eVision

eVe 3 Professional

API Reference Guide

eVision eVe 3 Professional API Reference Guide

Printed: 2001

Publication Number: EVE-300-APIRG-00

Mailing Address

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

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	*eVe High-Level API*	Discusses the eVe high-level API and its methods.
2	*eVe Low-Level API*	Discusses the eVe low-level API and its interfaces. If you want to perform a simple programming task, you may find it quicker to use the high-level API. However, if you want to perform more complicated tasks, or need to override system defaults, you must use the low-level API.
3	*XML Interface to eVe*	Describes the XML interface to eVe commands.
4	*Error Handling*	Describes the three forms of EveException, differentiated by the number and type of their arguments.

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 **Enter**. ...Click **Modify**.
	what a user types	type `RUN APP.EXE` in the Application field
Alternate color	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 ***Chapter 3, eVe High-Level API***.
Italic	words that are emphasized	...the entry *after* the current entry...
	the titles of other documents	*eVe Installation and Getting Started Guide*
	syntax variables	`COPY filename`
`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 API Reference Guide*, you might find it helpful to have these additional books available for reference:

- *eVe 3 Professional Installation Guide*
- *eVe 3 Professional Getting Started 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

Table of Contents

1

eVe High-Level API

This chapter discusses the eVe high-level API and its methods.

Overview

The eVe high-level API is a set of Java wrappers. These wrappers abstract the functionality of the eVe low-level API into one class, `MediaCollectionHL`. See ***Chapter 3***, ***eVe Low-Level API***. The methods in this class allow you to create and manipulate MediaCollections and MediaObjects stored on disk, as well as perform searches and more.

Note • You cannot use the high-level API to manipulate MediaCollections stored in a database.

Getting Started

Before you start using the eVe APIs, you must install the SDK and verify your installation. You might also have to modify the `Eve.properties` file, which the system reads at runtime to determine system default values. For more information, see the *Installation Guide* and *Getting Started Guide*.

MediaCollectionHL Class

There is one class in the eVe high-level API: `MediaCollectionHL`. The rest of this chapter is a reference to the methods within that class. Each section is organized alphabetically by method name.

Method Summary

This `MediaCollectionHL` class includes the following methods:

Method	Description	Method	Description
addFolder	Adds a folder of images to the current MediaCollection as MediaObjects.	***isImage***	Determines if a file is an image file.
addFolder (MediaCollection)	Adds a folder of images to the specified MediaCollection as MediaObjects.	***metadataSearch (String, String...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
addImage	Adds an image or a pre-analyzed EDF file to the current MediaCollection as a MediaObject.	***search(key...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
addImage (MediaCollection)	Adds an image or a pre-analyzed EDF file to the specified MediaCollection as a MediaObject.	***search(key, int similarity...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
close	Closes the MediaCollection.	***search (MediaObject...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
deleteImage	Removes an image from the current MediaCollection.	***search(MediaObject, int similarity...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
deleteImage (MediaCollection)	Removes an image from the specified MediaCollection.	***search(metatag...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.
exists	Determines whether a MediaCollection is stored in the specified location.	***search(metatag, int similarity...)***	Searches the MediaCollection for images similar to the given image, using the provided search options.

Method	Description	Method	Description
getCollection	Retrieves the name of the MediaCollection being searched.	**search** (String ...)	Searches the MediaCollection for images similar to the given image, using the provided search options.
getImagePath	Retrieves the path to where a MediaCollection's images are stored.	**searchResults andResults**	Performs a logical AND operation on two arrays of SearchResults objects.
getMetadataKeys	Retrieves all the unique metadata keys in the MediaCollection.	**searchResults appendResults**	Concatenates two arrays of SearchResults.
getMediaObject	Retrieves the MediaObject with the specified key.	**searchResults chopResults**	Truncates the array of SearchResults to the given length. If the array is already that length or shorter, it is returned unmodified.
getMediaObject (SearchResults ...)	Retrieves the MediaObject referenced by the SearchResults object.	**searchResults orResults**	Performs a logical OR operation on two arrays of SearchResults objects.
getMediaObjects	Retrieves all the MediaObjects in a MediaCollection.	**searchResults sortResults**	Sorts the search results by similarity in descending order.
getMediaObjects (SearchResults ...)	Retrieves the MediaObjects referenced by the SearchResults objects.	**size**	Reports the number of MediaObjects in the MediaCollection.
isEdf	Determines if a file is in EDF file format.	**search** (String ...)	Searches the MediaCollection for images similar to the given image, using the provided search options.

Methods

This sections lists and describes all the methods in the `MediaCollectionHL` class.

addFolder

format	`void addFolder(String path) throws Exception`
description	Adds a folder of images to the *current* MediaCollection. Before adding the images from the folder to the MediaCollection, this method converts them to MediaObjects (EDF files).
parameters	■ `path` — the full path of the folder to be added (for example, `c:\images\sunsets`)
returns	nothing (void)
throws	`addFolder()` throws an Exception if the addition fails or if `path` does not exist or is not a folder.

addFolder (MediaCollection)

format	`public void addFolder(String path, MediaCollection mCollection) throws Exception`
description	Adds a folder of images to the MediaCollection you specify. Use this method if you want to add a folder of images to a MediaCollection other than the current MediaCollection. Before adding the images from the folder to the MediaCollection, this method converts them to MediaObjects (EDF files).
parameters	■ `path` — the full path of the folder to be added (for example, `c:\images\sunsets`) ■ `mCollection` — the MediaCollection to which you want to add the folder of images
returns	nothing (void)
throws	`addFolder()` throws an Exception if the addition fails, if `path` does not exist or is not a folder, or if the `mCollection` does not exist or is not a MediaCollection.

addImage

format	`void addImage(String path) throws Exception`
description	Adds an image or a pre-analyzed EDF file to the *current* MediaCollection. Before adding an image to the MediaCollection, this method converts it to a MediaObject (EDF file).
parameters	■ `path` — the path and filename of the item to add (for example, `c:\images\sunsets\sunset1.gif`)
returns	nothing (void)
throws	`addImage()` throws an Exception if `path` does not exist, is not a file, or if the image is not a supported image type.

addImage (MediaCollection)

format	`void addImage(String path MediaCollection mCollection) throws Exception`
description	Adds an image or a pre-analyzed EDF file to the MediaCollection you specify. Use this method if you want to add an image to a MediaCollection other than the current MediaCollection. Before adding an image to the MediaCollection, this method converts it to a MediaObject (EDF file).
parameters	■ `path` — the path and filename of the image to add (for example, `c:\images\sunsets\sunset1.gif`) ■ `mCollection` — the MediaCollection to which you want to add the image
returns	nothing (void)
throws	`addImage()` throws an Exception if `path` does not exist, is not a file, if the image is not a supported image type, or if `mCollection` does not exist.

close

format	`void close() throws Exception`
description	Closes the MediaCollection.
parameters	none
returns	nothing (void)
throws	`close()` throws an Exception if the close operation fails.

deleteImage

format	`void deleteImage(String path) throws Exception`
description	Removes the specified image from the *current* MediaCollection.
parameters	■ `path` — the full path and filename of the image to be removed
returns	nothing (void)
throws	`deleteImage()` throws an Exception if path does not exist, if it is not a file, or if it is not a valid image type. As well, if an error occurs while the file is being removed from the MediaCollection, `delete()` throws that EveException as a string in an Exception.

deleteImage (MediaCollection)

format	`void deleteImage(String path MediaCollection mCollection) throws Exception`
description	Removes the image from the specified MediaCollection. Use this method if you want to delete an image from a MediaCollection other than the current MediaCollection.
parameters	■ `path` — the full path and filename of the image to be removed ■ `mCollection` — the MediaCollection from which you want to delete the image
returns	nothing (void)
throws	`deleteImage()` throws an Exception if path does not exist, if it is not a file, if it is not a valid image type, or if `mCollection` does not exist. Additionally, if an error occurs while the file is being removed from the MediaCollection, `delete()` throws that EveException as a string in an Exception.

exists

format	`boolean exists(String path)`
description	Determines whether a MediaCollection is stored in the specified location.
parameters	■ `path` — the full path (but not the filename) to the potential MediaCollection
returns	■ `TRUE` — if there is a MediaCollection in the specified location ■ `FALSE` — if there is no MediaCollection in the specified location
throws	`exists()` does not throw any exceptions.

getCollection

format	`getCollectionName() throws Exception`
description	Retrieves the name of the MediaCollection.
parameters	none
returns	the name of the MediaCollection
throws	The `getCollectionName()` method throws an Exception if the retrieval fails.

getImagePath

format	`getImagePath()`
description	Retrieves the path to where a MediaCollection's images are stored.
parameters	none
returns	the path to all images stored in the MediaCollection.
throws	The `getImagePath()` method does not throw any exceptions.

getMetadataKeys

format	`getMetadataKeys() throws Exception`
description	Retrieves all the unique metadata keys in the MediaCollection.
parameters	none
returns	an array of strings containing the keys of all the MediaCollection's metadata
throws	The `getMetadataKeys()` method throws an Exception if the retrieval fails.

getMediaObject

format	`public MediaObject getMediaObject( long key ) throws Exception`
description	Retrieves the MediaObject with the specified key.
parameters	■ `key` — the index key of the MediaObject you wish to retrieve
returns	the requested MediaObject
throws	The `getMediaObject()` method throws an Exception if the retrieval fails.

getMediaObject (SearchResults ...)

format	`public MediaObject getMediaObject( SearchResults sr ) throws Exception`
description	Retrieves the MediaObject referenced by the SearchResults object.
parameters	■ `sr` — a `SearchResults` object that references a MediaObject
returns	the requested MediaObject, if it exists
throws	The `getMediaObject()` method throws an Exception if the retrieval fails.

getMediaObjects

format	`public MediaObject[] getMediaObjects() throws Exception`
description	Retrieves all the MediaObjects in a MediaCollection.
parameters	none
returns	the MediaObjects in the MediaCollection
throws	The `getMediaObjects()` method throws an Exception if the retrieval fails.

getMediaObjects (SearchResults ...)

format	`public MediaObject[] getMediaObjects( SearchResults[] sr ) throws Exception`
description	Retrieves the MediaObjects referenced by the SearchResults objects.
parameters	■ `sr` — an array of `SearchResults` objects that reference MediaObjects
returns	an array of the requested MediaObjects
throws	The `getMediaObjects()` method throws an Exception if the retrieval fails.

isEdf

format	`public boolean isEdf(File f)`
description	Determines if a file is in EDF file format.
parameters	■ `f` — the name of the file to be checked
returns	■ `TRUE` — the file is an EDF file ■ `FALSE` — the file is not an EDF file
throws	The `isEdf()` method does not throw an exception.

isImage

format	`public boolean isImage(File f)`
description	Determines if a file is an image file.
parameters	■ `f` — the name of the file to be checked
returns	■ `TRUE` — the file is an image file ■ `FALSE` — the file is not an image file
throws	The `isImage()` method does not throw an exception.

metadataSearch (`String, String...`)

format	`SearchResults[] search( String path, String value ) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options.
parameters	■ `path` — the full path and filename of the EDF or image file containing the source image ■ `value` — the text of the EDF or file you wish to retrieve
returns	an array of `SearchResults` objects, sorted in descending order of relevancy
throws	The `metadataSearch()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search(key...)

format	`search( long key, int colorPercent, int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options.
parameters	■ `key` — the index key of the MediaObject you wish to retrieve ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search(key, int similarity...)

format	`search( long key, int similarity, int colorPercent, int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options (including `similarity`).
parameters	■ `key` — the index key of the MediaObject you wish to retrieve ■ `similarity` — 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. ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search (MediaObject...)

format	`SearchResults[] search(MediaObject mObj, int colorPercent,int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options.
parameters	■ `mObj` — the MediaObject containing the source image ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search(MediaObject, int similarity...)

format

```
search( MediaObject mObj, int similarity,int colorPercent, int
     shapePercent, int texturePercent, int objectPercent)
     throws Exception
```

description

Searches the MediaCollection for images similar to the given image, using the provided search options.

parameters

- `mObj` — the MediaObject containing the source image
- `similarity` — 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.
- `colorPercent` — the percentage weighting, from 0 to 100, of color in the search
- `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search
- `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search
- `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search

returns

an array of SearchResults, sorted in descending order of relevancy

throws

The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search(metatag...)

format	`search( int metatag, int colorPercent, int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options.
parameters	■ `metatag` — a metatag associated with the MediaObject you wish to retrieve ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search(metatag, int similarity...)

format	`search( int metatag, int similarity, int colorPercent, int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options (including `similarity`).
parameters	■ `metatag` — a metatag associated with the MediaObject you wish to retrieve ■ `similarity` — 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. ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

search (String ...)

format	`SearchResults[] search(String path, int colorPercent, int shapePercent, int texturePercent, int objectPercent) throws Exception`
description	Searches the MediaCollection for images similar to the given image, using the provided search options.
parameters	■ `path` — the full path and filename of the EDF or image file containing the source image ■ `colorPercent` — the percentage weighting, from 0 to 100, of color in the search ■ `shapePercent` — the percentage weighting, from 0 to 100, of shape similarity in the search ■ `texturePercent` — the percentage weighting, from 0 to 100, of texture similarity in the search ■ `objectPercent` — the percentage weighting, from 0 to 100, of object similarity in the search
returns	an array of SearchResults, sorted in descending order of relevancy
throws	The `search()` method throws an Exception if `path` does not exist, is not a file, or is not in a supported image file format. As well, it throws an Exception if any of the percentage parameters is less than `0` (zero) or greater than `100`.

searchResults andResults

format	`public SearchResults[] andResults( SearchResults[] lSource, SearchResults[] rSource )`
description	Performs a logical AND operation on two arrays of SearchResults objects.
parameters	■ `lSource` — the first array of SearchResults ■ `rSource` — the second array of SearchResults
returns	a one-dimensional array of SearchResults objects
throws	The `searchResults andResults ()` method does not throw any exceptions.

searchResults appendResults

format
```
public SearchResults[] appendResults( SearchResults[] source,
     SearchResults[] target )
```

description Concatenates two arrays of SearchResults.

parameters
- `source` — the first array of SearchResults
- `target` — the first array of SearchResults

returns a one-dimensional array of SearchResults objects

throws The `searchResults appendResults ()` method does not throw any exceptions.

searchResults chopResults

format
```
public SearchResults[] chopResults( SearchResults[] source,
     int pos )
```

description Truncates the array of SearchResults to the given length. If the array is already that length or shorter, it is returned unmodified.

parameters
- `source` — an array of search results
- `pos` — the position in the search results at which to truncate

returns a one-dimensional array of SearchResults objects with `pos` or fewer members

throws `searchResults chopResults ()` does not throw any exceptions.

searchResults orResults

format
```
public SearchResults[] orResults( SearchResults[] lSource,
     SearchResults[] rSource )
```

description Performs a logical OR operation on two arrays of SearchResults objects.

parameters
- `lSource` — the first array of SearchResults
- `rSource` — the first array of SearchResults

returns a one-dimensional array of SearchResults objects

throws The `searchResults orResults ()` method does not throw any exceptions.

searchResults sortResults

format	`public SearchResults[] sortResults( SearchResults[] source )`
description	Sorts the search results by similarity in descending order.
parameters	■ `source` — an array of search results
returns	a sorted one-dimensional array of ranked SearchResults objects
throws	The `searchResults sortResults ()` method does not throw any exceptions.

size

format	`long size() throws Exception`
description	Reports the number of MediaObjects in the MediaCollection.
parameters	none
returns	a long integer containing the number of MediaObjects in the MediaCollection
throws	`size()` throws an Exception if an error occurs while the system is counting the MediaObjects.

2

eVe Low-Level API

This chapter describes the eVe low-level API and its interfaces. If you want to perform a simple programming task, you may find it quicker to use the high-level API. However, if you want to perform more complicated tasks, or need to override system defaults, you must use the low-level API.

Overview

The classes and interfaces in the `com.evisionglobal.eve.kernel` package make up the eVe low-level API. These interfaces give you access to all of eVe's power and functionality.

If you want to perform a simple programming task, you may find it quicker to use the high-level API. See Chapter 2, ***eVe High-Level API*** for more information about the high-level API. If you want to perform more complicated tasks, or need to override system defaults, you must use the low-level API

Getting Started

Before you start using the eVe APIs, install the SDK and verify your installation. You may also have to modify the `Eve.properties` file. For details, refer to the *Installation and Getting Started Guide*.

Interface Summary

The following table summarizes the interfaces and methods available in the low-level API. .

Interface	Description
Analysis	
Analyze	Provides the following methods to perform analysis on a particular MediaObject. ■ ***analyze*** — Analyzes a given `MediaObject` and replaces it in its containing MediaCollection. `analyze()` also applies any current EvePatchesto the MediaObject after it has been analyzed. ■ ***setContext*** — Sets the context of the analysis. If you are using only one database, you likely do not need to use this method. However, if you are using more than one database, you need to use an EveContext object to enable the system to find the MediaObject you wish to analyze.
Distance	Provides the ***distance*** method for calculating the distance between the vectors of two EDFs.
EveContext	Provides the capability to override default eVe environment settings.

Interface	Description
FrameGrabber	Allows you to extract information from video files. ■ ***close*** — Closes the video file. ■ ***getMediaObject*** — Converts the specified frame from a video file into a MediaObject. ■ ***getControlComponent*** — Retrieves the controls (such as volume settings) of the appropriate video player, if available. ■ ***getVisualComponent*** — Retrieves the component of the appropriate video player that reads and plays the current video file. ■ ***getTotalDistance*** — Determines the visual similarity between the current and previous frame. If the visual similarity between frames is greater than the value defined for the `frameGrabberMinimumDistance` parameter in the `eve.properties` file, then the current frame is considered a keyframe. ■ ***getTotalFrames*** — Determines the total number of frames within a video file. ■ ***gotoFrame*** — Jumps to the specified frame within the video file. ■ ***gotKeyFrame*** — Determines the visual similarity between the specified frame and the previous frame in a video file. If the visual similarity is greater than the value defined for the `frameGrabberMinimumDistance` parameter in the `eve.properties` file, then the current frame is considered a keyframe. ■ ***open*** — Opens the video file stored in the specified location. ■ ***play*** — Uses the Java™ Media Framework API (JMF) to read/play the video file. ■ ***setContext*** — Sets the FrameGrabber's context information

Interface	Description
ImageManager	Provides the following methods for manipulating images and converting between image formats. ■ ***getImage*** — Retrieves an image from the buffer. ■ ***getImage (MediaObject)*** — Retrieves the image (.jpg) stored within a MediaObject. ■ ***getImageIcon (MediaObject)*** — Retrieves the image (.jpg) stored within a MediaObject. The image is retrieved as an icon and can be used within an application. ■ ***getSegmentationMask*** — Retrieves a MediaObject's segmentation mask. ■ ***getSegmentationMaskIcon*** — Retrieves a MediaObject's segmentation mask. The segmentation mask is retrieved as an icon and can be used within an application. ■ ***loadImage*** — Loads an image into eVe for processing. ■ ***newMediaObject*** — Creates a new MediaObject. This is useful, for example, if you wish to generate a query image on-the-fly. ■ ***resize*** — Resizes an image to `maxwidthOrHeight` pixels square. This method reproportions the image so that it is a square in order to facilitate image analysis. ■ ***saveImage*** — Saves an image to disk. ■ ***setContext*** — Sets the ImageManager's context information. ■ ***supportedImageTypes*** — Returns the list of image types that ImageManager can process. ImageManager currently supports 68 different image file formats.
Storage and Indexing	
EvePatch	Unsupported. For eVision internal use only.
MediaCollection	Provides the following methods for opening, closing, and manipulating MediaCollections and their component MediaObjects. ■ ***add*** — Adds a MediaObject to the MediaCollection. ■ ***close*** — Closes the current MediaCollection. ■ ***create*** — Creates a new MediaCollection in the specified location. ■ ***delete (MediaObject)*** — Removes a MediaObject from the MediaCollection. ■ ***delete (long)*** — Removes a MediaObject from the MediaCollection. ■ ***getCollectionName*** — Returns the name of the MediaCollection. ■ ***getKeys*** — Gets all the MediaObject keys in the MediaCollection. ■ ***getMediaObject (long)*** — Retrieves the MediaObject with the given key. ■ ***getMediaObject (long[])*** — Retrieves the MediaObjects with the given keys.

Interface	Description
MediaCollection (continued)	■ ***getMediaObject (SearchResults)*** — Retrieves the MediaObject referenced by the `SearchResults` object. ■ ***getMediaObject (SearchResults[])*** — Retrieves the MediaObjects referenced by the SearchResults objects. ■ ***getMetadataKeys*** — Returns all the unique metadata keys in the MediaCollection. ■ ***getProperties*** — Returns all the properties associated with the MediaCollection. Currently the only property set is the MediaCollection's name, but you can set any serializable object as a property. ■ ***getProperty*** — Retrieves a property object from the MediaCollection. Because MediaCollections may contain multiple property items with the same key, as long as the values are different, getProperty() returns an object that may contain multiple items. ■ ***metadataFind (String)*** — Finds metadata items with a particular key in all of the MediaObjects in the MediaCollection. ■ ***metadataFind (String, String)*** — Finds all MediaObjects containing a particular key-value pair in the MediaCollection. ■ ***open*** — Opens the MediaCollection contained in the directory stored in `path`. ■ ***reorganize*** — Optimizes the data structures within the MediaCollection to allow for faster searching. ■ ***save*** — Save the changes to the current MediaCollection without closing. ■ ***search (MediaObject, SearchParameters)*** — Performs a search against the MediaCollection using the given MediaObject as a source. ■ ***search (MediaObject, SearchParameters, SearchResults[])*** — Performs a search against a set of search results using the given MediaObject and search parameters. ■ ***search (MediaObject, SearchParameters, MediaObject[])*** — Performs a search against a list of MediaObjects, using the given MediaObject and search parameters. ■ ***setCollectionName*** — Sets the name of the MediaCollection to `collectionName`. ■ ***setContext*** — Sets the context information for the MediaCollection. ■ ***setProperty*** — Sets a property in a MediaCollection. ■ ***size*** — Reports the number of MediaObjects in the MediaCollection. ■ ***update*** — Replaces a MediaObject in the MediaCollection with a new MediaObject of the same name.

Interface	Description
MediaObject	Provides the following methods for opening, closing, and manipulating MediaObjects. ■ ***addMetadata*** — Adds a metadata item to the MediaObject. ■ ***applyPatch*** — Applies an EvePatch to the MediaObject. ■ ***deleteMetadata*** — Removes all metadata from the MediaObject. ■ ***deleteMetadata (`String`)*** — Removes a metadata item from the MediaObject. ■ ***deleteMetadata (`String, String`)*** — Removes a metadata item from the MediaObject. ■ ***getBlueChannel*** — Retrieves the contents of the image's blue channel as an array of pixel values. ■ ***getCollectionName*** — Retrieves the name of the MediaCollection in which the MediaObject is stored. ■ ***getGreenChannel*** — Retrieves the contents of the image's green channel as an array of pixel values. ■ ***getHeight*** — Retrieves the height of the MediaObject's image. ■ ***getIndex*** — Retrieves the image's index signature of the requested type from the MediaObject. ■ ***getKey*** — Retrieves the MediaObject's index key. ■ ***getMetadata*** — Retrieves all of the MediaObject's metadata. ■ ***getMetadata (`String, String`)*** — Retrieves the specified metadata item, if it exists. ■ ***getProperties*** — Retrieves all of a MediaObject's properties. ■ ***getProperty*** — Retrieves a property from the MediaObject. ■ ***getRedChannel*** — Retrieves the contents of the image's red channel as an array of pixel values. ■ ***getWidth*** — Retrieves the width of the MediaObject's image. ■ ***loadFrom*** — Loads a MediaObject from an EDF file on disk. ■ ***loadImage*** — Loads an image into the MediaObject. ■ ***makeArray*** — Creates a single-dimensional array of MediaObjects of the specified length. ■ ***purge*** — Cleans up a MediaObject. The `purge()` method removes everything from the MediaObject that is not needed for analysis. ■ ***saveTo*** — Writes the MediaObject to an EDF file on disk to support serialization. ■ ***setCollectionName*** — Sets the name of the MediaCollection in which the MediaObject is stored. ■ ***setColorPlanes*** — Allows you to manually create an image by specifying pixel maps for each of its three color planes.

Interface	Description
MediaObject (continued)	■ ***setContext*** — Sets the context information for the MediaObject. ■ ***setHeight*** — Sets the height of the MediaObject's image. ■ ***setIndex*** — Sets an index directly within the MediaObject. ■ ***setKey*** — Sets the MediaObject's index key. ■ ***setProperty*** — Sets the given property for the MediaObject. ■ ***setWidth*** — Sets the width of the MediaObject's image. ■ ***updateMetadata*** — Assigns a new value to an existing metadata item.
Metadata	Provides the following methods for adding and manipulating metadata. ■ ***getCollectionName*** — Gets the name of the MediaCollection in which the metadata is stored. ■ ***getID*** — Gets the unique ID of the metadata item. ■ ***getKey*** — Gets the key of the metadata item. ■ ***getValue*** — Gets the value of the metadata item. ■ ***setCollectionName*** — Sets the name of the MediaCollection in which the metadata item is stored. ■ ***setID*** — Sets the unique ID of the metadata item. ■ ***setKey*** — Sets the key of the metadata item. ■ ***setValue*** — Sets the value of the metadata item.
Search and Retrieval	
SearchParameters	Provides the following methods for setting and manipulating search parameters. ■ ***getAscending*** — Reports whether search results will be ordered in ascending or descending order. Normally, results are returned in descending order. ■ ***getSearch*** — Determines if a particular type of search is enabled. ■ ***getWeight*** — Reports the weighting of a particular search type, such as region or texture. ■ ***setAscending*** — Instructs the search to order results in ascending or descending order. ■ ***setSearch (`int, boolean, double`)*** — Sets the search options for the SearchParameters object. ■ ***setSearch (`int, double`)*** — Sets the search options for the SearchParameters object.

Interface	Description
SearchResults	Provides the following methods for getting and manipulating search results. ■ ***and*** — Performs a logical AND operation on two arrays of SearchResults objects. ■ ***append*** — Concatenates two arrays of SearchResults. ■ ***chop*** — Truncates the array of SearchResults to the given length. ■ ***distanceSort*** — Takes an unordered array of SearchResults objects and sorts it according to distance. ■ ***getCollectionName*** — Gets the name of the MediaCollection being searched. ■ ***getDistance*** — Gets the current result's distance from the target image. ■ ***getKey*** — Gets the key of the MediaObject to which the SearchResults object refers. ■ ***getRank*** — Retrieves the current result's rank relative to the other results of the search. ■ ***getSimilarity*** — Returns the similarity score of the search result against the target image. ■ ***makeArray (int)*** — Creates a one-dimensional array of SearchResults objects of the specified length. ■ ***makeArray (long[])*** — Creates a one-dimensional array of SearchResults objects and populates it with the given objects. ■ ***normalize*** — Normalizes a set of SearchResults and its associated parameters. ■ ***not*** — Performs a logical NOT operation on two arrays of SearchResults objects. ■ ***or*** — Performs a logical OR operation on two arrays of SearchResults objects. ■ ***rank*** — Assigns a ranking to each member of the given array of SearchResults objects. ■ ***rankSort*** — Sorts the given array of ranked (but unordered) SearchResults objects and populates it with the given SearchResults objects, ordered according to their rankings. ■ ***setCollectionName*** — Sets the name of the MediaCollection being searched. ■ ***setContext*** — Sets the context of the search results. ■ ***setDistance*** — Sets the current result's *distance* from the target image. ■ ***setKey*** — Sets the key of the SearchResults object. ■ ***setRank*** — Sets the rank of the current result relative to the other results of the search. ■ ***setSimilarity*** — Sets the degree of similarity between the search result and the target image. ■ ***similaritySort*** — Takes the given array of unordered SearchResults objects and sorts it according to the objects' similarity scores.

Interface	Description
Vocabulary	Provides access to functionality for creating a Visual Vocabulary. A Visual Vocabulary is a representative list of images that fit a certain set of criteria. ■ ***create(`Mediacollection, long[], SearchParameters, double, int`)*** — Use this method to generate a Visual Vocabulary for a specific location/folder based on the values defined for the parameters. ■ ***create(`Mediacollection, SearchParameters, double, int`)*** — Use this method to generate a Visual Vocabulary for a database of images based on the values defined for the parameters. ■ ***setContext*** — Sets the Vocabulary's context information.
`EveException`	Provides exception handling to eVe programs. This is an extension of the Java `Exception` class, and not actually an interface.

Interface Reference

The following sections are organized alphabetically by interface name. Refer to the summary table above for a list of interface names and their corresponding page numbers in this chapter.

Within a particular interface's section, methods are organized alphabetically. If you want to find a particular method name, but don't know its parent interface, look it up in the index at the end of this book.

Analyze

You use the `analyze()` method in this interface to analyze MediaObjects.

To create a new Analyze object, you use the `Eve.newAnalyze()` method. This method optionally takes an EveContext object as an argument.

analyze

format	`boolean analyze(MediaObject mediaObject,int maxRegions,int maxIterations) throws EveException`
description	Analyzes a given `MediaObject` and replaces it in its containing MediaCollection. `analyze()` also applies any current EvePatches (see the ***EvePatch*** section) to the MediaObject after it has been analyzed.
parameters	■ `mediaObject` — the object to be analyzed ■ `maxRegions` — the number of object regions (1 through *n*) to analyze ■ `maxIterations` — the maximum number (1 through *n*) of iterations to perform during image analysis
returns	■ `TRUE` — if the analysis was successful ■ `FALSE` — if the analysis failed
throws	The `analyze()` method throws an EveException if the analysis fails.

setContext

format	`void setContext(EveContext eveContext)`
description	Sets the context of the analysis. If you are using only one database, you likely do not need to use this method. However, if you are using more than one database, you need to use an EveContext object to enable the system to find the MediaObject you wish to analyze. For more information on EveContext, see the ***EveContext*** section.
parameters	■ `eveContext` — the context of the object to be analyzed
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

Distance

Use the `distance(java.util.Vector source, java.util.Vector target)` method to calculate the distance between the visual signatures of two EDF images.

distance

format	`distance(java.util.Vector source, java.util.Vector target)`
description	Retrieves 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 the target.
parameters	■ `source` — the index of the inital EDF (MediaObject) whose distance you want to calculate against another EDF ■ `target` — the EDF (MediaObject) whose indexed signature you want to comapre against the source EDF
returns	a double-precision floating-point value representing the current result distance from the target image
throws	The `distance()` method does not throw any exceptions.

EveContext

This class provides a way to override all of the system defaults established in the `Eve.properties` file. This can be useful, for instance, to allow you to take advantage of multiple databases and object stores. eVe is storage-independent, meaning that you can store each of your MediaCollections in databases or in files on disk, and all methods will access them in exactly the same way.

If all of your MediaCollections are stored in one place, such as a local database or filesystem, and the other system defaults work well for your application, you need not concern yourself with EveContext. In that case, you simply record your connection details in `Eve.properties` and let the system handle the rest.

However, if you need to access objects stored in more than one location, you need to use an EveContext object to override the settings in `Eve.properties` to direct your method to the appropriate storage location.

See the *Installation Guide* for more information on the `Eve.properties` file, which includes an explanation of each of the properties that you can set in an `EveContext` object.

To create a new EveContext object, you subclass EveContext directly, like so:

```
foo = new EveContext();
```

EvePatch

This interface exists to enable eVision to provide additional functionality and upgrades in the future through patches. The methods in this interface allow you to make updates to individual MediaObjects and even entire MediaCollections.

Note • This is not a supported API interface. eVision will not provide technical support or information regarding the `EvePatch` interface.

FrameGrabber

Overview

The methods in this interface allow you to extract information from video files. Specifically, you can:

- determine the keyframes (scene changes) within a video file
- display a keyframe
- index keyframes
- convert frames/keyframes into a MediaObject

This interface includes the following methods:

close

format	`void close()`
description	Closes the video file.
parameters	none
returns	nothing (void)
throws	The `close()` method throws an EveException if the close operation fails.

getMediaObject

format	`MediaObject getMediaObject(int frameNumber) throws EveException`
description	Converts the specified frame from a video file into a MediaObject.
parameters	`frameNumber` — the number of the frame within a video file that you want to convert into a MediaObject.
returns	a MediaObject containing information about the specified frame
throws	The `getMediaObject()` method throws an EveException if the retrieval fails.

getControlComponent

format	`Component getControlComponent()`
description	Retrieves the controls (such as volume settings) of the appropriate video player, if available.
parameters	none
returns	nothing (void)
throws	The `getControlComponent()` method does not throw any exceptions.

getVisualComponent

format	`Component getVisualComponent()`
description	Retrieves the component of the appropriate video player that reads and plays the current video file.
parameters	none
returns	nothing (void)
throws	The `getVisualComponent()` method does not throw any exceptions.

getTotalDistance

format	`double getTotalDistance()`
description	Determines the visual similarity between the current and previous frame. The value defined for the `frameGrabberStep` parameter in the `eve.properties` file determines which frames in a video file are checked by this method. If the visual similarity between frames is greater than the value defined for the `frameGrabberMinimumDistance` parameter in the `eve.properties` file, then the current frame is considered a keyframe.
parameters	none
returns	a double-precision floating-point value (0-100) representing the visual similarity between the current frame and the previous frame
throws	The `double getTotalDistance()` method does not throw any exceptions.

getTotalFrames

format	`int getTotalFrames()`
description	Determines the total number of frames within a video file.
parameters	none
returns	an integer containing the the total number of frames in a video file
throws	The `getTotalFrames()` method does not throw any exceptions.

gotKeyFrame

format	`boolean gotKeyFrame(int frameNumber) throws EveException`
description	Determines the visual similarity between the specified frame and the previous frame in a video file. If the visual similarity is greater than the value defined for the `frameGrabberMinimumDistance` parameter in the `eve.properties` file, then the current frame is considered a keyframe.
parameters	■ `frameNumber` — the number of the frame whose visual similarity to the previous frame you want to check
returns	■ `TRUE` — if there was enough visual difference between frames to indicate that the current frame is a keyframe ■ `FALSE` — if the current frame is not considered a keyframe
throws	The `gotKeyFrame()` method throws an EveException if the retrieval fails.

gotoFrame

format	`void gotoFrame(int frameNumber) throws EveException;`
description	Jumps to the specified frame within the video file.
parameters	■ `frameNumber` — the number of the frame to which you want to jump
returns	nothing (void)
throws	The `gotoFrame()` method throws an EveException if the retrieval fails.

open

format	`void open(String path) throws EveException`
description	Opens the video file stored in the specified location.
parameters	■ `path` — the full path and file name of the video to be opened
returns	nothing (void)
throws	The `open()` method throws an EveException if the open operation fails.

play

format	`play()`
description	Uses the Java™ Media Framework API (JMF) to read and play the video file.
parameters	none
returns	the appropriate video player
throws	The `play()` method does not throw any exceptions.

setContext

format	`void setContext(EveContext eveContext)`
description	Sets the FrameGrabber's context information. For more information on EveContext, see the ***EveContext*** section.
parameters	■ `context` — the context of the object to be analyzed
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

ImageManager

The methods in this interface allow you to manipulate image files on disk. You can load, save, and resize images using these methods.

You could use this interface, for example, to implement bulk image-preparation routines. In order to maximize the effectiveness of analyzing an image collection, you could use the methods in `ImageManager` to resize all the images in the collection to the same dimensions beforehand.

To create a new ImageManager object, you use the `Eve.newImageManager()` method. This method optionally takes an EveContext object as an argument.

This interface includes the following methods:

getImage

format	`Image getImage() throws EveException`
description	Retrieves an image from the buffer.
parameters	none
returns	displays the image
throws	The `getImage()` method throws an `EveException` if the retrieval fails.

getImage (`MediaObject`)

format	`Image getImage(MediaObject input) throws EveException`
description	Retrieves the image (.jpg) stored within a MediaObject.
parameters	`input` — the MediaObject whose image you wish to retrieve
returns	displays the image
throws	The `getImage()` method throws an `EveException` if the retrieval fails.

getImageIcon (`MediaObject`)

format	`ImageIcon getImageIcon(MediaObject input) throws EveException`
description	Retrieves the image (.jpg) stored within a MediaObject. The image is retrieved as an icon and can be used within an application.
parameters	`input` — the MediaObject whose image you wish to retrieve as an icon
returns	an icon representing the image stored in the MediaObject
throws	The `getImageIcon()` method throws an `EveException` if the retrieval fails.

getSegmentationMask

format	`byte[] getSegmentationMask(MediaObject input) throws EveException`
description	Retrieves a MediaObject's segmentation mask.
parameters	■ `input` — the MediaObject whose segmentation mask you wish to retrieve
returns	a byte array containing the segmentation mask
throws	The `getSegmentationMask()` method throws an `EveException` if the retrieval fails.

getSegmentationMaskIcon

format	`ImageIcon getSegmentationMaskImageIcon(MediaObject input) throws EveException`
description	Retrieves a MediaObject's segmentation mask. The segmentation mask is retrieved as an icon and can be used within an application.
parameters	■ `input` — the MediaObject whose segmentation mask you wish to retrieve as an icon
returns	an icon representing the MediaObject's segmentation mask
throws	The `getSegmentationMaskIcon()` method throws an `EveException` if the retrieval fails.

loadImage

format	`boolean loadImage(String path) throws EveException`
description	Loads an image into eVe for processing.
parameters	■ `path` — the full path and filename of the image to load
returns	■ `TRUE` — if the load was successuful ■ `FALSE` — if the load was unsuccessful
throws	The `loadImage()` method throws an `EveException` if the loading fails.

newMediaObject

format	`MediaObject newMediaObject() throws EveException`
description	Creates a new MediaObject. This is useful, for example, if you wish to generate a query image on-the-fly.
parameters	none
returns	an empty MediaObject
throws	The `newMediaObject()` method throws an `EveException` if the creation fails.

resize

format	`boolean resize(int maxWidthOrHeight) throws EveException`
description	Resizes an image to `maxwidthOrHeight` pixels square. This method reproportions the image so that it is a square in order to facilitate image analysis.
parameters	■ `maxWidthorHeight` — the new square dimension to which to resize the image
returns	■ `TRUE` — if the resize was successuful ■ `FALSE` — if the resize failed
throws	The `resize()` method throws an `EveException` if the resize fails.

saveImage

format	`boolean saveImage(String path) throws EveException`
description	Saves an image to disk.
parameters	■ `path` — the full path and filename of the image to create on disk
returns	■ `TRUE` — if the save was successuful ■ `FALSE` — if the save failed
throws	The `saveImage()` method throws an `EveException` if the save fails.

setContext

format	`void setContext(EveContext context)`
description	Sets the ImageManager's context information. For more information on EveContext, see the ***EveContext*** section.
parameters	■ `context` — the context of the object to be analyzed
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

supportedImageTypes

format	`String[] supportedImageTypes()`
description	Returns the list of image types that ImageManager can process. ImageManager currently supports 68 different image file formats.
parameters	none
returns	an array of strings containing the names of the supported file formats
throws	The `supportedImageTypes()` method does not throw any exceptions.

MediaCollection

The methods in this interface allow you to perform a wide variety of operations on MediaCollections.

To create a new MediaCollection object, you use the `Eve.newMediaCollection()` method, which returns an empty MediaCollection object. This method optionally takes an EveContext object as an argument.

Note • Each method within the `MediaCollection` interface is available in two formats: with before/after commands and without before/after commands. If you do not plan to use before/after commands within your method calls, use the first entry in the format section for each of the methods described below. See the *Using Before and After Commands* section for information about how to use the format containing the before and after commands within a method.

add

format	■ `void add(MediaObject eve) throws EveException` ■ `void add(MediaObject eve,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Adds a MediaObject to the MediaCollection.
parameters	■ `eve` — the MediaObject to be added ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	■ `TRUE` — if the addition was successful
throws	The `add()` method throws an EveException if the addition fails.

Note • After you add in image to a MediaCollection using the `add()` method, you must call the *reorganize* method to organize and optimize the data structures within the MediaCollection.

analyze

format	■ `void analyze(MediaObject mediaObject) throws EveException` ■ `void analyze(MediaObject mediaObject,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Analyzes a given `MediaObject` and replaces it in its containing MediaCollection. You do not have to have a MediaCollection open to perform the analysis.
parameters	■ `mediaObject` — the object to be analyzed ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	■ `TRUE` — if the analysis was successful ■ `FALSE` — if the analysis failed
throws	The `analyze()` method throws an EveException if the analysis fails.

close

format	■ `void close() throws EveException` ■ `void close(CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Closes the current MediaCollection.
parameters	■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	nothing (void)
throws	The `close()` method throws an EveException if the close operation fails.

create

format	■ `void create(String path) throws EveException` ■ `void create(String path,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Creates a new MediaCollection in the specified location.
parameters	■ `path` — the location in which to create the new MediaCollection ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	nothing (void)
throws	The `create()` method throws an EveException if the creation fails.

delete (`MediaObject`)

format	■ `void delete(MediaObject eve) throws EveException` ■ `void update(MediaObject eve,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Removes a MediaObject from the MediaCollection.
parameters	■ `eve` — the MediaObject to be removed ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	■ `TRUE` — if the deletion was successful
throws	The `delete()` method throws an EveException if the deletion fails.

Note • After you delete an image from a MediaCollection using the `delete(MediaObject)` method, you must call the *reorganize* method to organize and optimize the remaining data structures within the MediaCollection.

delete (long)

format	■ `void delete(long eveMediaObjectKey) throws EveException` ■ `void delete(long eveMediaObjectKey,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Removes a MediaObject from the MediaCollection.
parameters	■ `eveMediaObjectKey` — the key of the MediaObject to be removed ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	■ `TRUE` — if the deletion was successful
throws	The `delete()` method throws an EveException if the deletion fails.

Note • After you delete an image from a MediaCollection using the `delete(long)` method, you must call the *reorganize* method to organize and optimize the remaining data structures within the MediaCollection.

getCollectionName

format	■ `String getCollectionName() throws EveException` ■ `String getCollectionName(CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Returns the name of the MediaCollection.
parameters	■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	a string containing the name of the collection
throws	The `getCollectionName()` method throws an EveException if the retrieval fails.

getKeys

format	■ `long[] getKeys() throws EveException` ■ `long[] getKeys(CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Gets all the MediaObject keys in the MediaCollection.
parameters	■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	an array of `long` integers containing all the keys
throws	The `getKeys()` method throws an EveException if the retrieval fails.

getMediaObject (`long`)

format	■ `MediaObject getMediaObject(long key) throws EveException` ■ `MediaObject getMediaObject(long key,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Retrieves the MediaObject with the given key.
parameters	■ `key` — the index key of the MediaObject you wish to retrieve ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	the requested MediaObject
throws	The `getMediaObject()` method throws an EveException if the retrieval fails.

getMediaObject (`long[]`)

format	■ `MediaObject[] getMediaObject(long keys[]) throws EveException` ■ `MediaObject[] getMediaObject(long keys[],CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Retrieves the MediaObjects with the given keys.
parameters	■ `keys` — an array of index keys ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	an array of the requested MediaObjects
throws	The `getMediaObject()` method throws an EveException if the retrieval fails.

getMediaObject (`SearchResults`)

format	■ `MediaObject getMediaObject(SearchResults searchResults) throws EveException` ■ `MediaObject getMediaObject(SearchResults searchResult,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Retrieves the MediaObject referenced by the `SearchResults` object.
parameters	■ `searchResults` — a `SearchResults` object that references a MediaObject ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	the requested MediaObject, if it exists
throws	The `getMediaObject()` method throws an EveException if the retrieval fails.

getMediaObject (SearchResults[])

format
- `MediaObject[] getMediaObject(SearchResults searchResults[]) throws EveException`
- `MediaObject[] getMediaObject(SearchResults searchResults[],CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Retrieves the MediaObjects referenced by the SearchResults objects.

parameters
- `searchResults` — an array of `SearchResults` objects that reference MediaObjects
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of the requested MediaObjects

throws The `getMediaObject()` method throws an EveException if the retrieval fails.

getMetadataKeys

format
- `String[] getMetadataKeys() throws EveException`
- `String[] getMetadataKeys(CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Returns all the unique metadata keys in the MediaCollection.

parameters
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of strings containing the keys of all the MediaCollection's metadata

throws The `getMetadataKeys()` method throws an EveException if the retrieval fails.

getMetadataValues

format

- `String[] getMetadataValues(String key) throws EveException`
- `String[] getMetadataValues(String key,CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Returns all the unique values for the key you specify in the MediaCollection. 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".

parameters

- `key` — the name of the key containing the values you want to retrieve
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of strings containing the values of all the MediaCollection's metadata keys

throws The `getMetadataValues()` method throws an EveException if the retrieval fails.

getProperties

format

- `Hashtable getProperties() throws EveException`
- `Hashtable getProperties(CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Returns all the properties associated with the MediaCollection. Currently the only property set is the MediaCollection's name, but you can set any serializable object as a property.

parameters

- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns a hashtable containing all the properties associated with the MediaCollection

throws The `getProperties()` method throws an EveException if the retrieval fails.

getProperty

format
- `Object getProperty(String key) throws EveException`
- `Object getProperty(String key,CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Retrieves a property object from the MediaCollection. Because MediaCollections may contain multiple property items with the same key, as long as the values are different, getProperty() returns an object that may contain multiple items.

parameters
- `key` — the key of the metadata you wish to retrieve
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns The requested metadata, if it exists.

throws The `getProperty()` method throws an EveException if the retrieval fails.

metadataFind (`String`)

format
- `SearchResults[] metadataFind(String key) throws EveException`
- `SearchResults[] metadataFind(String key,CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Finds metadata items with a particular key in all of the MediaObjects in the MediaCollection.

parameters
- `key` — the key of the metadata items you wish to retrieve
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of `SearchResults` objects

throws The `metadataFind()` method throws an EveException if the operation fails.

metadataFind (String, String)

format	■ `SearchResults[] metadataFind(String key, String value) throws EveException` ■ `SearchResults[] metadataFind(String key,String value,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Finds all MediaObjects containing a particular key-value pair in the MediaCollection.
parameters	■ `key` — the key of the metadata you wish to retrieve ■ `value` — the text of the metadata you wish to retrieve ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	an array of `SearchResults` objects
throws	The `metadataFind()` method throws an EveException if the operation fails.

open

format	■ `void open(String path) throws EveException` ■ `void open(String path,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Opens the MediaCollection contained in the directory stored in `path`.
parameters	■ `path` — the path to the directory in which the MediaCollection is stored\ ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	nothing (void)
throws	The `open()` method throws an `EveException` if the open operation fails.

reorganize

format

- `void reorganize() throws EveException`
- `void reorganize(CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Optimizes the data structures within the MediaCollection to allow for faster searching. When you add items to a MediaCollection, no attempt is made to optimize their placement within the data store. However, after you have added a large number of MediaObjects, you should call `reorganize()` in order to optimize the indexes.

parameters

- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns nothing (void)

throws The `reorganize()` method throws an EveException if the reorganization fails.

save

format `void save() throws EveException`

description Use this method to persist the current MediaCollection to disk *while* you are making changes to that MediaCollection and before you invoke the `close()` method.

For example, if the server stops or eVe processing gets interrupted during the update of a MediaCollection, any changes made to that MediaCollection will not be saved. If you invoke this method while changes are being made to a MediaCollection, those changes will be saved.

The `close()` method automatically calls the `save()` method when invoked.

parameters none

returns nothing (void)

throws The `save` method throws an EveException if the save fails.

search (MediaObject, SearchParameters)

format

- `SearchResults[] search(MediaObject eve,SearchParameters parameters) throws EveException`
- `SearchResults[] search(MediaObject eve,SearchParameters parameters,CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Performs a search against the MediaCollection using the given MediaObject as a source. For more information on SearchParameters objects, see the ***SearchParameters*** section.

parameters

- `eve` — the MediaObject to use as the source image for the search
- `parameters` — the `SearchParameters` object containing the arguments to the search
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of `SearchResults` objects

throws The `search()` method throws an EveException if the search fails.

search (MediaObject, SearchParameters, SearchResults[])

format

- `SearchResults[] search(MediaObject eve,SearchParameters parameters, SearchResults target[]) throws EveException`
- `SearchResults[] search(MediaObject eve,SearchParameters parameters,SearchResults target[], CommandList beforeMethods,CommandList afterMethods) throws EveException`

description Performs a search against a set of search results using the given MediaObject and search parameters. For more information on `SearchParameters` objects, see the ***SearchParameters*** section.

parameters

- `eve` — the MediaObject to use as the source image for the search
- `parameters` — the `SearchParameters` object containing the arguments to the search
- `target` — the array of SearchResults against which to perform the search
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns an array of `SearchResults` objects

throws The `search()` method throws an EveException if the search fails.

search (MediaObject, SearchParameters, MediaObject[])

format

- `SearchResults[] search(MediaObject eve,SearchParameters parameters, MediaObject target[]) throws EveException`
- `SearchResults[] search(MediaObject eve,SearchParameters parameters,MediaObject target[], CommandList beforeMethods,CommandList afterMethods) throws EveException`

description

Performs a search against a list of MediaObjects, using the given MediaObject and search parameters. For more information on SearchParameters objects, see the ***SearchParameters*** section.

parameters

- `eve` — the MediaObject to use as the source image for the search
- `parameters` — the `SearchParameters` object containing the arguments to the search
- `target` — the list of MediaObjects against which to perform the search
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns

an array of `SearchResults` objects

throws

The `search()` method throws an EveException if the search fails.

setCollectionName

format

- `void setCollectionName(String collectionName) throws EveException`
- `void setCollectionName(String collectionName,CommandList beforeMethods,CommandList afterMethods) throws EveException`

description

Sets the name of the MediaCollection to `collectionName`.

parameters

- `collectionName` — the new name for the MediaCollection
- `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked
- `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed

returns

nothing (void)

throws

The `setCollectionName()` method throws an EveException if the naming fails.

setContext

format	`void setContext(EveContext context)`
description	Sets the context information for the MediaCollection. If you are using only one database, you likely do not need to use this method. However, if you are using more than one database, you need to use an EveContext object to make sure the system knows where to find your MediaCollection. For more information on EveContext, see the ***EveContext*** section.
parameters	■ `context` — the context of the MediaCollection
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

setProperty

format	■ `void setProperty(String key, Object value)` ■ `void setProperty(String key,Object value,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Sets a property in a MediaCollection. Currently the only property set is the MediaCollection's name, but you can set any serializable object as a property.
parameters	■ `key` — the key of the item you wish to set ■ `value` — the contents of the property ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	nothing (void)
throws	The `setProperty()` method throws an EveException if the set operation fails.

size

format	■ `int size() throws EveException` ■ `int size(CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Reports the number of MediaObjects in the MediaCollection.
parameters	■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	an integer representing the number of MediaObjects in the collection
throws	The `size()` method throws an EveException if the operation fails.

update

format	■ `void update(MediaObject eve) throws EveException` ■ `void update(MediaObject eve,CommandList beforeMethods,CommandList afterMethods) throws EveException`
description	Replaces a MediaObject in the MediaCollection with a new MediaObject of the same name.
parameters	■ `eve` — the MediaObject to be updated ■ `CommandList beforeMethods` — indicates that you want to run a command method(s) before the method is invoked ■ `CommandList afterMethods` — indicates that you want to run a command method(s) after the method is executed
returns	■ `TRUE` — if the update was successful
throws	The `update()` method throws an EveException if the update fails.

Using Before and After Commands

The methods in the `MediaCollection` interface allow you to execute commands before and after the action performed by those methods. To do this, each method within `MediaCollection` is provided in two formats: one with the before/after functionality and one without. The one that contains before/after functionality accepts the `CommandList beforeMethods` and `CommandList afterMethods` parameters. These parameters call a command list before or after a method is run and execute the commands(s) included in that list.

See the *Code Samples* section in the *Getting Started Guide* for an example of how to use before and after commands within MediaCollection methods.

MediaObject

This interface provides access to all functionality associated with MediaObjects. You use the methods in this interface to create, manipulate, and update all of the components in a MediaObject.

To create a new MediaObject, you use the `Eve.newMediaObject()` method, which returns an empty MediaObject. This method optionally takes an EveContext object as an argument.

addMetadata

format	`boolean addMetadata(String key, String value)`
description	Adds a metadata item to the MediaObject.
parameters	■ `key` — the unique key of the metadata item ■ `value` — the value of the metadata item
returns	■ `TRUE` — if the addition was successful ■ `FALSE` — if the addition failed
throws	The `addMetadata()` method does not throw any exceptions.

applyPatch

format	`boolean applyPatch(EvePatch patch)`
description	Applies an EvePatch to the MediaObject. EvePatches are not supported and are for eVision use only. For more information, see the ***EvePatch*** section.

deleteMetadata

format	`boolean deleteMetadata`
description	Removes all metadata from the MediaObject.
parameters	none
returns	■ `TRUE` — if the deletion was successful ■ `FALSE` — if the deletion failed
throws	The `deleteMetadata()` method does not throw any exceptions.

deleteMetadata (String)

format	`boolean deleteMetadata(String key)`
description	Removes a metadata item from the MediaObject.
parameters	■ `key` — the unique key of the metadata item
returns	■ `TRUE` — if the deletion was successful ■ `FALSE` — if the deletion failed
throws	The `deleteMetadata()` method does not throw any exceptions.

deleteMetadata (String, String)

format	`boolean deleteMetadata(String key, String value)`
description	Removes a metadata item from the MediaObject.
parameters	■ `key` — the unique key of the metadata item ■ value — the value of the metadata item you wish to delete
returns	■ `TRUE` — if the deletion was successful ■ `FALSE` — if the deletion failed
throws	The `deleteMetadata()` method does not throw any exceptions.

getBlueChannel

format	`double[][] getBlueChannel()`
description	Retrieves the contents of the image's blue channel as an array of pixel values.
parameters	none
returns	a two-dimensional array of double-precision floating-point pixel values
throws	The `getBlueChannel()` method does not throw any exceptions.

getCollectionName

format	`String getCollectionName()`
description	Retrieves the name of the MediaCollection in which the MediaObject is stored.
parameters	none
returns	a string containing the name of the MediaCollection
throws	The `getCollectionName()` method does not throw any exceptions.

getGreenChannel

format	`double[][] getGreenChannel()`
description	Retrieves the contents of the image's green channel as an array of pixel values.
parameters	none
returns	a two-dimensional array of double-precision floating-point pixel values
throws	The `getGreenChannel()` method does not throw any exceptions.

getHeight

format	`int getHeight()`
description	Retrieves the height of the MediaObject's image.
parameters	none
returns	an integer containing the height of the image
throws	The `getHeight()` method does not throw any exceptions.

getIndex

format	`Vector getIndex(int indexType)`
description	Retrieves the image's index signature of the requested type from the MediaObject.
parameters	■ `indexType` — the type of signature to retrieve, such as `Eve.COLOR` or `Eve.TEXTURE`
returns	the requested index signature
throws	The `getIndex()` method does not throw any exceptions.

getKey

format	`long getKey()`
description	Retrieves the MediaObject's index key. This key uniquely identifies the MediaObject within the MediaCollection, and thus any MediaObject can be uniquely identified within an eVe system by referring to its key and the key of the MediaCollection in which it is stored.
parameters	none
returns	a long integer containing the MediaObject's index key
throws	The `getKey()` method does not throw any exceptions.

getMetadata

format	`Vector getMetadata()`
description	Retrieves all of the MediaObject's metadata.
parameters	none
returns	a vector containing the MediaObject's metadata items
throws	The `getMetadata()` method does not throw any exceptions.

getMetadata (`String, String`)

format	`Metadata getMetadata(String key, String value)`
description	Retrieves the specified metadata item, if it exists.
parameters	■ `key` — the key of the metadata you wish to retrieve ■ `value` — the text value of the metadata you wish to retrieve
returns	the requested metadata, if it exists; otherwise, the method returns `NULL`.
throws	The `getMetadata()` method does not throw any exceptions.

getProperties

format	`Hashtable getProperties()`
description	Retrieves all of a MediaObject's properties.
parameters	none
returns	a hashtable containing all of the MediaObject's metadata items
throws	The `getProperties()` method does not throw any exceptions.

getProperty

format	`Object getProperty(String key)`
description	Retrieves a property from the MediaObject.
parameters	■ `key` — the unique key of the property
returns	an object containing the requested property item
throws	The `getProperty()` method does not throw any exceptions.

getRedChannel

format	`double[][] getRedChannel()`
description	Retrieves the contents of the image's red channel as an array of pixel values.
parameters	none
returns	a two-dimensional array of double-precision floating-point pixel values
throws	The `getRedChannel()` method does not throw any exceptions.

getWidth

format	`int getWidth()`
description	Retrieves the width of the MediaObject's image.
parameters	none
returns	an integer containing the width of the image
throws	The `getWidth()` method does not throw any exceptions.

loadFrom

format	`MediaObject loadFrom(String path)`
description	Loads a MediaObject from an EDF file on disk. EDF files are MediaObjects that have been saved to disk.
parameters	■ `path` — the path and filename of the EDF file
returns	a MediaObject loaded from the EDF file located at `path`
throws	The `loadFrom()` method does not throw any exceptions.

loadImage

format	`boolean loadImage(String path)`
description	Loads an image into the MediaObject. The image is loaded as a property of the MediaObject.
parameters	■ `path` — the path and filename of the image you wish to load
returns	■ `TRUE` – if the load was successful ■ `FALSE` — if the load failed
throws	The `loadImage()` method does not throw any exceptions.

makeArray

format	`MediaObject[] makeArray(int length)`
description	Creates a single-dimensional array of MediaObjects of the specified length. You can use this method, for example, to create an array with which to populate a new MediaCollection.
parameters	■ `length` — the number of elements to create in the array
returns	a one-dimensional array of MediaObjects of the specified length.
throws	The `makeArray()` method does not throw any exceptions.

purge

format	`void purge()`
description	Cleans up a MediaObject. The `purge()` method removes everything from the MediaObject that is not needed for analysis.
parameters	none
returns	nothing (void)
throws	The `purge()` method does not throw any exceptions.

saveTo

format	`boolean saveTo(String path)`
description	Writes the MediaObject to an EDF file on disk to support serialization.
parameters	■ `path` — the path and filename of the EDF file
returns	■ `TRUE` — if the save was successful ■ `FALSE` — if the save failed
throws	The `saveTo()` method does not throw any exceptions.

setCollectionName

format	`void setCollectionName(String collectionName)`
description	Sets the name of the MediaCollection in which the MediaObject is stored.
parameters	■ `collectionName` — the MediaCollection's name
returns	nothing (void)
throws	The `setCollectionName()` method does not throw any exceptions.

setColorPlanes

format	`boolean setColorPlanes(double red[][],double green[][],double blue[][])`
description	Allows you to manually create an image by specifying pixel maps for each of its three color planes.
parameters	■ `red` — the red color plane ■ `green` — the green color plane ■ `blue` — the blue color plane
returns	■ `TRUE` — if the operation was successful ■ `FALSE` — if the operation failed
throws	The `setColorPlanes()` method does not throw any exceptions.

setContext

format	`void setContext(EveContext eveContext)`
description	Sets the context information for the MediaObject. For more information on EveContext, see the ***EveContext*** section.
parameters	■ `eveContext` — the context of the MediaObject
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

setHeight

format	`boolean setHeight(int height)`
description	Sets the height of the MediaObject's image.
parameters	■ `height` — the new height of the image
returns	■ `TRUE` — if the image was successfully resized ■ `FALSE` — if the resize failed
throws	The `setHeight()` method does not throw any exceptions.

setIndex

format	`boolean setIndex(int indexType, Vector indexes)`
description	Sets an index directly within the MediaObject.
parameters	■ `indexType` — the type of index, such as `Eve.REGION or Eve.TEXTURE` ■ `indexes` — the vector containing the index values
returns	■ `TRUE` — if the assignment was successful ■ `FALSE` — if the assignment failed
throws	The `setIndex()` method does not throw any exceptions.

setKey

format	`boolean setKey(long key)`
description	Sets the MediaObject's index key. This key uniquely identifies the MediaObject within the MediaCollection, and thus any MediaObject can be uniquely identified within an eVe system by referring to its key and the key of the MediaCollection in which it is stored.
parameters	■ `key` — the MediaObject's unique index key
returns	■ `TRUE` — if the key was successfully assigned ■ `FALSE` — if the key was not assigned
throws	The `setKey()` method does not throw any exceptions.

setProperty

format	`boolean setProperty(String key, Serializable value)`
description	Sets the given property for the MediaObject.
parameters	■ `key` — the unique key of the property ■ `value` — the new contents of the property
returns	■ `TRUE` — if the assignment was successful ■ `FALSE` — if the assignment failed
throws	The `setProperty()` method does not throw any exceptions.

setWidth

format	`boolean setWidth(int width)`
description	Sets the width of the MediaObject's image.
parameters	■ `width` — the new width of the image
returns	■ `TRUE` — if the image was successfully resized ■ `FALSE` —if the resize failed
throws	The `setWidth()` method does not throw any exceptions.

updateMetadata

format	`boolean updateMetadata(String key, String newValue)`
description	Assigns a new value to an existing metadata item.
parameters	■ `key` — the unique key of the metadata item ■ `newValue` — the value to be assigned to the metadata item
returns	■ `TRUE` —if the update was successful ■ `FALSE` —if the update failed
throws	The `updateMetadata()` method does not throw any exceptions.

Metadata

This interface provides access to all functionality associated with metadata. You use the methods in this interface to create, manipulate, and update metadata.

To create a new Metadata object, you use the `Eve.newMetadata()` method, which returns an empty Metadata object.

getCollectionName

format	`String getCollectionName()`
description	Gets the name of the MediaCollection in which the metadata is stored.
parameters	none
returns	a string containing the name of the MediaCollection
throws	The `getCollectionName()` method does not throw any exceptions.

getID

format	`long getID()`
description	Gets the unique ID of the metadata item.
parameters	none
returns	a long integer containing the ID of the metadata item
throws	The `getID()` method does not throw any exceptions.

getKey

format	`String getKey()`
description	Gets the key of the metadata item.
parameters	none
returns	a string containing the key of the metadata item
throws	The `getKey()` method does not throw any exceptions.

getValue

format	`String getValue()`
description	Gets the value of the metadata item.
parameters	none
returns	a string containing the value of the metadata item
throws	The `getValue()` method does not throw any exceptions.

setCollectionName

format	`void setCollectionName(String name)`
description	Sets the name of the MediaCollection in which the metadata item is stored.
parameters	■ `name` — the name of the containing MediaCollection
returns	nothing (void)
throws	The `setCollectionName()` method does not throw any exceptions.

setID

format	`void setID(long id)`
description	Sets the unique ID of the metadata item.
parameters	■ `id` — the unique ID of the metadata item
returns	nothing (void)
throws	The `setID()` method does not throw any exceptions.

setKey

format	`void setKey(String key)`
description	Sets the key of the metadata item.
parameters	■ `key` — the key of the metadata item
returns	nothing (void)
throws	The `setKey()` method does not throw any exceptions.

setValue

format	`void setValue(String value)`
description	Sets the value of the metadata item.
parameters	■ `value` — the value of the metadata item
returns	nothing (void)
throws	The `setValue()` method does not throw any exceptions.

SearchParameters

This interface provides access to all functionality associated with parameters for searching. You use the methods in this interface to create, manipulate, and update SearchParameters objects.

To create a new SearchParameters object, you use the `Eve.newSearchParameters()` method, which returns an empty SearchParameters object.

getAscending

format	`boolean getAscending()`
description	Reports whether search results will be ordered in ascending or descending order. Normally, results are returned in descending order.
parameters	none
returns	■ `TRUE` — if search results will be returned in ascending order ■ `FALSE` — if search results will be returned in descending order
throws	The `getAscending()` method does not throw any exceptions.

getSearch

format	`boolean getSearch(int indexType)`
description	Determines if a particular type of search is enabled. For example, you would call `getSearch(Eve.COLOR)` to determine if a color search is enabled in the object. You set the weighting of a particular type of search with the `setSearch()` methods. See also ***setSearch (int, boolean, double)***.
parameters	■ `indexType` — the type of search for which to query
returns	■ `TRUE` — if the search is enabled ■ `FALSE` — if the search is disabled
throws	The `getSearch()` method does not throw any exceptions.

getWeight

format	`double getWeight(int indexType)`
description	Reports the weighting of a particular search type, such as region or texture.
parameters	■ `indexType` — the type of search for which to query, such as `Eve.COLOR`
returns	a double-precision floating-point value between 0 and 1.0 that represents the weighting of the search type
throws	The `getWeight()` method does not throw any exceptions.

setAscending

format	`void setAscending(boolean value)`
description	Instructs the search to order results in ascending or descending order. By default, results are returned in descending order.
parameters	■ `value` — set this to `TRUE` to sort search results in ascending order
returns	nothing (void)
throws	The `setAscending()` method does not throw any exceptions.

setSearch (`int, boolean, double`)

format	`void setSearch(int indexType, boolean value, double weight)`
description	Sets the search options for the SearchParameters object. 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.
parameters	■ `indextype` — the type of search to perform, such as `Eve.REGION` ■ `value` — set to `TRUE` if you want to include this type of search in your query ■ `weight` — the relative importance of this particular search type, on a scale of 0 to 1.0
returns	nothing (void)
throws	The `setSearch()` method does not throw any exceptions.

setSearch (int, double)

format	`void setSearch(int indexType, double weight)`
description	Sets the search options for the SearchParameters object. 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.
parameters	■ `indextype` — the type of search to perform ■ `weight` — the relative importance of this particular search type, on a scale of 0 to 1.0
returns	nothing (void)
throws	The `setSearch()` method does not throw any exceptions.

SearchResults

This interface provides access to all functionality associated with the results of a search. You use the methods in this interface to create, manipulate, and update SearchResults objects.

To create a new SearchResults object, you use the `Eve.newSearchResults()` method, which returns an empty SearchResults object. This method optionally takes an EveContext object as an argument.

and

format	`SearchResults[] and(SearchResults arg1[], SearchResults arg2[])`
description	Performs a logical AND operation on two arrays of SearchResults objects.
parameters	■ `arg1` — the first array of SearchResults ■ `arg2` — the second array of SearchResults
returns	a one-dimensional array of SearchResults objects
throws	The `and()` method does not throw any exceptions.

append

format	`SearchResults[] append(SearchResults arg1[], SearchResults arg2[])`
description	Concatenates two arrays of SearchResults.
parameters	■ `arg1` — the first array of SearchResults ■ `arg2` — the second array of SearchResults
returns	a one-dimensional array of SearchResults objects
throws	The `append()` method does not throw any exceptions.

chop

format	`SearchResults[] chop(SearchResults arg1[], int maxLength)`
description	Truncates the array of SearchResults to the given length. If the array is already that length or shorter, it is returned unmodified.
parameters	■ `arg1` — the array of SearchResults ■ `maxLength` — the length to which to truncate the array
returns	a one-dimensional array of SearchResults objects with `maxLength` or fewer members
throws	The `chop()` method does not throw any exceptions.

distanceSort

format	`SearchResults[] distanceSort(int indexType, SearchResults arg1[])`
description	Takes an unordered array of SearchResults objects and sorts it according to distance. You can only sort based on one index type at a time. For example, you sort an array based on *texture* distance like this: `SearchResults sortedArray = distanceSort( Eve.TEXTURE, results[] );`
parameters	■ `indexType` — the type of index (such as `Eve.REGION`) to which the distance applies ■ `arg1` —the array of SearchResults to be sorted
returns	an array of SearchResults objects sorted in descending order of the distance specified in `indexType`
throws	The `distanceSort()` method does not throw any exceptions.

getCollectionName

format	`String getCollectionName()`
description	Gets the name of the MediaCollection being searched.
parameters	none
returns	a string containing the name of the MediaCollection
throws	The `getCollectionName()` method does not throw any exceptions.

getDistance

format	`double getDistance(int indexType)`
description	Gets the current result's distance from the target image. This only applies to one index type at a time. For example, you could get the result's *texture* distance from the target.
parameters	■ `indexType` — the type of index (such as `Eve.REGION`) to which the distance applies
returns	a double-precision floating-point value representing the current result's distance from the target image
throws	The `getDistance()` method does not throw any exceptions.

getKey

format	`long getKey()`
description	Gets the key of the MediaObject to which the SearchResults object refers. SearchResults objects are persistent, and the object's key provides a way to retrieve it later on.
parameters	none
returns	a long integer containing the object's unique key
throws	The `getKey()` method does not throw any exceptions.

getRank

format	`int getRank()`
description	Retrieves the current result's rank relative to the other results of the search.
parameters	none
returns	an integer representing the item's rank
throws	The `getRank()` method does not throw any exceptions.

getSimilarity

format	`double getSimilarity()`
description	Returns the similarity score of the search result against the target image.
parameters	none
returns	a double-precision floating-point value representing the degree of similarity between the search result and the target image
throws	The `getSimilarity()` method does not throw any exceptions.

makeArray (int)

format	`SearchResults[] makeArray(int length)`
description	Creates a one-dimensional array of SearchResults objects of the specified length.
parameters	■ `length` — the length of the new array
returns	a one-dimensional array of SearchResults objects
throws	The `makeArray()` method does not throw any exceptions.

makeArray (long[])

format	`SearchResults[] makeArray(long keys[])`
description	Creates a one-dimensional array of SearchResults objects and populates it with the given objects.
parameters	■ `keys` — an array of keys of SearchResults objects
returns	a one-dimensional array of SearchResults objects
throws	The `makeArray()` method does not throw any exceptions.

normalize

format	`SearchResults[] normalize(SearchParameters parameters, SearchResults results[])`
description	Normalizes a set of SearchResults and its associated parameters.
parameters	■ `parameters` — the search parameters to normalize ■ `results` — the array of SearchResults to be normalized
returns	a one-dimensional array of SearchResults objects
throws	The `normalize()` method does not throw any exceptions.

not

format	`SearchResults[] and(SearchResults arg1[], SearchResults arg2[])`
description	Performs a logical NOT operation on two arrays of SearchResults objects.
parameters	■ `arg1` — the first array of SearchResults ■ `arg2` — the second array of SearchResults
returns	a one-dimensional array of SearchResults objects
throws	The `not()` method does not throw any exceptions.

or

format	`SearchResults[] or(SearchResults arg1[], SearchResults arg2[])`
description	Performs a logical OR operation on two arrays of SearchResults objects.
parameters	■ `arg1` — the first array of SearchResults ■ `arg2` — the second array of SearchResults
returns	a one-dimensional array of SearchResults objects
throws	The `or()` method does not throw any exceptions.

rank

format	`SearchResults[] rank(SearchResults arg1[])`
description	Assigns a ranking to each member of the given array of SearchResults objects.
parameters	■ `arg1` — the array of SearchResults objects to be ranked
returns	a one-dimensional array of ranked SearchResults objects
throws	The `rank()` method does not throw any exceptions.

rankSort

format	`SearchResults[] rankSort(SearchResults arg1[])`
description	Sorts the given array of ranked (but unordered) SearchResults objects and populates it with the given SearchResults objects, ordered according to their rankings.
parameters	■ `arg1` — the array of ranked SearchResults objects to be sorted
returns	a sorted one-dimensional array of ranked SearchResults objects
throws	The `rankSort()` method does not throw any exceptions.

setCollectionName

format	`void setCollectionName(String collectionName)`
description	Sets the name of the MediaCollection being searched. This name is necessary to support distributed searches on several MediaCollections, perhaps across several servers.
parameters	■ `collectionName` — the name of the MediaCollection being searched
returns	nothing (void)
throws	The `setCollectionName()` method does not throw any exceptions.

setContext

format	`void setContext(EveContext eveContext)`
description	Sets the context of the search results. If you are using only one database, you likely do not need to use this method. However, if you are using more than one database, you need to use an EveContext object to make sure the system knows where to find your search results.
parameters	■ `eveContext` — the context of the SearchResults
returns	nothing (void)
throws	The `setContext()` method does not throw any exceptions.

setDistance

format	`void setDistance(int indexType, double value)`
description	Sets the current result's *distance* from the target image. This only applies to one index type at a time. For example, you could set the result's texture distance from the target to 0.7, you would call the method like this: `setDistance(Eve.TEXTURE, 0.7)`
parameters	■ `indexType` — the type of index (such as color or region) to which the distance applies ■ `value` — the result's distance from the target image
returns	nothing (void)
throws	The `setDistance()` method does not throw any exceptions.

setKey

format	`vvoid setKey(long key)`
description	Sets the key of the SearchResults object. Because SearchResults objects are serializable, the object's key provides a way to retrieve it later on.
parameters	■ `key` — the object's new key
returns	nothing (void)
throws	The `setKey()` method does not throw any exceptions.

setRank

format	`void setRank(int rank)`
description	Sets the rank of the current result relative to the other results of the search.
parameters	■ `rank` — the item's rank
returns	nothing (void)
throws	The `setRank()` method does not throw any exceptions.

setSimilarity

format	`void setSimilarity(double similarity)`
description	Sets the degree of similarity between the search result and the target image. Similarity is a value between 0 (zero) and 1 representing the target image's combined raw distance scores.
parameters	■ `similarity` — represents the degree of similarity between the search result and the target image
returns	nothing (void)
throws	The `setSimilarity()` method does not throw any exceptions.

similaritySort

format	`SearchResults[] similaritySort(SearchResults arg1[])`
description	Takes the given array of unordered SearchResults objects and sorts it according to the objects' similarity scores.
parameters	■ `similarity` — represents the degree of similarity between the search result and the target image
returns	a one-dimensional array of SearchResults objects sorted in descending order by similarity
throws	The `similaritySort()` method does not throw any exceptions.

Vocabulary

This interface provides access to functionality for creating a Visual Vocabulary. A Visual Vocabulary is a representative list of images that fit a certain set of criteria. See the *Getting Started Guide* for an overview Visual Vocabularies.

create(Mediacollection, long[], SearchParameters, double, int)

format

```
Vector create(MediaCollection mediaCollection,long
      keys[],SearchParameters parameters,double threshold,int
      minimumClusterSize) throws EveException
```

description

Use this method to generate a Visual Vocabulary for a *specific* subset or series of images based on the values defined for the parameters. If you want to generate a visual vocabulary for an entire database of images, use the following method, ***create(Mediacollection, SearchParameters, double, int)***.

parameters

- `mediacollection` — the MediaCollection for which you want to build a Visual Vocabulary
- `keys` — an array of keys of SearchResults objects.
- `parameters` — the `SearchParameters` object containing the arguments for the search
- `threshold` — a value (0-100) that determines how similar an image must be to appear within a group in the vocabulary. A high threshold means images must be very similar. A low threshold relaxes the criteria. For example, if you set this value to 90, then images that are 90% visually similar are organized within the same group.
- `minimumClusterSize` — (0-10) the value that determines the minimum size of the groups in which visually similar images (based on the `threshold` value) are organized. For example, the lower the value, the more groups that are created and the less images that appear within those groups; the higher the value, the less groups that are created and the more images that appear within those groups.

  For large databases of images, we recommend using a higher number for the value of this parameter.

returns

nothing (void)

throws

The `create()` method throws an EveException if the create fails.

create(Mediacollection, SearchParameters, double, int)

format

```
Vector create(MediaCollection mediaCollection, SearchParameters
      parameters,double threshold,int minimumClusterSize)
      throws EveException
```

description Use this method to generate a Visual Vocabulary for a images based on the values defined for the parameters. If you want to generate a visual vocabulary for a specific subset or series of images, use the method, *create(Mediacollection, long[], SearchParameters, double, int)*.

parameters

- `mediacollection` — the MediaCollection for which you want to build a Visual Vocabulary
- `parameters` — the `SearchParameters` object containing the arguments for the search
- `threshold` — a value (0-100) that determines how similar an image must be to appear within a group in the vocabulary. A high threshold means images must be very similar. A low threshold relaxes the criteria. For example, if you set this value to 90, then images that are 90% visually similar are organized within the same group.
- `minimumClusterSize` — (0-10) the value that determines the minimum size of the groups in which visually similar images are organized. For example, the lower the value, the more groups that are created and the less objects that appear within those groups; the higher the value, the less groups that are created and the more objects that appear within those groups.

  For large databases of images, we recommend using a higher number for the value of this parameter.

returns nothing (void)

throws The `create()` method throws an EveException if the create fails.

setContext

format

```
void setContext(EveContext context)
```

description Sets the Vocabulary's context information. For more information on EveContext, see the *EveContext* section.

parameters

- `context` — the context of the object to be analyzed

returns nothing (void)

throws The `setContext()` method does not throw any exceptions.

3

XML Interface to eVe

This chapter describes how to use eVe's XML socket interface.

The eVe XML interface enables you to perform visual search functions by passing XML data from an application to eVe. The following lists:

- the XML commands supported by eVe
- the output produced by those commands in both XML and pipe-delimited strings (which can be easily parsed) format

XML Command / Syntax	XML Output	Pipe-Delimited String Output

■ `addMetadata`
XML input syntax:
`<EveCommand>`
`  <command>addMetadata</command>`
`  <arg1>metadata key</arg1>`
`  <arg2>metadata value</arg2>`
`  <shortResponse>true</shortResponse>`
`</EveCommand>`
	`<EveResponse>`
	`  <errorFlag>false</errorFlag>`
	`  <errorMessage>OK</errorMessage>`
	`</EveResponse>`
		`"ok"`
		or
		`"error|error message"`

■ `analyze`
XML input syntax:
`<EveCommand>`
`  <command>analyze</command>`
`  <shortResponse>true</shortResponse>`
`</EveCommand>`
	`<EveResponse>`
	`  <errorFlag>false</errorFlag>`
	`  <errorMessage>OK</errorMessage>`
	`</EveResponse>`
		`"ok"`
		or
		`"error|error message"`

XML Command / Syntax	XML Output	Pipe-Delimited String Output
binaryImage XML input syntax: <EveCommand> <command>binaryImage</command> <arg1>foreign key value</arg1> <shortResponse>true</shortResponse> </EveCommand> **Note • This command triggers two reads by the server - the first to read the XML command itself, followed by a second binary read of the client to get the image.**	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok"
closeDatabase XML input syntax: <EveCommand> <command>closeDatabase</command> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok" or "error|error message"
createDatabase XML input syntax: <EveCommand> <command>createDatabase</command> <arg1>name of database</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok" or "error|error message"
deleteRecord XML input syntax: <EveCommand> <command>deleteRecord</command> <arg1>foreign key</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok"
foreignKeyRegionSearch XML input syntax: <EveCommand> <command>foreignKeyRegionSearch</ command> <arg1>colorWeight</arg1> <arg2>regionWeight</arg2> <arg3>shapeWeight</arg3> <arg4>textureWeight</arg4> <arg5>foreign key</arg5> <arg6>region1|region2|etc.</arg6> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."

XML Command / Syntax	XML Output	Pipe-Delimited String Output
■ foreignKeySearch XML input syntax: <EveCommand> <command>foreignKeySearch</command> <arg1>colorWeight</arg1> <arg2>regionWeight</arg2> <arg3>shapeWeight</arg3> <arg4>textureWeight</arg4> <arg5>foreign key</arg5> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."
■ getFirstMetadataValue XML input syntax: <EveCommand> <command>getFirstMetadataValue</command> <arg1>metadata key</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|metadata value"
■ getKeys XML input syntax: <EveCommand> <command>getKeys</command> <shortResponse>true<shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."
■ getSegmentationMask XML input syntax: <EveCommand> <command>getSegmentationMask</command> <arg1>foreignKey</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	ok|width|height|bunch of 0's, 1's, 2's, etc"
■ getVocabulary XML Input Syntax: <EveCommand> <command>getVocabulary</command> <arg1>colorWeight|regionWeight|shapeWeight|textureWeight|similarity</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok" or "ok|foreign key 1|foreign key 2|etc."

XML Command / Syntax	XML Output	Pipe-Delimited String Output
loadImageFromURL XML input syntax: <EveCommand> <command>loadImageFromURL</command> <arg1>url string</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok" or "error|error message"
loadImage XML input syntax: <EveCommand> <command>loadImage</command> <arg1>media object key</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok"
metadataFindExact XML input syntax: <EveCommand> <command>metadataFindExact</ command> <arg1>metadata key</arg1> <arg2>metadata value</arg2> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."
metadataFind XML input syntax: <EveCommand> <command>metadataFind</command> <arg1>metadata key</arg1> <arg2>metadata value</arg2> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."
openDatabase XML input syntax: <EveCommand> <command>openDatabase</command> <arg1>name of database</arg1> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok" or "error|error message"
reorganize XML input syntax: <EveCommand> <command>reorganize</command> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok"

XML Command / Syntax	XML Output	Pipe-Delimited String Output
■ saveImage XML input syntax: <EveCommand> <command>saveImage</command> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok"
■ search XML input syntax: <EveCommand> <command>search</command> <arg1>colorWeight</arg1> <arg2>regionWeight</arg2> <arg3>shapeWeight</arg3> <arg4>textureWeight</arg4> <shortResponse>true</shortResponse> </EveCommand>	<EveResponse> <errorFlag>false</errorFlag> <errorMessage>OK</errorMessage> </EveResponse>	"ok|key 1|key 2|etc."

4

Error Handling

This chapter describes the three forms of `EveException`, differentiated by the number and type of their arguments.

Overview

The `com.evisionglobal.eve.kernel.eveException` package defines eVe's own exception class.

When you are writing a method that throws or catches an eveException, keep the following things in mind:

- catch exceptions as early as possible
- EveExceptions log themselves, so long as `logErrors` in the `Eve.properties` file is `TRUE`

eVision recommends that you enclose all method calls within try...catch blocks so that you can trap exceptions that occur and deal with them while the source of the exception is clear.

There are three forms of EveException, differentiated by the number and type of their arguments.

EveException `(String, String, Exception)`

format	`EveException(String fromClass,String fromMethod,Exception e)`
description	This is the simplest form of EveException. This EveException simply wraps a standard Java (or other) exception. It contains the name of the throwing class, the throwing method, and the exception that was thrown. It also contains a stack trace in *`ExceptionName`*`.stackTrace`.
parameters	■ `fromClass` — the class of the method throwing the exception ■ `fromMethod` — the method throwing the exception ■ `e` — the actual exception thrown
returns	nothing
throws	The `EveException()` method does not explicitly throw any exceptions.

EveException (String, String, String)

format	`EveException(String fromClass,String fromMethod,String message)`
description	This type of EveException does not wrap another exception. If you want to throw an EveException of your own, use this EveException.
parameters	■ `fromClass` — the class of the method throwing the exception ■ `fromMethod` — the method throwing the exception ■ `message` — a message explaining the exception
returns	nothing
throws	The `EveException()` method does not explicitly throw any exceptions.

EveException (String, String, String, Exception)

format	`EveException(String fromClass,String fromMethod,String message,Exception e)`
description	This style of exception is the most comprehensive. It allows you to wrap a Java exception within an EveException and add your own message text, such as the calling method. It also contains a stack trace in *`ExceptionName`*`.stackTrace`.
parameters	■ `fromClass` — the class of the method throwing the EveException ■ `fromMethod` — the method throwing the EveException ■ `message` — a message explaining the exception ■ e — the actual Java exception that `fromMethod` caught
returns	nothing
throws	The `EveException()` method does not explicitly throw any exceptions.

EveException *(String, String, String, Exception)*

Index

H

I

L

M

N

O

P

R

S

U

V

X