

One of the most important things you can do before working with your application code is to clarify what capabilities within eVe can help you meet your application's business requirements. Determining the visual search functions you want to employ will speed up eVe integration and application deployment while ensuring that you address your customer's needs.

With eVe, some of the main visual search capabilities you can enable within your application include:

eVe Feature	Description
multiple search types	<p>Use eVe to make your application capable of performing a visual search based on any or all of the following properties of an image: texture, shape, color, or object region. You can allow users to search on one property at a time, or perform a search based on a combination of properties (such as color + shape). Additionally, you can enable users to specify the relative importance of each of the four basic properties within a search.</p> <p>Determining how a user of your application will initiate and define a search is critical to successfully integrating eVe.</p>
partial image searches	<p>Use this feature to enable users to select one or more regions within an image to use as search criterion. Because of the advanced segmentation technique built into eVe, partial image searching enables users to identify specific objects within an image and accurately search for both a whole image and for parts (or objects) of an image.</p> <p>If you choose to integrate this feature within your application, see the <i>Code Samples</i> chapter for an example of how to incorporate it into an application.</p>
Visual Vocabulary	<p>A Visual Vocabulary is a set of images that is representative of a larger set or “collection” of images. For example, if you have 1000 images of zebras, you can program eVe within your application to select a subset of zebra images from those 1000 images that are most representative of the entire collection.</p> <p>If you choose to integrate this feature within your application, you can enable users to base their searches on an image from a Visual Vocabulary rather than on a text-based entry. eVe allows you to programmatically automate the generation of a Visual Vocabulary for a database of images, and/or allow users to manually create their own vocabularies.</p>
similarity ranking	<p>eVe provides the <code>SearchResults</code> interface for creating, manipulating, and updating the results of a visual search. Within your application you can use eVe to sort, combine, truncate, rank, and append the results of a visual search.</p> <p>How you want to manipulate the results of a visual search will determine what methods from the <code>SearchResults</code> interface you will implement.</p>

eVe Feature	Description
multiple platform support	<p>eVe consists of 100% JAVA APIs. This allows you to quickly and easily integrate eVe into your new and existing applications on many different OS platforms.</p> <p>Check out the <i>Installation Guide</i> for a list of the Operating Systems that eVe supports.</p>
automatic asset ingestion	<p>You can use eVe's API to set up automatic generation of visual metadata, proxy, and thumbnail images for your media assets. This allows you to ingest, analyze, and index assets automatically.</p>
database independence	<p>eVe supports flat files, ObjectStore, and Oracle. You can choose the optimal data storage environment for your application.</p>

By determining the requirements of your application, you can ensure the best use of eVe to address your customers' functional requirements. During use case modeling and requirements review, you can nail down users' behavior in great detail; software behavior is dictated by user requirements.

Analyze and Design

After determining the visual search requirements of your application, you must understand and design how the application will actually perform a visual search and incorporate eVe's capabilities. In this phase, you must analyze and design your application's integration with eVe.

Using The APIs

eVe provides the API mechanism for enabling visual search within an application. This mechanism is designed to provide you with two different approaches for integrating eVe within applications: it consists of methods and classes organized within a Low-Level and High-Level API.

- **Low-Level API.** Use this API to have detailed control over the implementation of all the functions and processes available within the eVe visual search engine. The classes and interfaces in the `com.evisionglobal.eve.kernel` package make up the eVe low-level API and provide you with maximum access to eVe functionality.

If you use this API, see the *Low-Level API* chapter in the *API Reference Guide* for a complete description of all the interfaces and methods available for your use. If you want to perform a simple programming task without implementing the methods from the low-level API, you might find it quicker to use the high-level API.

- **High-Level API.** Use the methods within this API to quickly and easily implement some of the main features of eVe visual search within your application. The high-level API is a set of Java wrappers. These wrappers abstract the functionality of the eVe low-level API into one class, `MediaCollectionHL`. By abstracting some of eVe's functionality into one class, you can use the high-level API you to make fewer method calls to perform visual search

functions than with the low-level API. This allows you to quickly build applications that take advantage of eVision's visual search technology with the minimum of programming time.

If you use this API, see the *High-Level API* chapter in the *API Reference Guide* for a complete description of all the methods available for your use. If you want more control over the complete set of features and functions available within eVe, you might find that the low-level API better meets your development needs.

Making eVe Work with an Application

Whether you use the high- or low-level eVe API, there are five fundamental actions that your application must perform to enable eVe's visual search engine. These actions include:

- 1 Analyzing Files.** Before you can use eVe to search and retrieve an image from a database or flat file, you must *analyze* that source image. During analysis, eVe segments an image into distinct object regions, then generates scale-independent descriptions of those regions, known as visual signatures.

To generate a visual signature for an image, you instantiate the `MediaObject`, load the image into it, and pass it to the `analyze` method within the `analyze` class. See the *Analyzing a File* code sample in the *Code Samples* chapter for an example of how to code image analysis within an application.

- 2 Storing/Creating a MediaCollection.** eVe stores information about images in `MediaObjects` and `MediaCollections`. A `MediaObject` contains a single image and all the information related to that image. A `MediaCollection` is a container for multiple `MediaObjects`. You must store analyzed images in `MediaObjects` and `MediaCollections` before you can use eVe to perform visual searches.

See the *Creating a MediaCollection* code sample in the *Code Samples* chapter for an example of how to add an image to a `MediaCollection`.

- 3 Indexing.** After visual signatures for images are generated, they are organized in a proprietary indexing scheme for extremely fast retrievals. When using the high-level API, eVe automatically performs the indexing of images as you add or delete images for a `MediaCollection`. When using the low-level API, you must manually invoke methods for image indexing when deleting or adding images for a `MediaCollection`.
- 4 Performing a Search.** After analyzing and storing images, use eVe's API to enable the searching and sorting of images within a `MediaCollection`. eVe provides users with a number of methods for initiating a search, including text, image, object selection, and visual vocabulary.
- 5 Displaying Search Results.** Once a visual search is completed, use eVe to display the results of that search.

These five functions of the eVe SDK provide the building blocks for developing a visual search application. Use the code samples provided in the *Code Samples* chapter to see examples of how to code these functions within your application.

Implement Design

This phase forms the heart of a software project and is where the detailed design for your application is developed as well as the corresponding source code. The goal of this phase is to produce and test the software and supporting documentation to be transitioned to your user base.

You can view a sample implementation of the eVe SDK in the *eVe Demo Implementation* section.

eVe Demo Implementation

This section describes how the features and functions within eVe were implemented in the Basic Visual Search demo available on the eVision web site (<http://www.evisionglobal.com/tech/demo.html>). The demo demonstrates a sample application and interface that can be built using the capabilities within the eVe SDK. Search Engines, Digital Asset Management systems, and software developers can use the eVe SDK to create their own custom implementations.

Use the information in this section to get a better idea of how to use the different eVe interfaces and methods within your own application.

Overview

The web demo uses eVe's visual search technology to enable users to search and retrieve images from a sample set of real data. Users can visually search on five categories of images: Clothes, Jewelry, Medical, Sunsets, and Textures.

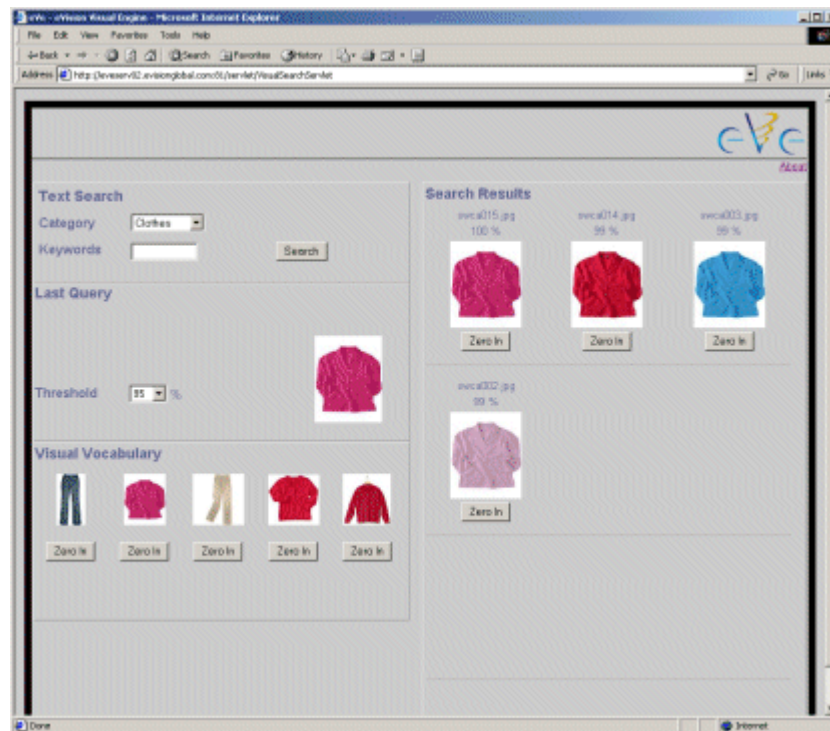


Figure 2-1 • eVe Web Demo

To run the eVe demo:

- 1 Point your web browser to <http://www.evisionglobal.com/tech/demo.html>.
- 2 Follow the onscreen instructions.

Note • We recommend you run the eVe demo before continuing with this section.

The eVe web demo is a Java servlet-based web application where servlets are used to:

- handle requests from the browser
- retrieve information from the database via the eVe API
- dynamically construct the HTML pages delivered to the client web browser.

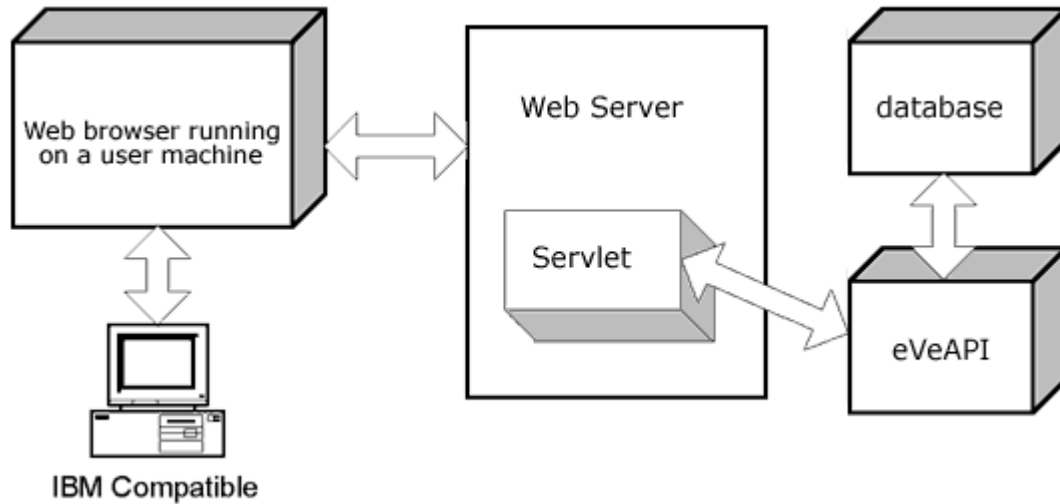


Figure 2-2 • Demo Application Overview

Java servlets on the webserver use the eVe SDK toolkit to perform the visual searches. The three main classes from the toolkit used by the Java servlets are:

- **MediaCollection**. Represents the database of images.
- **MediaObject**. Represents an image in the MediaCollection.
- **SearchResults**. A list of MediaObjects that satisfy the visual search criteria.

See the following section, *eVe Interfaces Used in eVe Demo*, for information about the eVe interfaces and methods used in the demo.

eVe Interfaces Used in eVe Demo

This section describes the interfaces and methods used by the eVe demo. See the *eVe SDK API Reference Guide* for more detailed information about these interfaces and methods.

MediaCollection

The following methods from the `MediaCollection` interface were used in the eVe demo:

method	what does it do?
<code>open(String Path)</code>	Opens the <code>MediaCollection</code> contained in the directory stored in path.
<code>getKeys</code>	Returns the keys for all <code>MediaObjects</code> in the <code>MediaCollection</code> .
<code>getMediaObject(long key)</code>	Retrieves the <code>MediaObject</code> with the specified key.
<code>search(MediaObject eve, SearchParameter parameters)</code>	Searches an opened <code>MediaCollection</code> for <code>MediaObjects</code> that are visually similar to the specified <code>MediaObject</code> .

MediaObject

The following method from the `MediaObject` interface was used in the eVe demo:

- `getProperty("originalFilename")` — returns the file name for the image represented by the `MediaObject`.

SearchResults

The following methods from the `SearchResults` interface were used in the web demo:

method	what does it do?
<code>getKey()</code>	Returns the key for a <code>SearchResults</code> object.
<code>getSimilarity()</code>	Returns the similarity for a <code>SearchResults</code> object based on the search criteria.

SearchParameters

The following method from the `SearchParameters` interface was used in the web demo:

- `setSearch(int indextype, Boolean value, float percent)` — use this method to tell the object what type of search it is to perform (such as color or texture) and the weighting of that search. The search parameter (color, shape, texture, or object) used in the search of the image database is fixed in the demo. However, the full eVe SDK allows the adjustment of search parameters by the user.

How the eVe Demo Works

A single servlet, `VisualSearchServlet`, uses three classes to dynamically build new HTML document pages based on search results: `TextSearch`, `VisualSearch`, and `SearchParameter`. Both `TextSearch` and `VisualSearch` are derived from an abstract base class, `SearchEngine`.

To process a visual search request, the `VisualSearchServlet`:

- 1 creates an instance of `MediaCollection` and uses its `open()` method to load all images stored in the corresponding image database. The value supplied to the `path` argument in the `open()` method determines what images are loaded.
- 2 uses `MediaCollection.getKeys` to retrieve a list of the keys to the `MediaObjects` in the `MediaCollection` category.
- 3 uses `MediaCollection.getMediaObject` to get a list of all the `MediaObjects` in the opened image database. Each retrieved `MediaObject` instance can be used as input to `MediaCollection.search` to find `MediaObjects` representing visually similar images.

Sample Code for Visual Search

```
String imagepath =
    "c:\\com\\visionglobal\\eve\\sample\\categories\\Clothes";

MediaCollection mediaCollection = (MediaCollection)
    Eve.newMediaCollection();
mediaCollection.open( imagepath );
long[] keys = mediaCollection.getKeys();
MediaObject[] mObjects = mediaCollection.getMediaObject( keys );
int colorPercent = 100, shapePercent=0; texturePercent=0; objectPercent=0;

for ( int i = 0; i < mObjects.length(); ++i )
{
    SearchParameters sp = (SearchParameters) Eve.newSearchParameters();

    sp.setSearch(Eve.COLOR, true, (float)(colorPercent/100));
    sp.setSearch(Eve.SHAPE, true, (float)(shapePercent/100));
    sp.setSearch(Eve.TEXTURE, true, (float)(texturePercent/100));
    sp.setSearch(Eve.REGION, true, (float)(objectPercent/100));
    MediaObject mo = mediaCollection.getMediaObject( mObject[i].getKey() );

    SearchResults[] results = mediaCollection.search(mo, sp);
}
```



3

Code Samples

This chapter contains code samples and reference code designed to help you get started using eVe as quickly as possible. While the samples do not form a complete sample application, most are useful chunks of code in their own right.

Analyzing a File	3-2
Creating a MediaCollection	3-3
Performing a Search	3-4
Refining a Search	3-6
Using EveContext	3-7
Searching Multiple Collections	3-8
Searching with Metadata and Images	3-9
Performing Partial Image Selection	3-10
Overview	3-10
Code Samples	3-11
<i>To Get A Segmentation Mask Image:</i>	3-11
<i>To Search for One Region in an Image:</i>	3-12
Determining the Distance Between Images	3-13
EveExceptions	3-14
Using Before and After Command Methods	3-15

Analyzing a File

Note • Image files must be in one of the following formats before you run them through analysis: JPEG, GIF, or PNG. For information about how to prepare images for analysis, see the *Additional Configuration* section in the *Installation Guide*.

In this sample, we:

- create a `MediaObject` to contain the image
- add the image to the `MediaObject`
- analyze the image
- save the `MediaObject` to a file

```
try
{
    // create a MediaObject and load the image
    MediaObject mo = (MediaObject) Eve.newMediaObject();
    mo.loadImage(path);
    // keep track of where the image came from
    mo.setProperty("originalPath",path);
    // create an instance of the Analyze class to perform the analysis
    Analyze analyze = (Analyze) Eve.newAnalyze();
    try
    {
        // do the analysis
        analyze.analyze(mo,Eve.maxRegions,Eve.maxIterations);
    }
    catch (Exception e)
    {
        System.out.println("Error During Analyze: " + e);
        System.exit(0);
    }
    // add some dummy metadata
    mo.addMetadata("testMetadata","testMetadata");
    // take out everything we can't serialize
    mo.purge();
    // serialize the MediaObject and send it out to disk
    mo.saveTo(path + ".edf");
}
catch (Exception e)
{
    // handle the exception here
}
```

Creating a MediaCollection

In this sample, we:

- create a MediaCollection on disk
- add a pre-analyzed EDF file (perhaps made with the previous code sample, *Analyzing a File*)
- close the collection

```
// set up the file names and locations
String path = "\\temp\\mymediacollection"
String edf = "\\temp\\mymediaobject"

// create a MediaCollection object and a MediaObject
MediaCollection mediaCollection = (MediaCollection) Eve.newMediaCollection();
MediaObject mediaObject = (MediaObject) Eve.newMediaObject();

try
{
    // create the collection on disk
    mediaCollection.create (path);

    // populate the MediaObject
    mediaObject.loadFrom (edf);

    // add the MediaObject to the collection
    mediaCollection.add (mediaObject);

    // close the collection
    mediaCollection.close();
}
catch (Exception e)
{
    System.out.println("Error: " + e);
    e.printStackTrace();
    // always try to close the MediaCollection
    try
    {
        mediaCollection.close();
    }
    catch (Exception ignore)
    {
    }
}
System.exit(0);
```

Performing a Search

In this sample, we:

- use an existing `MediaCollection` to select a `MediaObject` to use as a source for the search
- perform the search
- retrieve and display the resulting images.

```
// create a MediaCollection object to hold the MediaCollection on disk
MediaCollection mediaCollection = (MediaCollection) Eve.newMediaCollection();

try
{
    // open the collection and get the keys
    mediaCollection.open ("\\temp\\myCollection");
    long results[] = mediaCollection.getKeys();
    // make a MediaObject to hold the query image
    MediaObject mediaObject = (MediaObject) Eve.newMediaObject();
    mediaObject = mediaCollection.getMediaObject (results[0]);
    // make a SearchParameters object
    SearchParameters searchParameters = (SearchParameters)
    Eve.newSearchParameters();
    // set the parameters to do a 50-50 shape and texture search
    searchParameters.setSearch(Eve.SHAPE, true, 0.5);
    searchParameters.setSearch(Eve.TEXTURE, true, 0.5);
    // make a SearchResults object
    SearchResults searchResults[] = mediaCollection.search
    (mediaObject, searchParameters);

    // iterate through the results and output the filenames
    for (i = 0; i < searchResults.length; i++)
    {
        MediaObject temp = mediaCollection.getMediaObject
        (searchResults[i].getKey());

        String filename = (String) temp.getProperty("originalFile");

        System.out.println(
            "Target: " + i+ " File: " + filename + " : " + searchResults[i]);

        // if everything checks out, raise the flag
        if (mediaObject.getKey() == searchResults[i].getKey())
            flag = true;
    }
    // close the collection
    mediaCollection.close();
    // report our final status
    if (flag)
        System.out.println("EDFSearchTest: OK");
    else
        System.out.println("EDFSearchTest: Error: Find Failed");
}
catch (Exception e)
```

```
{
    System.out.println("Error: " + e);
    e.printStackTrace();
    System.out.println("EDFSearchTest: Error");
    try
    {
        // always try to close the collection
        mediaCollection.close();
    }
    catch (Exception ignore)
    {
    }
}
```

Refining a Search

Your first set of search results may not contain exactly what you want. If this is the case, you'll need to refine your results.

The following code:

- opens a `MediaCollection`
- gets a list of keys from the collection
- does a random search
- resubmits the fifth item in the result set for another search using the same parameters

```
// make a new MediaCollection object
MediaCollection mc = (MediaCollection) Eve.newMediaCollection();

try
{
    mc.open (directoryName);
}
catch (Exception e)
{
    // handle the exception
}

// Get a list of keys from the collection
long keys[] = mediaCollection.getKeys();

// Use the third key entry to get a source MediaObject
MediaObject source = mediaCollection.getMediaObject (keys[2]);

// Use this source object for a 50% color/50% region search
SearchParameters parameters = Eve.newSearchParameters();
parameters.setSearch (Eve.COLOR, true, 0.5);
parameters.setSearch (Eve.REGION, true, 0.5);

// Perform the search
SearchResults results[] = mediaCollection.search( source, parameters );

// To refine the search (for example, to select other entries),
// simply resubmit with one of the previous results
SearchResults refinedResults[] =
mediaCollection.search(results[4],parameters);
```

Using EveContext

Eve and EveContext contain the `maxSearchBins` parameter which tells the search engine how many potential index “clusters” it should examine before returning a list of search results.

WARNING • Setting this to an extremely high value forces the search engine to search every cluster. Be forewarned that this creates a very large performance hit. Do not use a high value as your default setting.

```
// open the MediaCollection and set maxSearchBins to an extremely
// high value to guarantee you'll search every bin
MediaCollection mc = Eve.newMediaCollection();
mc.open (directoryName);
EveContext context = new EveContext();
context.maxSearchBins = 999999;

// tell the collection to use the new parameters
mc.setContext( context );

// Perform the search
SearchResults results[] = mc.search( source, parameters );
```

Be aware that certain parameters are used at various points in the engine. As an example, if the `MediaCollection` resides in a relational database, the `MediaCollection.open()` method opens a connection to the database. Thus, to use a different password than the default in `Eve.properties`, you must do the following:

```
// create a MediaCollection object and context
MediaCollection mc = Eve.newMediaCollection();
EveContext context = new EveContext();

// set the database options
context.databaseUser = "MyUserName";
context.databasePassword = "MyPassword";

// tell the collection to use the new parameters
mc.setContext( context );

// open the mediaCollection
// note that you must still specify a path to the disk-based portions
// of the collection such as the index
mc.open( directoryPath );
```

Searching Multiple Collections

The following code sample assumes that you want to find similar images of a car from several media collections. In this example, we:

- open two disk-based collections
- search each of the collections
- combine the results of the two searches
- sort the combined results

```
// create the two MediaCollection objects and open the two collections
MediaCollection mc01 = Eve.newMediaCollection();
MediaCollection mc02 = Eve.newMediaCollection();
mc01.open( directoryPath1 );
mc02.open( directoryPath2 );

// read in a media object containing the car picture
MediaObject source = Eve.newMediaObject();
source.loadFrom (“\\images\\carpicture.jpeg.edf”);

// set up a 50-50 color and region search
SearchParameters parameters = Eve.newSearchParameters();
parameters.setSearch(Eve.COLOR,true,0.5);
parameters.setSearch(Eve.REGION,true,0.5);

// perform the two searches using the same source object and parameters
SearchResults results1[] = mc01.search(source,parameters);
SearchResults results2[] = mc02.search(source,parameters);

// add the results of the two searches together
SearchResults worker = Eve.newSearchResults();
SearchResults finalResults[] = worker.append(results1,results2);

// sort the combined results by similarity
finalResults = worker.similaritySort(finalResults);

// truncate the results after the first 32 entries
// (results are sorted in descending order of similarity)
finalResults = worker.chop(finalResults,32);
```

At this point you have a single array with (possible) entries from each collection. To determine which collection generated a particular result, use the `SearchResults.getCollectionName()` method.

Searching with Metadata and Images

Assume that in a `MediaCollection` you have images of boats and cars, and that each image has an `imageType` metadata tag set to either `car` or `boat`. In addition, each image also has a metadata tag of `primaryColor` set to `red`, `green`, or `blue`. Thus, given a particular red car, you can conduct a metadata search followed by a visual search as follows:

```
// make a new MediaCollection object and populate it from a collection on disk
MediaCollection mc = Eve.newMediaCollection();
mc.open( directoryPath );

// find all MediaObjects tagged as cars in the collection
SearchResults carResults[] = mc.metadataFind( "imageType", "car" );

// get a list of all objects tagged as red in the collection
SearchResults redResults[] = mc.metadataFind( "color", "red" );

// combine the two lists using a Boolean AND operation
SearchResults worker = Eve.newSearchResults();
SearchResults target[] = worker.and( carResults, redResults );

// load the source image (the red car)
MediaObject source = Eve.newMediaObject();
Source.loadFrom( myMediaObjectPath );

// perform the visual search
SearchParameters parameters = Eve.newSearchParameters();
Parameters.setSearch( Eve.REGION, true, 1.0 );
SearchResults finalResults[] = mc.search( source, parameters, target[] );
```

Performing Partial Image Selection

Overview

A digital image consists of a number of visual groups formed by visually similar pixels. These visual groups are referred to as objects, and any given image contains a number of objects that appear together to define that image. Segmentation is the process by which an image is divided into object regions, with each region identifying a similar set of objects. Once these object regions are identified, features such as color, texture, and shape are extracted from each of the object regions.

A `MediaObject` contains the following information:

- a copy of the image that was analyzed
- a segmentation mask, which is a two-dimensional byte array that maps each pixel in the image to a specific class. Each byte in the array contains a value of 0, 1, or 2 corresponding to the class to which the image's pixels were mapped.

Note • If the `maxRegions` parameter in the `eve.properties` file is set to a value other than 3, each byte in the array will contain values other than 0, 1, 2. For example, if `maxRegions` is set to 4, there would be four distinct colors in a segmentation mask and the two-dimensional byte array would contain the values 0, 1, 2, 3.

Because of the advanced segmentation technique built into `eVe`, you can enable users to identify specific objects within an image and accurately search for both a whole image and for parts (or objects) of an image.

The following describes how you can use `eVe` to enable partial image selection for users:

- Determine the classes on which the user is clicking
- Extract the signatures for the classes from the corresponding `MediaObject`
- Construct a "dummy" `MediaObject` containing the signatures
- Pass the "dummy" `MediaObject` to search as the source

The following steps describe in detail how to program `eVe` for partial image selection within your application:

- 1 Obtain a source image and display. Use the methods in the `ImageManager` interface to perform this step.
- 2 Obtain the segmentation mask for the source image and display. Use the methods in the `ImageManager` interface to perform this step.

- 3 Obtain the 2D byte array corresponding to the segmentation mask.
(`byte segmentationMask[][] = (byte[][]) mediaObject.getProperty("segmentationMask")`)

4 When the user selects the output from either step 1 or step 2, capture the x,y coordinates of the mouse click and compare these against the results from step 3 to determine the class in which the user is interested.

5 When users activate the search function, use eVe to perform the following:

a construct a new dummy media object

b iterate over the particular classes of interest:

```
Vector newColorSignatures = new Vector();  
Vector newRegionSignatures = new Vector();  
etc.
```

c for each class selected, code:

```
newColorSignatures.add(source.getIndex(Eve.COLOR,<selected class>);  
newRegionSignatures.add(source.getIndex(Eve.REGION,<selected class>);
```

```
dummyMediaObject.setIndex(Eve.COLOR,newColorSignatures);  
dummyMediaObject.setIndex(Eve.REGION,newRegionSignatures);
```

d call the media collection's search method:

```
SearchResults results[] = mediaCollection.search(dummyMediaObject,<your  
search parameters>);
```

Code Samples

To Get A Segmentation Mask Image:

```
MediaObject mo=mc.getMediaObject(12345); //get a MediaObject somehow.
```

```
Image i=Eve.newImageManager().getSegmentationMaskImage(moo);
```

To Search for One Region in an Image:

```
String regs="01"; //this indicates we want to search using only regions 0 and 1, or "red"
                and "green"
                //we would set this to "02" for "red" and "blue".
                //or "12" for "green" and "blue".

                //for an understanding of these scare-quoted colors, look at a
                segmentation mask image.

MediaObject newmo=Eve.newMediaObject(); //create a temporary media object, currently
empty.

Vector[] vs={mo.getIndex(Eve.COLOR), mo.getIndex(Eve.TEXTURE), mo.getIndex(Eve.SHAPE),
mo.getIndex(Eve.REGION)};
//create a working vector containing all indexes for all search axes.

int[] ins={Eve.COLOR, Eve.TEXTURE, Eve.SHAPE, Eve.REGION};
//this is a temporary array to allow us to reduce code by looping.

/*
    Now, for each basic search axis,

*/
for (int i=0;i<4;i++){

    /*
        insert into the new MediaObject only the indexes for the regions we have
        selected,thus creating an "artificial" query object that reflects only our
        chosen regions.
    */

    Vector newvec=new Vector();
    Vector oldvec=vs[i];
    for (int j=0;j<oldvec.size();j++){
        if (regs==null || regs.length()==0 || regs.indexOf((j)+"")!=-1){

            //here, add the index but only if the index is one of our selected indexes.
            //TO BE CLEAR, oldvec is a vector that contains as many indexes as there
            //are regions in the image - one index per region, in red, green, blue
            //order. Note that there may be fewer regions, depending on the
            //EveContext used during analysis or the Eve.properties file used during
            //analysis.

            newvec.addElement(oldvec.elementAt(j));

        }
    }
    newmo.setIndex(ins[i], newvec);
}
mo=newmo;
```

Determining the Distance Between Images

Use eVe's Distance method to retrieve a floating point value representing the distance between the indexed signatures of two EDFs. This only applies to one index type at a time. For example, you could get the result texture distance from a source image to a target image.

```
// open a new media collection
//
MediaCollection mediaCollection = Eve.newMediaCollection();
try
{
    mediaCollection.open("<some path>");
    //
    // get list of existing keys
    //
    long keys[] = mediaCollection.getKeys();
    //
    // pick up the first two entries and retrieve the MediaObjects
    //
    MediaObject record1 = mediaCollection.getMediaObject(keys[0]);
    MediaObject record2 = mediaCollection.getMediaObject(keys[1]);
    //
    // now - determine the color distance between the two records
    //
    Distance colorDistance = Eve.newColorDistance();
    Vector record1ColorSignature = record1.getIndex(Eve.COLOR);
    Vector record2ColorSignature = record2.getIndex(Eve.COLOR);
    double distance =
colorDistance.distance(record1ColorSignature,record2ColorSignature);
    System.out.println("Distance: " + distance);
}
catch (Exception e)
{
    }
}
```

EveExceptions

You can instantiate an `EveException` in one of three ways:

- `EveException(String fromClass,String fromMethod,Exception e)`
- `EveException(String fromClass,String fromMethod,String message,Exception e)`
- `EveException(String fromClass,String fromMethod,String message)`

`EveExceptions` are designed primarily for logging purposes. For example, if you instantiate an `EveException` with an exception, it will create a stack trace, log it to disk, and return. The `fromMethod` and `fromClass` features are used to designate the method in the code that triggered the exception.

Note • Do not forget to enable error logging. Refer to the error logging section of *The eve.properties File* section in the *Installation Guide* for more information on enabling error logging.

The following code catches a Java exception, creates a new `EveException` to wrap it, and throws the new `EveException`:

```
Public void myMethod( myMethodOptions )
{
    try
    {
        // Something that could blow up
    }
    catch (Exception exception)
    {
        throw new EveException( "myClass", "myMethod", "your very own error
        message",
        exception);
    }
}
```

`EveException.stackTrace` (a string variable) contains a formatted stack trace. This stack trace is exactly the trace that is logged, with timestamp, to the error log directory specified in the `Eve.properties` file.

Using Before and After Command Methods

The methods in the `MediaCollection` interface of the low-level API allow you to execute commands before and after the action performed by those methods. To do this, each method within `MediaCollection` accepts the `CommandList` `beforeMethods` and `CommandList` `afterMethods` parameters. These parameters call a command list before or after a method is run and execute the command(s) included in that list. You can use before and after command methods to perform actions against an image (EDF) that is specified within a method.

The basic steps for incorporating before and after commands within methods are:

- 1 Write the command you want to perform.
- 2 Place the command into the `commandList`.
- 3 Incorporate a call to the commands within a `MediaCollection` method.

In the following example, we created a before command method that will gray scale an image before it goes to analysis. This command method, named `GrayScaleCommand`, is included within a `mediaCollection.analyze` method call. This first section of code calls the command method and passes an EDF to that command.

```

.
.
.
    MediaObject edf = (MediaObject) Eve.newMediaObject();
    System.out.println("Loading JPG: " + files[i]);
    edf.loadImage(directory + Eve.directorySeparator + files[i]);
    edf.setProperty("originalDirectory",directory);
    edf.setProperty("originalFilename",files[i]);

    CommandList commandList = context.newCommandList();
    commandList.setContext(context);
    commandList.add("com.evisioglobaleve.commands.GrayScaleCommand");

    try
    {
        mediaCollection.analyze(edf,commandList,null);
    }
    catch (Exception e)
    {
        System.out.println("Error During Analyze: " + e);
        System.exit(0);
    }
.
.
.

```

where:

- *CommandList* is the code that contains a list of commands that will be executed sequentially before or after the *MediaCollection* method. *CommandList* also examines the results of a command and determines the next action to perform.

```

public interface CommandList
{
    final int CONTINUE = 0;
    final int RETURN = 1;
}

```

```
final int ABORT = 2;
```

There are three possible response codes from the result of a command:

- CONTINUE. Continue processing the next command or invoke the method (if no more commands are specified).
- ABORT. Exit the command sequence without performing an action.
- RETURN. Exit the command and return a value.
- *mediaCollection.analyze* is the method containing the before and after commands. After the before command method is executed, the specified EDF will be analyzed.
- *edf* represents the image against which you want to perform a command method and analysis.
- the location of *commandList* in the line indicates that you want to perform the command before the method is passed to the eVe engine.
- the location of *null* in the line indicates that there are no after method commands to perform

The following is the actual code found in the `GrayScaleCommand` command method. It will perform the following actions:

- pull an image from a `MediaObject`
- turn that image into a gray scale image
- place the image back into the `MediaObject`
- pass the `MediaObject` to be analyzed

```
package com.evisionglobal.eve.commands;

import com.evisionglobal.eve.*;
import com.evisionglobal.eve.kernel.*;

import java.awt.*;
import java.awt.image.*;
import java.awt.image.renderable.*;

import java.util.*;
import java.io.*;

public class GrayScaleCommand extends BaseCommand
{
    public void execute()
    {
        try
        {
            //
            // initialize
            //
```

```
lastStatus = CONTINUE;
lastException = null;

//
// make sure we are being given the right stuff
//

if (parameter1 == null)
{
    lastStatus = ABORT;
    lastException = new EveException("GrayScaleCommand","execute","Null
    Parameter1");
    return;
}

if (!(parameter1 instanceof MediaObject))
{
    lastStatus = ABORT;
    lastException = new
    EveException("GrayScaleCommand","execute","Parameter1 Not
    MediaObject");
    return;
}

MediaObject eve = (MediaObject) parameter1;

double red[][] = eve.getRedChannel();
double green[][] = eve.getGreenChannel();
double blue[][] = eve.getBlueChannel();

int i,j;

int width = red.length;
int height = red[0].length;

int pixels[] = new int[width * height];

for (i = 0; i < width; i++)
    for (j = 0; j < height; j++)
    {
        double temp = (0.30 * red[i][j]) + (0.59 * green[i][j]) + (
        0.11 * blue[i][j]);
        red[i][j] = green[i][j] = blue[i][j] = temp;
        pixels[j * width + i] = (new Color((int) temp,(int) temp,(int)
        temp)).getRGB();
    }

Image newImage = Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(width,height,pixels,0,width));
ByteArrayOutputStream stream = new ByteArrayOutputStream(width *
height);
JpegEncoder encoder = new JpegEncoder(newImage,100,stream);
encoder.Compress();
byte buffer[] = stream.toByteArray();

eve.setColorPlanes(red,green,blue);
eve.setProperty("image",buffer);
```

```
        eve.setProperty("GrayScaleCommand","Executed");
    }
    catch (Exception e)
    {
        lastStatus = ABORT;
        lastException = new EveException("GrayScaleCommand","execute",e);
    }
}
}
```

where:

- *Basecommand* contains the basic parameters that the `commandList` references to execute the command method. This includes input parameters, output parameters (the results of the command), and an execute command. To enable before and after commands, you must subclass the *Basecommand* within your command method code.



Index

A

Analysis
code sample [3-2](#)

B

before and after command methods, code sample [3-15](#)

C

code samples
analyzing a file [3-2](#)
creating a mediacollection [3-3](#)
determining distance between images [3-13](#)
object/region selection [3-10](#)
performing a search [3-4](#)
refining a search [3-6](#)
searching with Metadata and images [3-9](#)
searching with multiple collections [3-8](#)
using before and after command methods [3-15](#)
using eVeContext [3-7](#)

D

distance, code sample [3-13](#)

E

EveContext, code sample [3-7](#)
EveException, code sample [3-14](#)

M

MediaCollection
code samples
creating a MediaCollection [3-3](#)

O

object selection, code sample [3-10](#)

R

region selection, code sample [3-10](#)

S

Searching
combining Metadata and Images [3-9](#)
multiple MediaCollections [3-8](#)
performing a search [3-4](#)
refining a search [3-6](#)

