

KeyHole

a picture library system

Design Document

Group 18

Åke Blomdahl - ake.blomdahl@eldfluga.com

Anders Hård - anders@trotsa.se

Jonas Moberg - md99-jmo@nada.kth.se

Martin Hargel - hargel@kth.se

Rafael Osorio - osorio@kth.se

Innehåll

1. INTRODUCTION	7
PURPOSE	7
SCOPE	7
THE INTENDED AUDIENCE	7
VERSION HISTORY	7
RELATED DOCUMENTS	7
GLOSSARY	8
ABSTRACT	9
NAMING AND CODING CONVENTIONS	9
<i>Naming</i>	9
<i>Commenting Conventions</i>	10
2. SYSTEM OVERVIEW	11
2.1 GENERAL DESCRIPTION	11
<i>Tags</i>	11
<i>Picture piles</i>	11
<i>Albums</i>	11
<i>Picture manipulation</i>	11
<i>Web export</i>	11
<i>System design</i>	11
Client-server advantages	11
Client-server disadvantages	11
2.2 OVERALL ARCHITECTURE DESCRIPTION	11
2.3 DETAILED ARCHITECTURE	13
<i>The Database Model</i>	13
<i>Making a selection</i>	15
<i>Picture manipulation</i>	15
<i>Web export</i>	17
3. DESIGN CONSIDERATIONS	18
3.1 ASSUMPTIONS AND DEPENDENCIES	18
<i>Metadata in JPEG picture</i>	18
<i>Database engine</i>	18
3.2 GENERAL CONSTRAINTS	18
<i>Metadata in JPEG picture</i>	18
<i>Network Communication</i>	18
<i>Verification and validation requirements</i>	18
4. GRAPHICAL USER INTERFACE	19
4.1 OVERVIEW	19
FRAMES AND FORMS	20
<i>Frame: frmTreeFiles (Filträd)</i>	20
<i>Frame: frmTreeTags (Taggträd)</i>	21
<i>Frame: frmTreeAlbum (Albumträd)</i>	22
<i>Frame: frmMenuBar (Menu)</i>	23
<i>Frame: frmToolBar</i>	24
<i>Frame: frmPicturesMin (Miniatyrvisning)</i>	24
<i>Frame: frmPicturesDet (Detaljvisning)</i>	25
<i>Frame: frmPicturesFull (Hel bildvisning)</i>	26
<i>Form: frmPictureInfo (Visa bildinformation)</i>	27
<i>Form: frmSearch (sökning)</i>	29
<i>Form: frmCreateWebpage (Skapa Webb sida)</i>	30
<i>Dialogue: frmNewUser (Ny användare)</i>	31
<i>Dialogue: frmAddTag</i>	32
5 DESIGN DETAILS	33

5.1 CLASS RESPONSIBILITY COLLABORATOR (CRC) CARDS	33
<i>Client application classes</i>	33
Class ClientApp	33
Class ClientSettings	33
Class ClientsideCom	33
Class FileManager	34
Class WebPage	34
GUI classes	35
Class GUIFilesystemTree	35
Class GUILogin	35
Class GUIUserInfoForm	35
Class GUIPictureView	35
Class GUIPicInfo	36
Class GUISearchForm	36
Class GUITree	36
<i>Server application</i>	37
Class ServerApp	37
Class ServersideCom	37
Class DBConnection	37
Class ServerSettings	37
Class ServerLogfile	38
<i>Common classes</i>	38
Class WebTemplate	38
Class Picture	38
Class PictureSet	38
Class PicturePile	39
Class User extends Photographer	39
Class Photographer	39
Class TagNode	40
Class TagTree	40
Class Album	40
Class AlbumPicture	40
Class AlbumTree	41
Class Manipulator	41
Class NetworkObject	41
5.2 CLASS DIAGRAM	42
5.3 STATE CHARTS	43
<i>The Server application starts up</i>	43
<i>Client startup</i>	44
5.4 INTERACTION DIAGRAMS	45
<i>Change view</i>	45
<i>Import of pictures to KeyHole</i>	46
<i>Mark a picture with a tag</i>	47
<i>Add a new tag to an existing tag-tree</i>	48
<i>Search by means of tags</i>	49
<i>Creating a webpage</i>	51
<i>Starting KeyHole</i>	52
5.5 DETAILED DESIGN	53
SERVER CLASSES	53
Class ServerApp	53
Fields	53
Methods	53
Class ServerLogFile	53
Fields	53
Methods	53
Class ServerSettings	53
Fields	54
Methods	54
Class DBConnection	54
Fields	54

Methods	54
Class <i>ServersideCom</i>	57
Fields	57
Constructor	58
Methods	58
CLIENT CLASSES	59
Class <i>FileManager</i>	59
Fields	59
Methods	59
Class <i>ClientsideCom</i>	60
Fields	60
Constructor	60
Methods	61
Class <i>ClientApp</i>	61
Fields	61
Methods	61
Class <i>WebPage</i>	63
Fields	63
Constructor	63
Methods	63
Class <i>ClientSettings</i>	64
Fields	64
Methods	64
COMMON CLASSES	65
Class <i>SearchForm</i>	65
Fields	65
methods	65
Class <i>Picture</i>	65
Fields	65
Methods	66
Class <i>PicturePile</i>	67
Fields	67
Methods	67
Class <i>PictureSet</i>	67
Fields	67
Methods	67
Class <i>TagNode</i>	68
Fields	68
Constructor	68
Methods	68
Class <i>TagTree</i>	69
Fields	69
Methods	69
Class <i>Album</i>	70
Fields:	70
Constructor	70
Methods	70
Class <i>AlbumPicture</i>	71
Fields	71
Methods	71
Class <i>AlbumTree</i>	72
Fields	72
Methods	72
Class <i>Manipulator</i>	72
Methods	72
Class <i>Photographer</i>	74
Fields	75
Constructor	75
Methods	75
Class <i>User</i>	76

Fields	76
Constructor	76
Methods	76
Class <i>NetworkObject</i>	78
Fields	78
Constructor	78
Methods	79
Class <i>WebTemplate</i>	79
Fields	79
Constructor	79
Methods	79
Database model	80
T-matrix:	81
Logical database model:	81
Requirements document cross references	82
5.6 PACKAGE DIAGRAM	88
Package <i>khServerApplication</i>	88
Package <i>khClientApplication</i>	88
Package <i>khCommon</i>	88
6. FUNCTIONAL TEST CASES	89
Import	89
Delete a picture	89
Delete a picture – insufficient rights	90
Delete a picture – locked by another user	90
Create an album	91
Create an album on a search result set	91
Change album properties	91
Change album name	92
Changing album name to invalid name	92
Changing sort order for an album	93
Removing pictures from an album	93
Deleting an album	94
Assigning a tag to a picture	94
Adding a new tag to the tag tree	95
Adding a new tag to the tag tree – expected failure	95
Removing a tag from the tag tree	96
Searching for pictures using the tag tree	96
Searching for pictures using the search form	97
Big result set	98
Creating a new user	98
Creating a new user – expected failure	99
Create a picture pile	99
Add a picture to a picture pile	100
Assigning a new main picture of a picture pile	100
Inherit properties from the main picture of a picture pile	101
Remove a picture from a picture pile	101
Dissolve a picture pile	101
Scaling a picture - percent	102
Scaling a picture - pixel	102
Automatic scaling	103
Crop a picture	104
Rotate a picture	104
Change picture information	105
Change picture information - lock failure	105
Change copyright agreement – user rights failure	105
Create a webpage	106
Logging in	107
Incorrect username or password	107
First start up of a client	107

<i>Moving pictures</i>	108
<i>Lost communication</i>	108
<i>minimal required hardware</i>	109
REFERENCES	110
DIAGRAMMING TECHNIQUE REFERENCES	110
OTHER REFERENCES	110
APPENDIX I	111
THE BITMAP .BMP FILE FORMAT	111
<i>Introduction:</i>	111
<i>Basic structure:</i>	111
<i>Exact structure:</i>	111
The BITMAPFILEHEADER:	111
The BITMAPINFOHEADER:	112
The RGBQUAD array:	112
The pixel data:	113
INDEX	114

1. Introduction

Purpose

The purpose of this document is to clearly define what the programmers have to do. The task is to create a working product that meets the expectations defined in the prior “Requirements Document” (version 2) for the “KeyHole” project.

Scope

KeyHole is a computer program system that basically consists of a server program (KH-server) a database and a client program that may be running in several instances. The database architecture is defined in this document, including queries and possibly stored procedures. The database is accessible through a database engine (DBMS, a Database Management System). The DBMS is expected to be installed and up and running as a system requirement. It’s working and construction is out of the scope of this project. There must exist various operating system API:s that the system could use. Copying files and managing disks is not addressed by KeyHole.

The intended audience

The first audience is the people who are supposed to implement this described system, e.g. the programmers.

But there is also other persons that have interest in it. Other fellow designers who are expected to read and comment. If the document is clear and understandable it is possible for them to find bugs, mistakes and weak spots. Early feedback from experienced people could save a lot of time and money in the long run.

If the system will be further developed it’s very important to have insight in the intent and assumptions of the developers of the first version.

If this was a commercial product there would be managers that want to see results. They will need time estimates and a way to evaluate the progress.

Version history

This is the first version.

Related documents

There is in this text extensive reference to the “Requirements Document” (version 2). The reader of this document is expected to be familiar with the “Requirements Document”.

Glossary

As we wrote the prior document in Swedish we have a need to clarify some of the terms as they appear in English.

album: An ordered set of links to pictures stored in the system. One particular picture may very well occur in several albums.

bitmap picture: a picture built on a right angled raster of color dots (pixel).

bmp: Windows bitmaps format. An old strait forward uncompressed pixel raster description file format.

client: Here we normally refer to the client, or KH-client, as the application that is running on a local workstation. The client shows an interface (GUI) with pictures, buttons, menus and text fields. The client sends requests and tasks to the *server* program.

DBMS, a Database Management System

dialogue box: a small window displayed on top of the main window. It have one or a few editable fields. It have buttons like “Yes” and “no”, “OK” and “Cancel”, “Change” and “Abort”.

event: When a user interact with the program, for instance clicks a button, an “event” is triggered causing an event message in the operating system, finally reporting back to some defined method.

file system: the hierarchical file system with folders and files as it is given by the operating system

form: a form is a panel or frame with textfields, comboboxes and checkboxes and other user intractable widgets. It is the standard way for the users to enter and edit data into the system.

frame: a part of the client *window*, almost equivalent to a *panel*. Action in one frame, for instance the menu bar, may result in changed form or content in another frame.

jpg/jpeg: A bitmap file format that allows very high compression. Some information will be lost in the process.

KH: KeyHole, e.g. the system under development.

KH-window: The complete standard interface of the *client program*, including all *frames*.

main picture: (Sw: "huvudbild") A *picture pile* has at least one file. The first one, or most representative one, is the “main picture”. The main picture will represent the whole pile as long as it is handled as a package.

message box: A small window used to display information, varnings and error messages. It has only a close button.

metadata: A jpg file starts with a header where various text and number data could optionally be stored.

miniature: a small picture that fits in a standard size square (96x96 or 128x128 pixel)

panel: a rectangular area of graphics, a container of other user interaction widgets (buttons, textfields ...).

picture pile: (Sw: "fotohög") A set of picture files, each of them has been created from the same original, they are to be considered as versions of the same picture. Being precise, all pictures in KeyHole are stored as “picture piles”, but most of them consist of only one (main) picture. In most cases in the following text, when we talk about a picture pile, we understand a pile consisting of more than one picture.

picture tree: A picture tree is the common name for any treelike hierarchical structure of pictures. It may show albums, tags or files.

pixel: the atom of a bitmap picture.

tag: a word used as index entry associated with a picture.

tag tree: KH help the user to organize the tags in an hierarchical way. The tree has a root and each node have an arbitrary number of branches.

thread: Simple computer programs execute the program statements sequentially, with jumps and loop but in an absolutely defined order. They can only do one task at the time. A threaded program can (simulate) parallel processing of two or several tasks.

wild card: in a search query you may sometimes write question mark (?) to say that any letter will do in this position. If instead you write a star (*) there could be none or several letters in this place. These marks are called ‘wild cards’.

Abstract

The KeyHole is a picture-organizing tool. It is a multi-user system for organizing large quantities of pictures. It offers a way to classify pictures, making it possible to find them later on.

Chapter 2 of this document gives a general description of this system. The client-server architecture in general and in detail is explored.

Chapter 3 discusses design considerations we have had to make. It especially addresses complication in the use of the jpg file format. It also explores dependencies of other software, such as a database engine.

Chapter 4 deals with the user interface, the forms, fields and buttons that are available to the end user.

Chapter 5 describes in detail the classes, attributes and methods that must work together in order to make it a functional program.

Chapter 6 gives detailed instructions on how a human tester could test the system.

Naming and Coding Conventions

Use the coding conventions described at

“Code Conventions for the Java™ Programming Language”,

<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

Naming

A good naming convention:

Will cause logical variable grouping in the lists

Will allow the each of the variable types to use the same basic name (e.g. ClientAddress-t, a text variable, and ClientAddress-c, a computation variable)

Will be descriptive enough to tell you exactly what the variable is

Will be predicable enough to allow you to find variables easily, and

Will be consistent enough to allow you to share answer files across templates

(“The HotDoc Computation Archives”, <http://www.legalcs.com/hotdocs/0134.htm>)

All variables and classes should be in English..

Classes

Class names should be nouns. First character should be upper case.

```
class Raster;  
class ImageSprite;
```

Methods

A name of a method should be a verb, using both upper and lower case letters. The first letter should be lower case. Use getXxxx when a property are returned and setZzz when a property is set.

```
run();  
runFast();  
getBackground();
```

Variables

All variables and instances of objects both upper and lower case letters. The first letter should be lower case.

```
int i;  
char cp;  
float myWidth;
```

Second, a variable name should reflect something about the properties of the variable, such as its data type. Many programmers use a convention in which the first few characters of a variable's name indicate the data type of the variable. This is sometimes referred to as a *Hungarian naming convention*, after the Hungarian programmer Charles Simonyi, who is credited with its invention.

(Steven Roman, http://www.romanpress.com/Articles/Naming_R/Naming.htm)

Constants

Constants should consist of only upper case letters, digit characters and _.

```
static final int MIN_WIDTH = 4;
```

```
static final int MAX_WIDTH = 999;
```

```
static final int GET_THE_CPU = 1
```

Commenting Conventions

Use Java doc comments:

“How to Write Doc Comments for the Javadoc Tool”,

<http://java.sun.com/j2se/javadoc/writingdoccomments/index.html>

2. System Overview

2.1 General Description

The KeyHole is a picture-organizing tool. It is a multi-user system for organizing large quantities of pictures. It offers a way to classify pictures, making it possible to find them later on.

Tags

As the user adds pictures to the system he classifies the pictures using tags. The tags are structured into hierarchical trees defined by the user. The user can add a new tag to the tree at any time, the tag trees can consequently be extended with new branches as the users add more pictures.

Picture piles

A picture pile is a set of pictures depicting the same scene with some alteration of for example color, contrast or photo angle. Pictures in a picture pile share the same meta-information and just the thumbnail of a picture pile's main picture is shown when the user browses through the photos.

Albums

Another way of organizing pictures is using albums. Albums are a way to group pictures that belong together. A picture can be a member of many albums and organizing pictures in albums doesn't alter the file system. An album can be either personal or shared between all users.

Picture manipulation

Scaling and rotation of pictures will be supported. It will as far as possible support lossless manipulation of pictures.

Web export

To facilitate easier distribution of pictures the KeyHole it is possible to export a set of pictures to a web page. After choosing a set of pictures and making the preferred settings for the web page the system will create and display the result.

System design

The system uses a client-server architecture. The pictures are stored in a file system preferably on a file server. The advantage of having the pictures on a file system instead of directly stored in a database is that there will be no data loss if the system fails. Furthermore it is easier to change systems not having to export the pictures from the database. All the information, for example file location and tag tree hierarchy is stored in a database. The database and file system can only be accessed from the server part of the system, making it possible to manage user permissions. The user authentication is done when a client tries to establish a connection with the server.

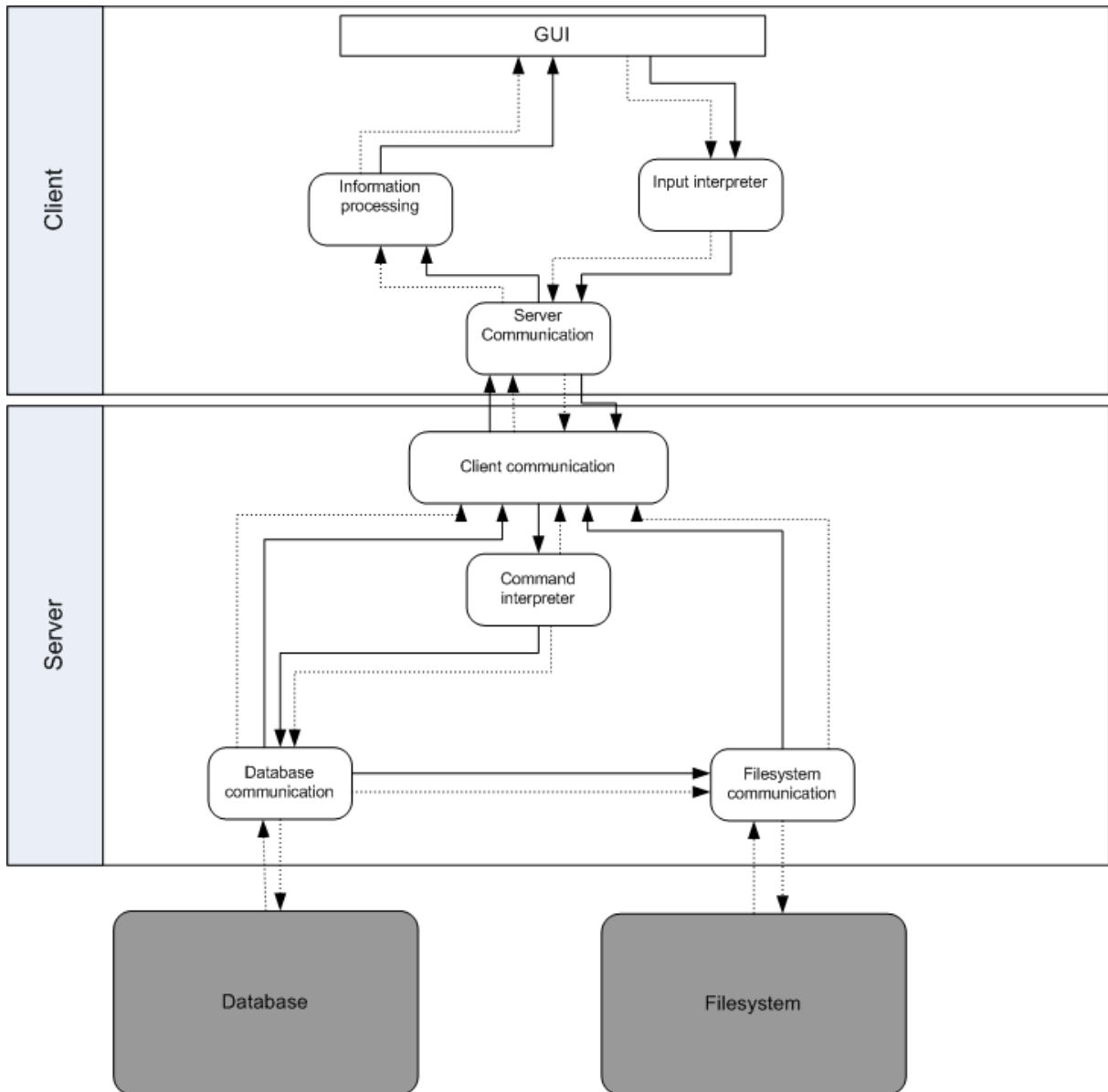
CLIENT-SERVER ADVANTAGES

- Security. It's possible to control user permissions without having to change anything in the database. Direct user access to the file system can be prevented.
- Upgrading. It's possible to change Database Management System without having to upgrade the clients.

CLIENT-SERVER DISADVANTAGES

- Reliability. If the server part of the system goes down the whole system goes down.
- When many clients connect at the same time the server will have a heavy work load.

2.2 Overall Architecture Description

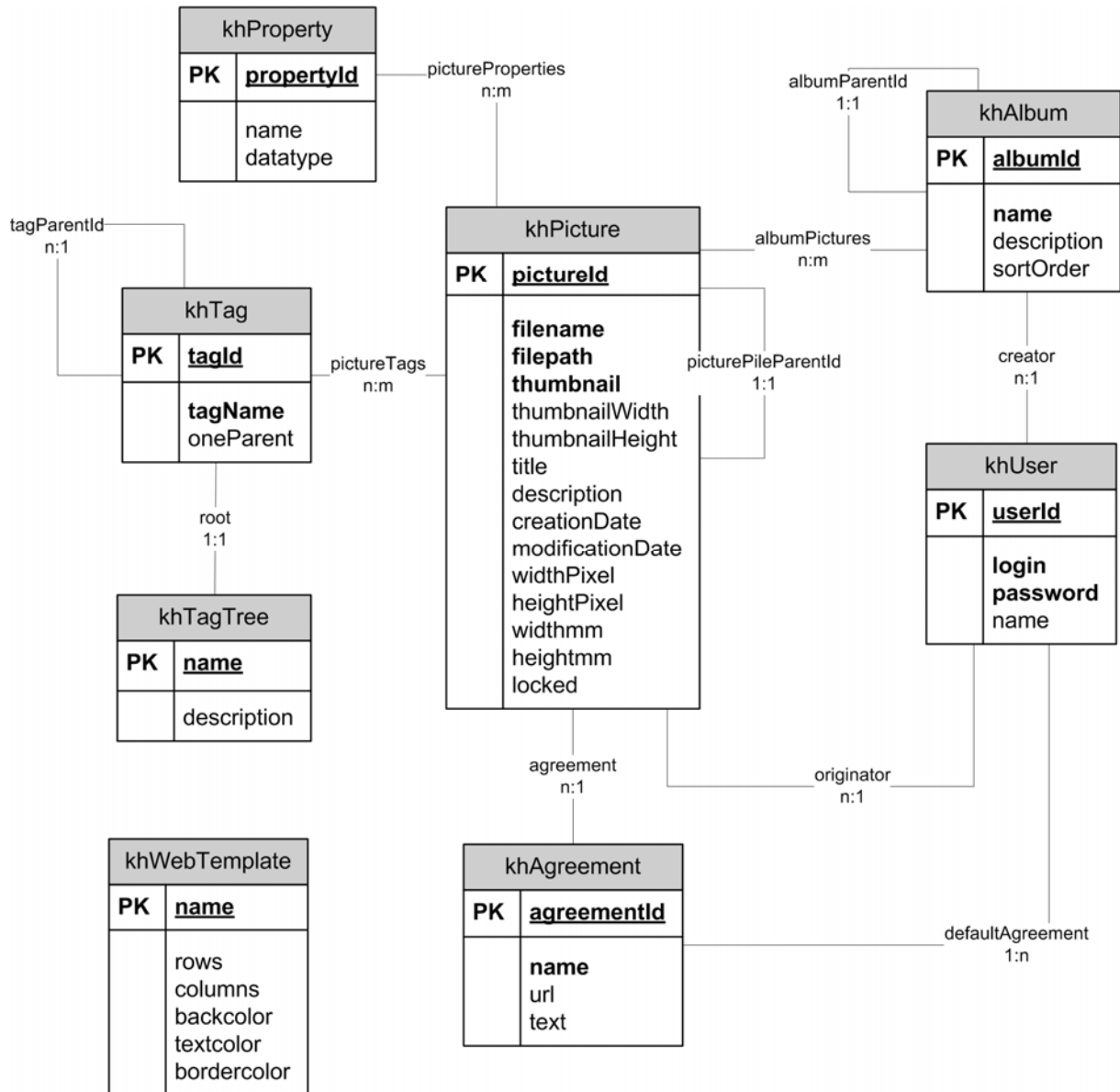


The client interacts with the user via a graphical user interface. An input from the user is interpreted by the input interpreter. The interpreter catches the inputs in the form of a request for, or manipulation of information and sends it to the server communicator. The server communicator sends the command to the server.

The client communicator receives commands from the client and sends them to the command interpreter. The input from the communicator is interpreted and transformed into a command, either to the database or the file system. The file system communicator requests the database communicator for the location of the picture and either manipulates the picture or sends the picture back to the client communicator. The database communicator requests the database for the information and sends it back to the client communicator.

2.3 Detailed Architecture

The Database Model



The entity-relationship model describes the proposed database schema. In the model entities are represented with boxes containing the name of the entity followed by the list of attributes. The relationships between the entities are represented as lines connecting two or more boxes. Relationships have a cardinality associated with them that can be "1 to 1" (1:1), "1 to many" (1:n) or "many to many" (n:m).

The khPicture entity contains information on all pictures in the system. For each picture in the system there is a unique id and other information, such as location of the file on the file system. Additional optional information about the pictures, such as metadata, will be stored in a separate entity called khProperty.

Picture piles are represented in the model with the khPictureSet entity. Since a picture can only be part of one picture pile but each picture pile can contain many pictures the relationship Is_member_of between hkPictureSet and khPicture is 1:n. And as each picture pile has exactly one main picture there is also a 1:1 relationship between the same two entities.

The tag trees are represented by the hkTagTree entity. The tags are identified with a unique id in the hkTag entity. There is a recursive relationship, Is_parent, between tags since each tag has (at least) one parent. Since a tag-word may occur in different trees, or even repeated in the same tree, we need a way to distinguish between them. That is why we have the attribute "one_parent", which is the text from some ancestor tag on that branch.

There are three relationships (has_tag_set, has_tag_pic and has_tag_album) for representing what has been tagged. This is because you can tag not only pictures but also picture piles and albums.

The album entity contains the names and descriptions of the albums in the system. The relationship has_pictures between hkPicture and hkAlbum tells us which pictures are in an album. Since each picture could be in several albums and an album probably contains more than one picture the relationship is n:m.

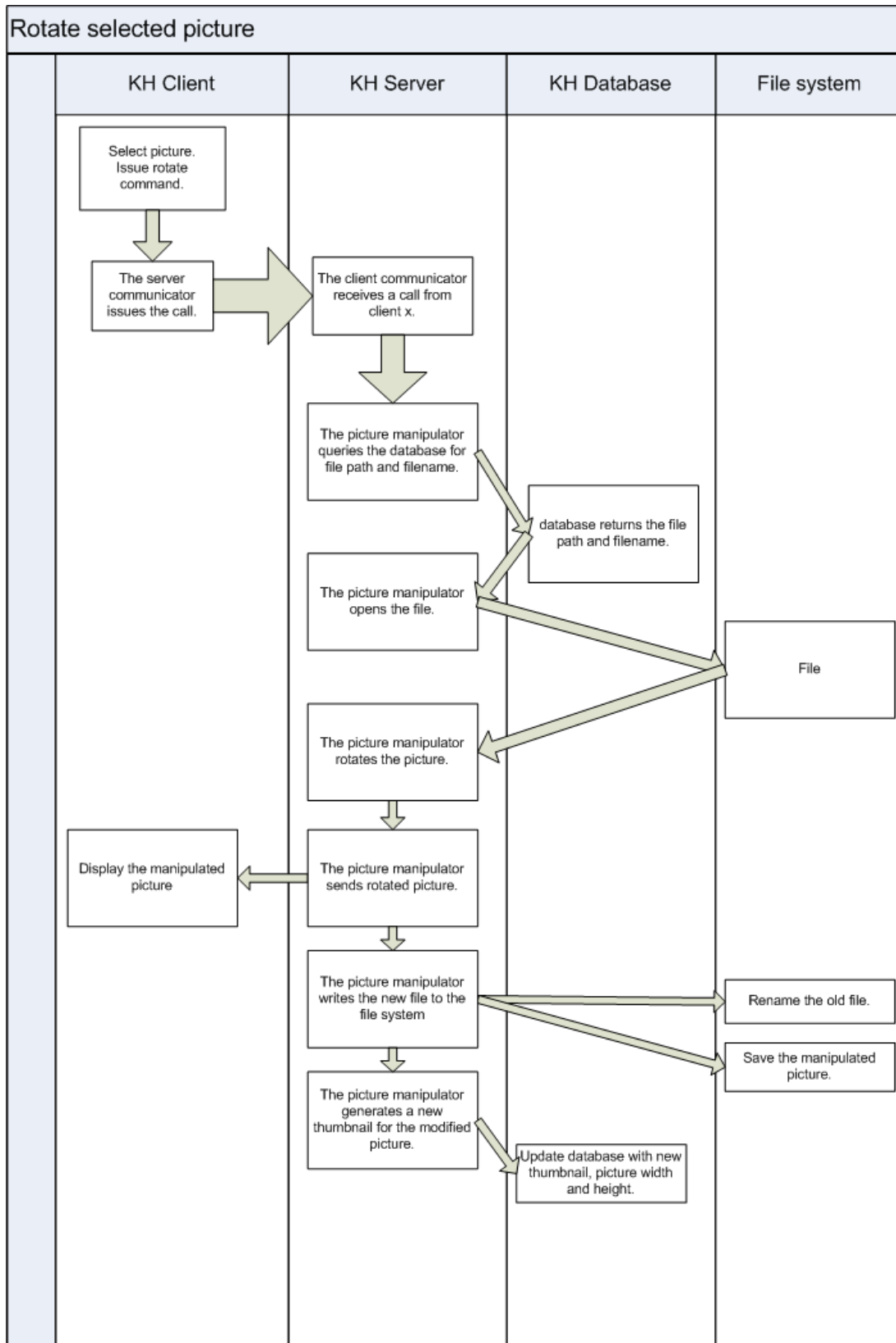
Making a selection

The following diagram shows the control and data flow for making a big selection to show in the client GUI. This is basically a network critical operation. The speed is limited by network capacity. Therefore the uploading process has to be divided into manageable steps.



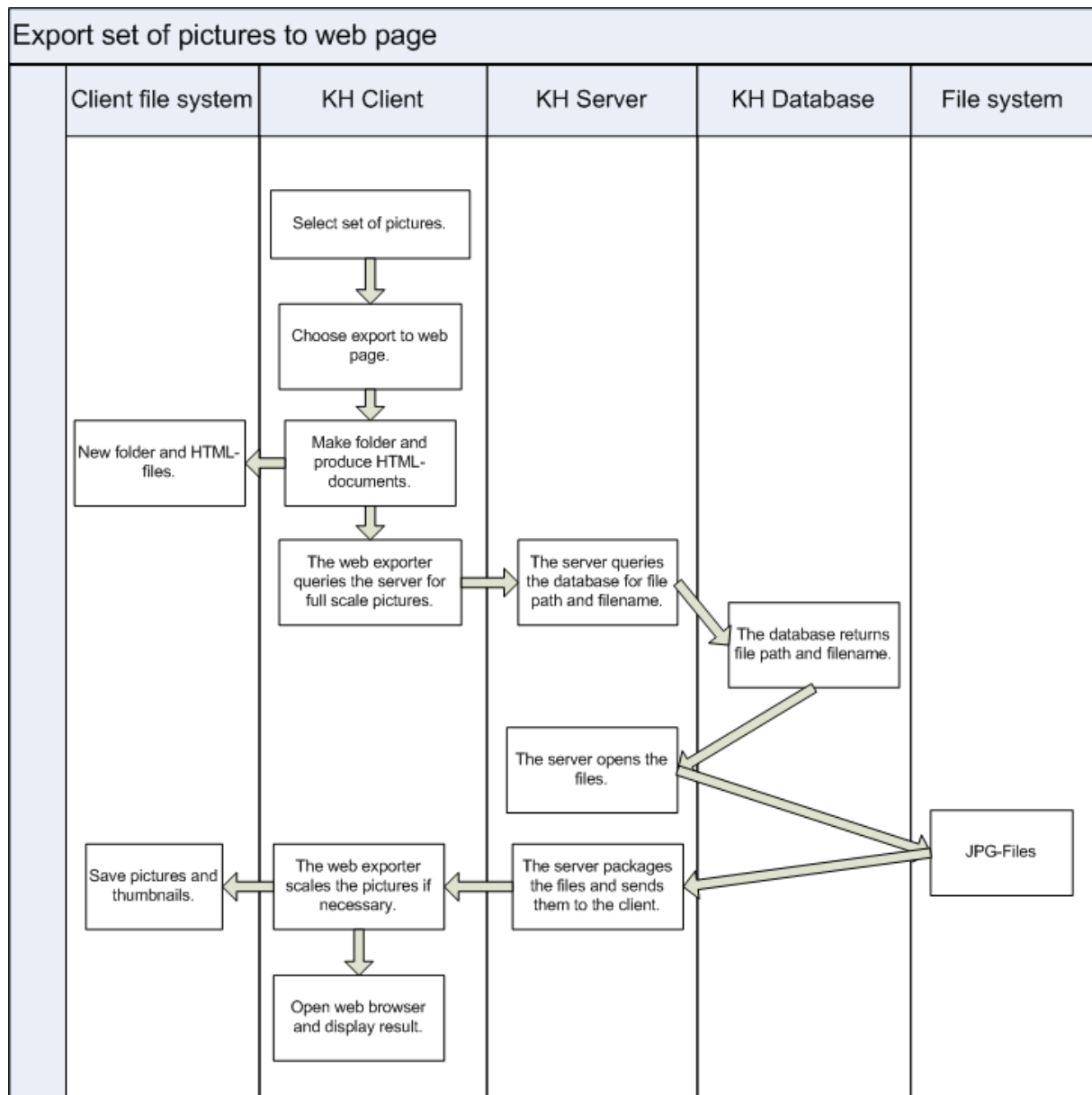
Data and control flow model (UML) for making a big (>25 rows) selection of pictures to display.

Picture manipulation



Data flow model of a lossless jpg-picture manipulating done by the server module. In this case the server program does the basic operations.

Web export



Data flow model of exporting a selection as a set of webpages. The client module does the basic processing and HTML-files are created in a user defined folder in the local client filesystem.

3. Design Considerations

3.1 Assumptions and Dependencies

Metadata in JPEG picture

Reading and writing metadata to a jpeg picture is not straightforward using only the java API. We will therefore use an open source library called Metadata Extractor (<http://www.drewnoakes.com/code/exif/>) to help us with this.

Database engine

We intend to use Microsoft SQL-Server. We assume this server is already installed and up and running. We also assume we have necessary software to connect to MS SQL-server. This is a requirement and not a part of the project.

3.2 General Constraints

Metadata in JPEG picture

The library Metadata Extractor has the constraint that it only reads metadata from the picture. There is no way to change information and write it back to the picture. We will therefore have to store changes to this information in the database only. This will lead to that the information in the database and the information in the picture will be inconsistent.

Network Communication

Verification and validation requirements

Classes that can be taken out of their context with reasonable effort should be tested with an automated test set (using JUnit).

4. Graphical User Interface

4.1 Overview

The graphical user interface is based on a framed window like the one bellow.

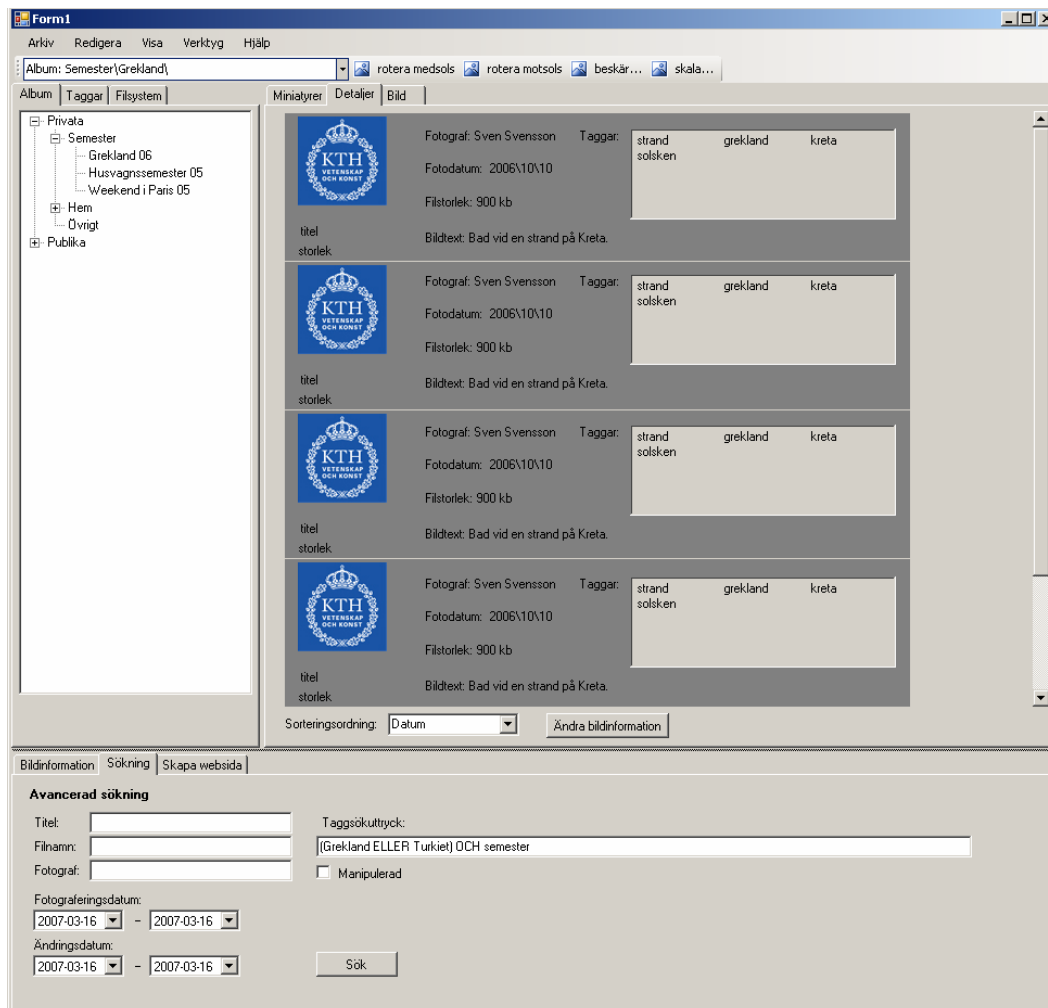
To the right is the big list frame. Here a set of pictures are represented by miniatures or scaled pictures one by one. There are four possible states of this frame: miniatures with minimal information, miniatures listed in one column with big information (as bellow), one picture at a time scaled to fit this frame, and this frame could also be expanded to cover a full screen.

To the left is a picture tree. It could be shown three different views: Albums (as bellow), tags or the file system with beginning in the KeyHole root-folder on the file server.

In the bottom of the form there is a frame that could display four functionally different forms. First it can show (1) detailed information about one or several selected pictures. It could also show the information in an editable form (2). This frame is also used to specify search criteria (3. as in the screenshot below). Finally this frame shows a form for the user who exports a set of pictures to a web page (4).

In top of the page there is a menu bar where all functions and views are immediately available.

Bellow the menu there is a tools bar. It has a address field with history in a drop down combo. When one (or more) pictures are selected rotation or scaling could be applied.



The basic user interface of KeyHole.

Class GUI extends JFrame { ... }

This class collects all user interface elements. It consists of the following frames:

frm

GUI winClientGUI = new GUI();

Frames and forms

Frame: frmTreeFiles (Filträd)



Displayed when the tab "Filsystem" is pressed which trigger a call to showFileTree() in GUI.

Fields:

JTree treeFiles

Controls:

onDoubleClick() calls searchDB(Searchform searchform) in ClientApp.

Frame: frmTreeTags (Taggträd)



Displayed when the tab "Taggar" is pressed.

Event: showTagTree() in GUI.

Fields:

JTree treeTags

Controls:

onDoubleClick() calls searchDB(Searchform searchform) in ClientApp.

Button butChoose – labeled "välj" calls chooseTag(Tag tag) in ClientApp.

ContextMenuItem addTag - "Lägg till tagg..." - Displays the form dialogAddTag.

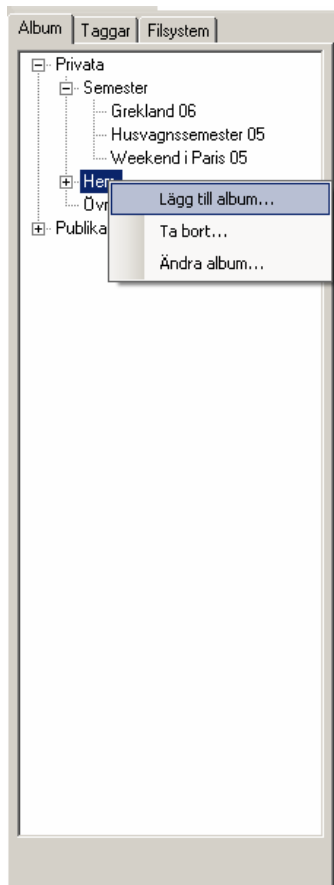
ContextMenuItem removeTag - "Ta bort tagg..." - Displays a dialog asking the user to confirm removing this tag.

References to Requirements Document:

USE CASE: "LÄGGA TILL EN NY TAGG" p.27

USE CASE: "TA BORT EN TAGG UR TAGGTRÄDET" p.28

Frame: frmTreeAlbum (Albumträd)



Displayed when the tab "Album" is pressed.

Event: showAlbumTree() in GUI.

Fields:

JTree treeAlbum – shows both the user's personal albums and the public ones in a tree.

Controls:

onDoubleClick() calls searchDB(Searchform searchform) in ClientApp.

ContextMenuItem addTag - "Lägg till album..." - Displays the form dialogAdd Album.

ContextMenuItem removeTag - "Ta bort..." - Displays a dialog asking the user to confirm removing this album.

ContextMenuItem addTag - "Ändra album..." - Displays the form dialogChangeAlbum.

Frame: frmMenuBar (Menu)



The menu is always displayed. Submenus are opened as drop down menus.

Event: GUI() in GUI.

Fields:

Controls:

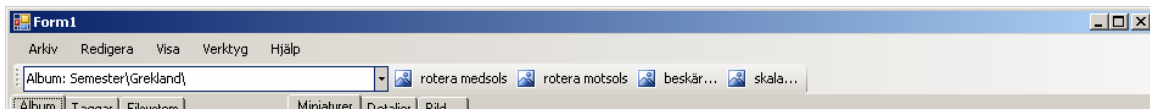
MenuItem import – “importera” calls importPictures() in ClientApp.

MenuItem savePicture – “spara bild som” calls savePictureAs() in ClientApp

MenuItem settings – “inställningar” calls settings() in ClientApp

MenuItem exit – “avsluta” calls exit() in ClientApp.

Frame: frmToolbar



Is called by GUI() in GUI.

Fields:

Controls:

Button crop calls crop() in ClientApp.

Button scale calls scale() in ClientApp.

Button rotateClockwise calls rotate(90) in ClientApp.

Button rotateAntiClockwise calls rotate(270) in ClientApp.

ComboBox recent calls searchDB(SearchForm searchForm) in ClientApp

Frame: frmPicturesMin (Miniatyrvisning)

Displayed when the tab “Miniatyrer” is pressed.

Event: showThumbnailView() in GUI.

Fields:

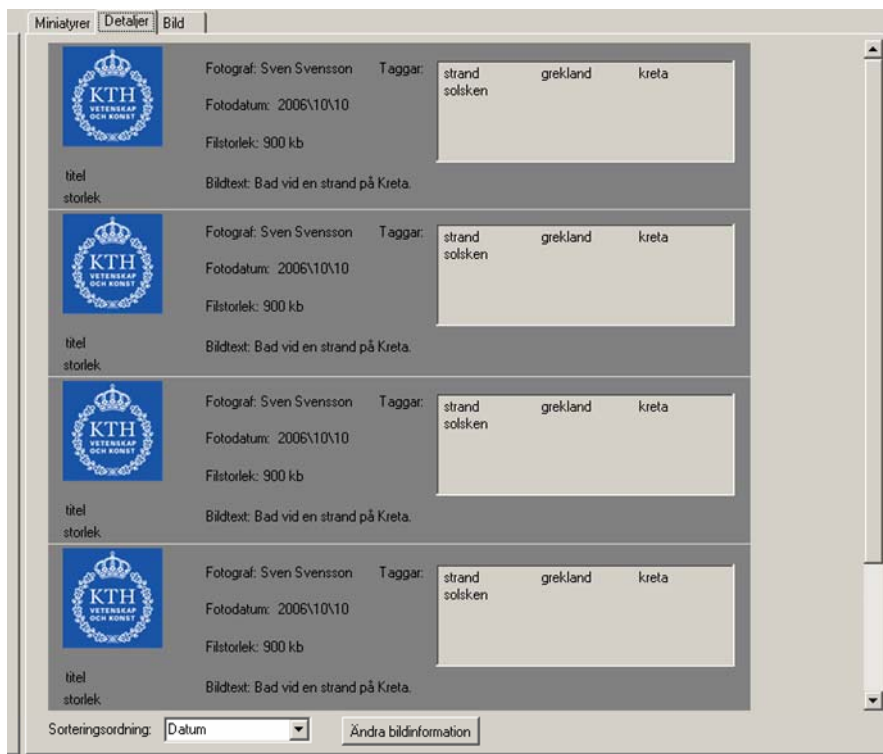
pictureField – displays pictures as thumbnails.

Controls:

ComboBox sortOrder – “sorteringsordning” calls sortOrder(ComboBoxItem sort) in ClientApp.

Button change – “ändra bildInformation” calls changePicInfo() in ClientApp.

Frame: frmPicturesDet (Detaljvisning)



Displayed when the tab “Detaljer” is pressed.

Event: showDetailsView() in GUI.

Fields:

pictureField – displays pictures as thumbnails with details.

Controls:

ComboBox sortOrder – “sorteringsordning” calls sortOrder(ComboBoxItem sort) in ClientApp.

Button change – “ändra bildinformation” calls changePicInfo() in ClientApp.

Frame: frmPicturesFull (Hel bildvisning)



Displayed when the tab "Bild" is pressed.

Event: picView() in GUI.

Fields:

pictureField – shows pictures in big format, one at a time.

Controls:

Button change – "ändra bildInformation" calls changePicInfo() in ClientApp.

Button next – ">" calls nextPic() in ClientApp.

Button last – "<" calls lastPic() in ClientApp.

Form: frmPictureInfo (Visa bildinformation)

The screenshot shows a Windows-style application window titled "Bildinformation" with sub-tabs "Sökning" and "Skapa websida". The main area is titled "Ändra bildinformation". It contains several input fields: "Titel" (Sol och bad i Grekland), "Fotograf" (Sven Svensson), "Bildtext" (Bad vid en strand på Kreta), and "Beskrivning" (Ett bad en het sommar dag på norra Kreta. Mitt på dagen 16/7 2006. På bilden syns Sven). There are also labels for "Sökväg", "Senaste ändring", and "Taggar" (strand, grekland, kreta, solsken). Technical details include "Storlek" (600 x 600 px), "Storlek vid utskrift" (8 x 8 cm), and "Filstorlek" (900 kb). Camera settings like "bländare", "exponeringstid", "fokallängd", and "blixt" are also present. A "Uppdatera information" button is at the bottom right.

Figure 1 The form displays picture information.

This screenshot is identical to Figure 1, showing the same form in view mode. The fields are populated with the same data as in Figure 1.

Figure 2 The same form as above but in editing mode.

Displayed when the tab "Bildinformation" is pressed or when menu item "Bildinformation" is chosen or when a picture is selected. The fields will be opened for change when button "Ändra bildinformation" is pressed.

Event: picInfo() in GUI.

Fields:

TextBox Title - "Titel"

TextBox Photographer - "Fotograf" ' "

TextBox Caption - "Bildtext"

TextBox Description - "Beskrivning"

Label SearchPath - "Sökväg"

Label LastChange - "Senaste ändring"

ListBox Tags- "Taggar"

Label SizePixels - "Storlek"

Label SizeCm - "Storlek vid utskrift"

Label FileSize - "Filstorlek"

ComboBox CopywriteAgreement - "Upphovsrätt"

CheckBox Manipulated – “Manipulerad”
TextBox PhotoDate – “Fotograferingsdatum”
TextBox Diaphragm - “Bländare”
TextBox ExposureTime - “Exponeringstid”
TextBox FocalLength - “Fokallängd”
TextBox Flash - “Blixt”

Controls:

Button updateInfo - “Uppdatera Bildinformation” – A button that, when pressed, stores the changed information by calling ClientApp.updateInfo(pictureSet) where pictureSet contains the selected pictures.

References to Requirements Document

USE CASE: "ÄNDRA BILDINFORMATION" p.36

USE CASE: "TILLDELNING AV TAGG FRÅN PRIMÄRA TAGGTRÄDET" p.25

USE CASE: "VÄLJ TAGGAR FRÅN ANDRA TAGGTRÄD" P.26

Form: frmSearch (sökning)

Bildinformation Sökning Skapa websida

Avancerad sökning

Titel:

Filnamn:

Fotograf:

Fotograferingsdatum: 2007-03-16 - 2007-03-16

Ändringsdatum: 2007-03-16 - 2007-03-16

Taggsökuttryck:

Manipulerad

Sök

Figure 3 The form lets the user search for pictures.

Displayed when the tab "Sökning" is pressed or when menu item "Sökning" is chosen.

Event: searchForm() in GUI

Fields:

TextBox "titel" - title

TextBox "filnamn" – filePath

TextBox "fotograf" - photographer

ComboBox photoDateStart

ComboBox photoDateEnd

ComboBox changeDateStart

ComboBox changeDateEnd

TextBox "Taggsökuttryck" – tagSearchExpression

Checkbox "Manipulerad" - manipulated

Controls:

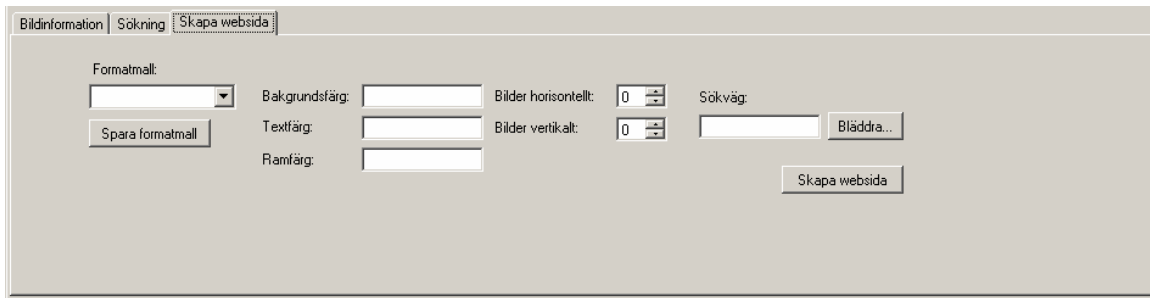
Button "Sök" - search calls searchDB(SearchForm searchform) in ClientApp.

References to Requirements Document

USE CASE: "AVANCERAD SÖKNING" p.29

USE CASE: "SÖKNING MED HJÄLP AV TAGGAR" p.29

Form: frmCreateWebpage (Skapa Webbsida)



The screenshot shows a Windows-style dialog box titled "Skapa websida" with three tabs: "Bildinformation", "Sökning", and "Skapa websida". The "Skapa websida" tab is active. It contains the following controls:

- Formatmall:** A dropdown menu.
- Bakgrundsfärg:** A text input field.
- Textfärg:** A text input field.
- Ramfärg:** A text input field.
- Bilder horisontellt:** A numerical up/down spinner set to 0.
- Bilder vertikalt:** A numerical up/down spinner set to 0.
- Sökväg:** A text input field.
- Buttons:** "Spara formatmall", "Bläddra...", and "Skapa websida".

Figure 4: The form let's the user export pictures to a webpage.

Displayed when the tab "Skapa websida" is pressed or when menu item "Skapa websida" is chosen.

Event: searchForm() in GUI.

Fields:

ComboBox "Formatmall" – webTemplate

TextBox "Bakgrundsfärg" – backColor

TextBox "Textfärg" – textColor

TextBox "Ramfärg" – borderColor

NumericalUpDown "Bilder horisontellt" - picCols

NumericalUpDown "Bilder vertikalt" – picRows

TextBox "Sökväg" – path

Controls:

Button butCreateWebpage labeled "Skapa websida" - createWebpage calls createPages() in Webpage.

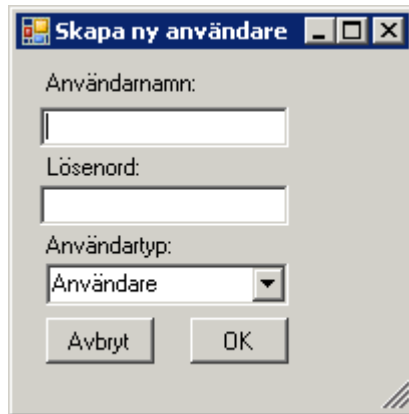
Button butSaveTemplate labeled "Spara formatmall" – saveWebTemplate calls constructor in WebTemplate.

Button butBrowse labeled "Bläddra", calls folderBrowserDialog()

References to Requirements Document

USE CASE: "SKAPA WEBBSIDA" p. 37

Dialogue: frmNewUser (Ny användare)



The image shows a Windows-style dialog box titled "Skapa ny användare". It has a standard title bar with minimize, maximize, and close buttons. The dialog contains three input fields: "Användarnamn:" (a text box), "Lösenord:" (a text box), and "Användartyp:" (a dropdown menu with "Användare" selected). At the bottom, there are two buttons: "Avbryt" and "OK".

Displayed when an administrator choose the menu item "Skapa ny användare".

Fields:

TextBox userName - "Användarnamn"

TextBox password - "Lösenord"

ComboBox privileges - "Användartyp"

Controls:

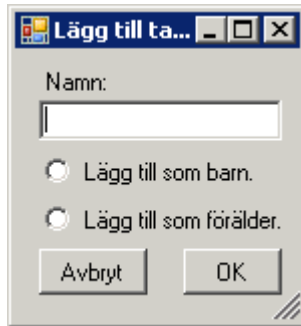
Button cancel - "Avbryt" disposes the form without making any changes to the system.

Button ok - "OK" calls ClientApp.newUser()

References to Requirements Document:

USE CASE: "SKAPA EN NY ANVÄNDARE" p.31

Dialogue: frmAddTag



Is called by form treeTags when user chooses "Lägg till tagg".

Fields:

TextBox name "Namn"

RadioButton "addAsChild" "Lägg till som barn"

RadioButton "addAsParent" "Lägg till som förälder"

Controls:

Button cancel - "Avbryt" disposes the form without making any changes to the system.

Button ok - "OK" calls constructor in TagNode.

References to Requirements Document:

USE CASE: "LÄGGA TILL EN NY TAGG" p.27

5 Design Details

5.1 Class Responsibility Collaborator (CRC) Cards

* Indirect collaboration via the relatively passive socket communication

Client application classes

CLASS CLIENTAPP	
Responsibilities	Collaborators
Process requests for viewing pictures. Receive recordsets containing pictures and create a PictureSet from it. Keep track of currently shown pictures in the GUI. Request full size pictures from the server. Process request for manipulating pictures. Create a picture.	PictureSet Manipulator CacheManager GUI ClientsideCom User

CLASS CLIENTSETTINGS	
Responsibilities	Collaborators
Knows root directory Knows server location Knows location of settings file Knows temporary files directory	ClientApp

CLASS CLIENTSIDECOM	
Responsibilities	Collaborators
Receive information from the server Send information to the server Parse information from the server	ServersideCom ClientApp NetworkObject

CLASS FILEMANAGER	
Responsibilities	Collaborators
Keep track of cache size Knows file download timestamp get(filename) = wrapped {Find temp/upload full size picture if new version available} Find thumbnail Upload a set of thumbnails Check version (filename) Save a file – send it to the server	ServerApp* ClientsideCom ClientApp Picture

CLASS WEBPAGE	
Responsibilities	Collaborators
Knows web page template Write files Call scale picture Get standard root directory	Album PictureSet Manipulator ClientSettings

GUI CLASSES

CLASS GUIFILESYSTEMTREE	
Responsibilities	Collaborators
Show the structure of the filesystem. Keep track of users' browsing history in the filesystem tree. Let the user expand and collapse the view of the filesystem. If the tree is a tag tree, display a menu letting the user manipulate the tag tree.	ClientApp PictureView TagTree AlbumTree

CLASS GUILOGIN	
Responsibilities	Collaborators
Lets a user login to the program.	User ClientApp

CLASS GUIUSERINFOFORM	
Responsibilities	Collaborators
Let the user change his/her info.	User

CLASS GUIPICTUREVIEW	
Responsibilities	Collaborators
Show a PictureSet as <ol style="list-style-type: none">1. thumbnails listing with mini info2. thumbnails in column list with more info3. one picture at a time with full info4. full screen picture with no info Select one or more pictures Knows sort order	PictureSet ClientSettings Album

CLASS GUIPICINFO	
Responsibilities	Collaborators
Display text fields for editable attributes If more than one picture is selected: Gray fields where the set have different values. Allow change. All will be given the same value.	Picture ClientApp PictureSet Photographer

CLASS GUISEARCHFORM	
Responsibilities	Collaborators
Display searchable fields. Let the user select Tags to search for in one or more tag trees.	ClientApp

CLASS GUITREE	
Responsibilities	Collaborators
Show the structure of the filesystem. Keep track of users' browsing history in the filesystem tree. Let the user expand and collapse the view of the filesystem. If the tree is a tag tree, display a menu letting the user manipulate the tag tree.	ClientApp PictureBox TagTree AlbumTree

Server application

CLASS SERVERAPP	
Responsibilities	Collaborators
Initiate picture manipulation Read server setting Handles picture files	ServerSettings Manipulator (server) DBConnection ServersideCom ClientApp* ServerLogFile

CLASS SERVERSIDECOM	
Responsibilities	Collaborators
Receive information from the client. Send information to the client. Parse information from the client.	ClientsideCom ServerApp DBConnection NetworkObject

CLASS DBCONNECTION	
Responsibilities	Collaborators
Open database connection Create SQL queries to the database Communicates with database Translates resultsets	ServerApp

CLASS SERVERSETTINGS	
Responsibilities	Collaborators
Knows database location Knows root directory (filesystem) Knows server port Reads settings file (server)	used by ServerApp used by ServerLogfile used by DBConnection

CLASS SERVERLOGFILE	
Responsibilities	Collaborators
Write logfile create new file	used by ServerApp

Common classes

CLASS WEBTEMPLATE	
Responsibilities	Collaborators
Knows the design of a webpage	

CLASS PICTURE	
Responsibilities	Collaborators
Knows picture information Knows picture file location Has a set of Tags Show a Thumbnail Show jpg-metainfo Add tag Remove tag	FileManager used by GUI used by PicturePile used by PictureSet

CLASS PICTURESET	
Responsibilities	Collaborators
Ordered list of Picture or AlbumPicture Add picture Remove picture Knows sort order Create custom sort order based on current order Find matching files	used by WebPage Picture AlbumPicture used by PicturePile used by ClientApp Filemanager

CLASS PICTUREPILE	
Responsibilities	Collaborators
Has a PictureSet Has a main picture Set main-picture Add picture to set Remove picture Disband pile Open picture list Open big/full picture view Open picture pile info form= main picture info form + few fields Add tag Remove tag Has tag	PictureSet Picture GUI-PictureView GUI-PicturePile-info-form

CLASS USER EXTENDS PHOTOGRAPHER	
Responsibilities	Collaborators
Knows password Knows work directory Knows latest search Knows latest folder Knows latest action Knows latest album Handles login	ClientsideCom GUILogin

CLASS PHOTOGRAPHER	
Responsibilities	Collaborators
Name Standard license agreement	User used by Picture

CLASS TAGNODE	
Responsibilities	Collaborators
Knows tag name Knows parent Has a unique ancestor Knows children	TagTree Database* - DBConnection*

CLASS TAGTREE	
Responsibilities	Collaborators
Knows the root tag of the tree. Knows name	Tag

CLASS ALBUM	
Responsibilities	Collaborators
Knows name Knows sort order Knows description Knows pictureset Knows parent Knows unique ancestor Knows children create a <i>new child</i> and add it in the list of children	PictureSet ClientSettings AlbumTree WebPage

CLASS ALBUMPICTURE	
Responsibilities	Collaborators
new (set link to a Picture object) set custom sort order index(int) int <- get custom sort order index set album text(string) string <- get album text()	Picture used by PictureSet used by Album

CLASS ALBUMTREE	
Responsibilities	Collaborators
Knows name	Album
Knows root album	

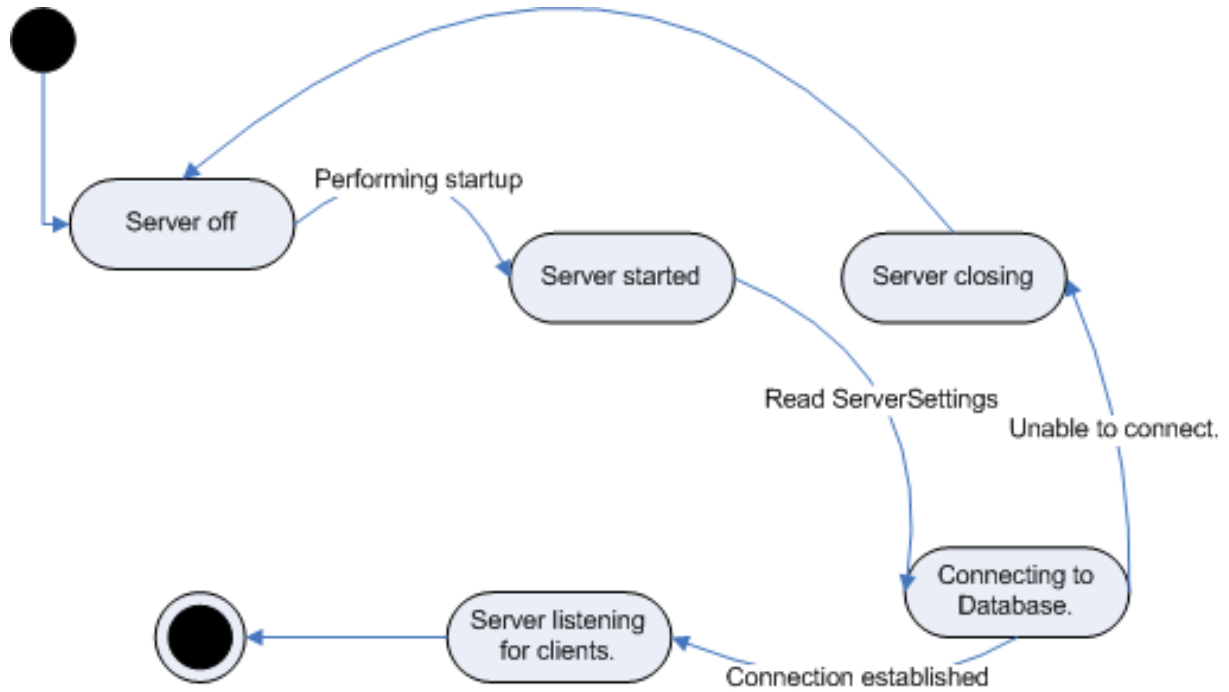
CLASS MANIPULATOR	
Responsibilities	Collaborators
Rotate picture.	Picture
Scale picture.	FileManager
Crop picture	

CLASS NETWORKOBJECT	
Responsibilities	Collaborators
Holds objects to send over the network	

5.3 State Charts

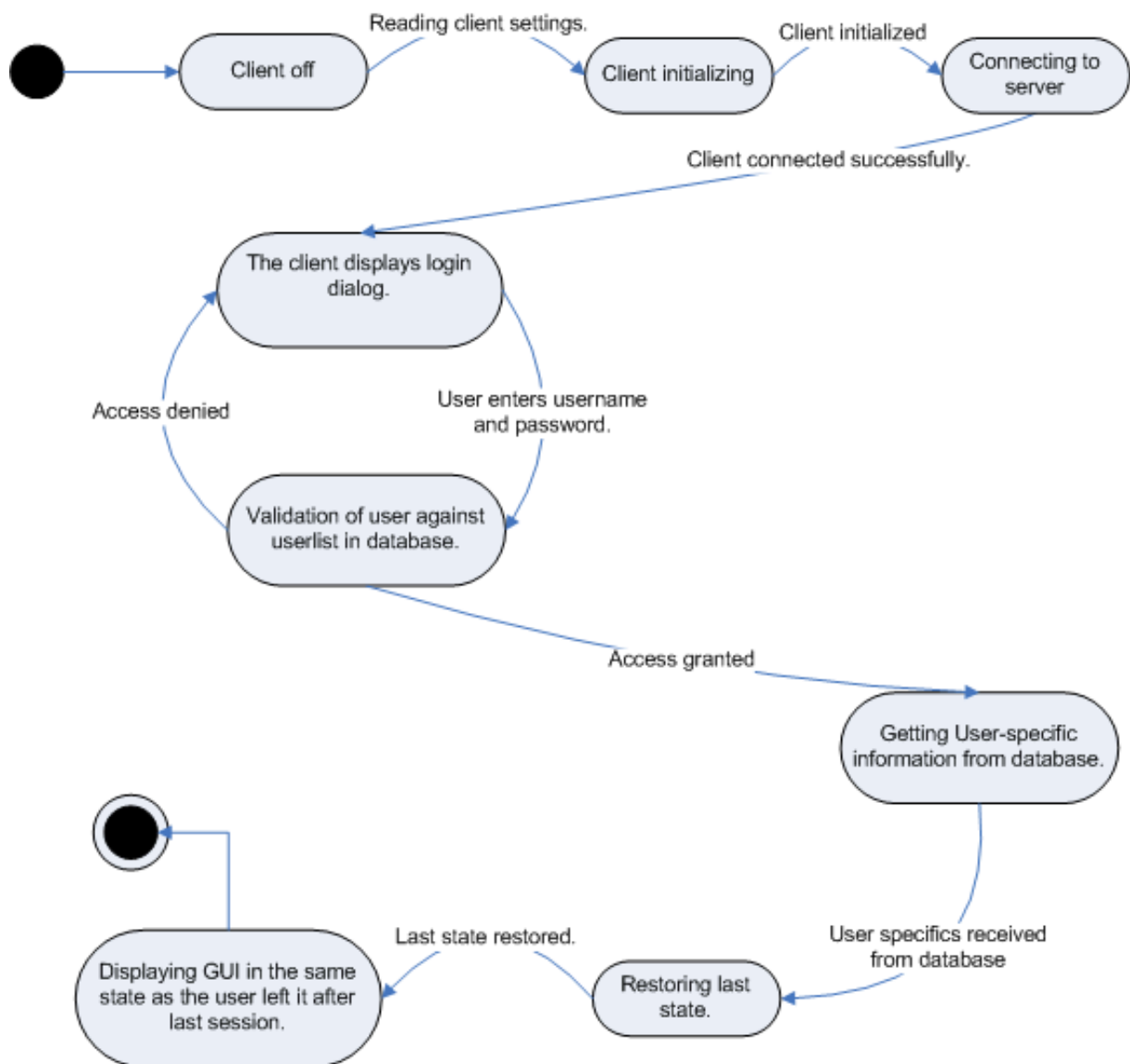
The Server application starts up

This shows going from initially no running server to a stable running server ready for connections.



Client startup

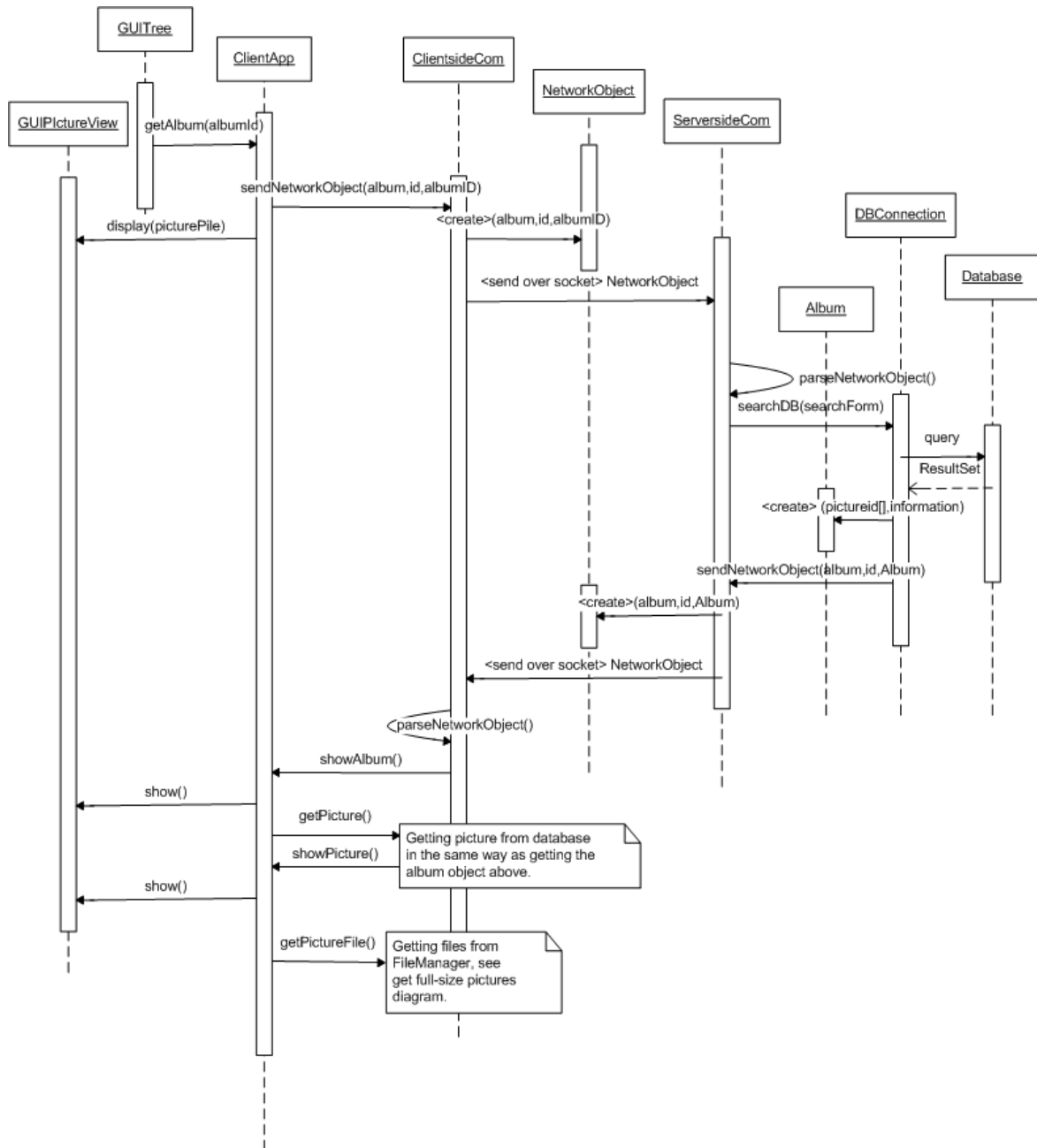
This shows going from the application not running to a stable connected application ready for user input.



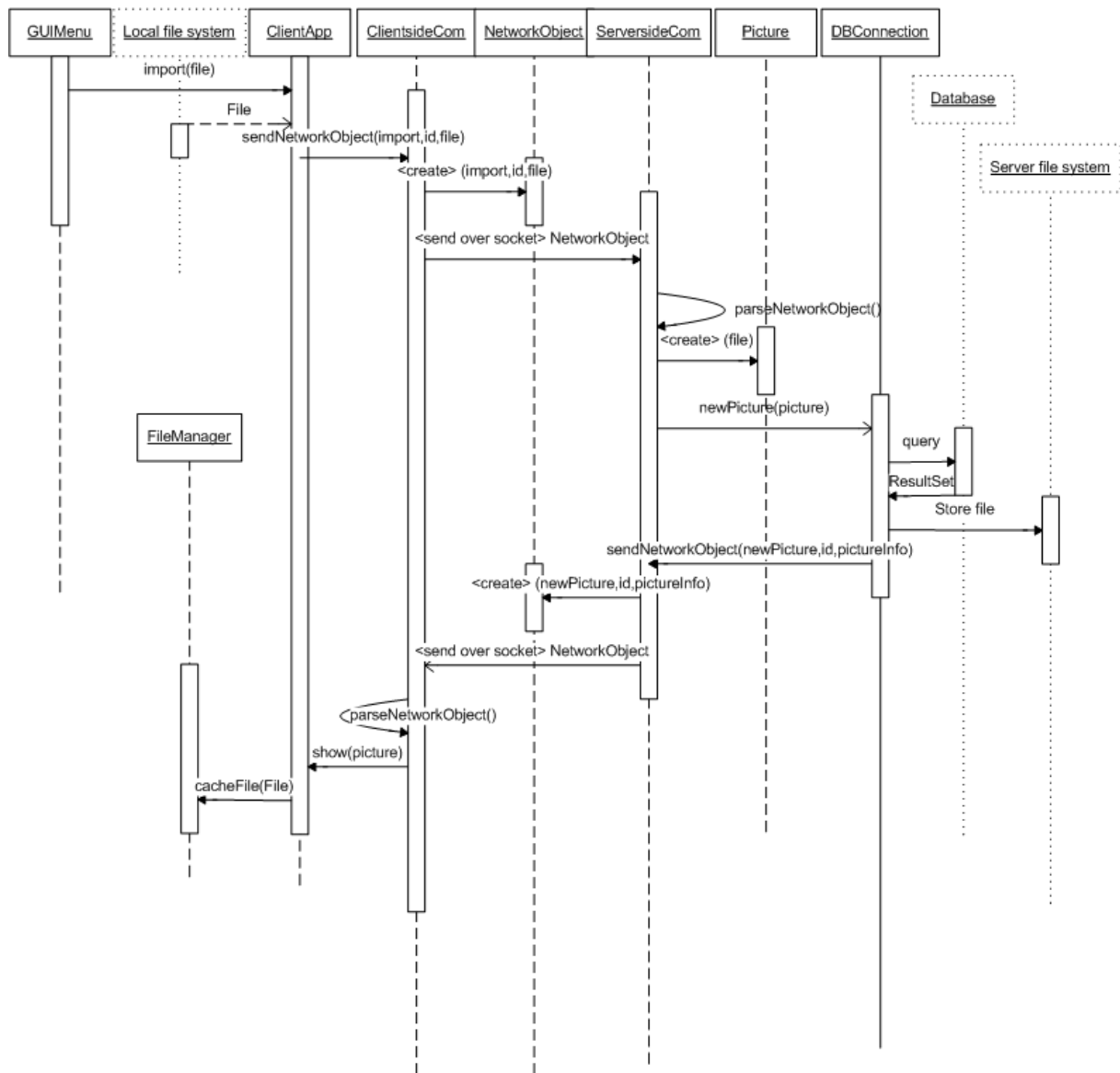
5.4 Interaction Diagrams

Change view

This diagram describes the process of opening an album from action initiation in the GUI to a stable representation in the GUI. References use case on page 19, Requirements Document (version 2).

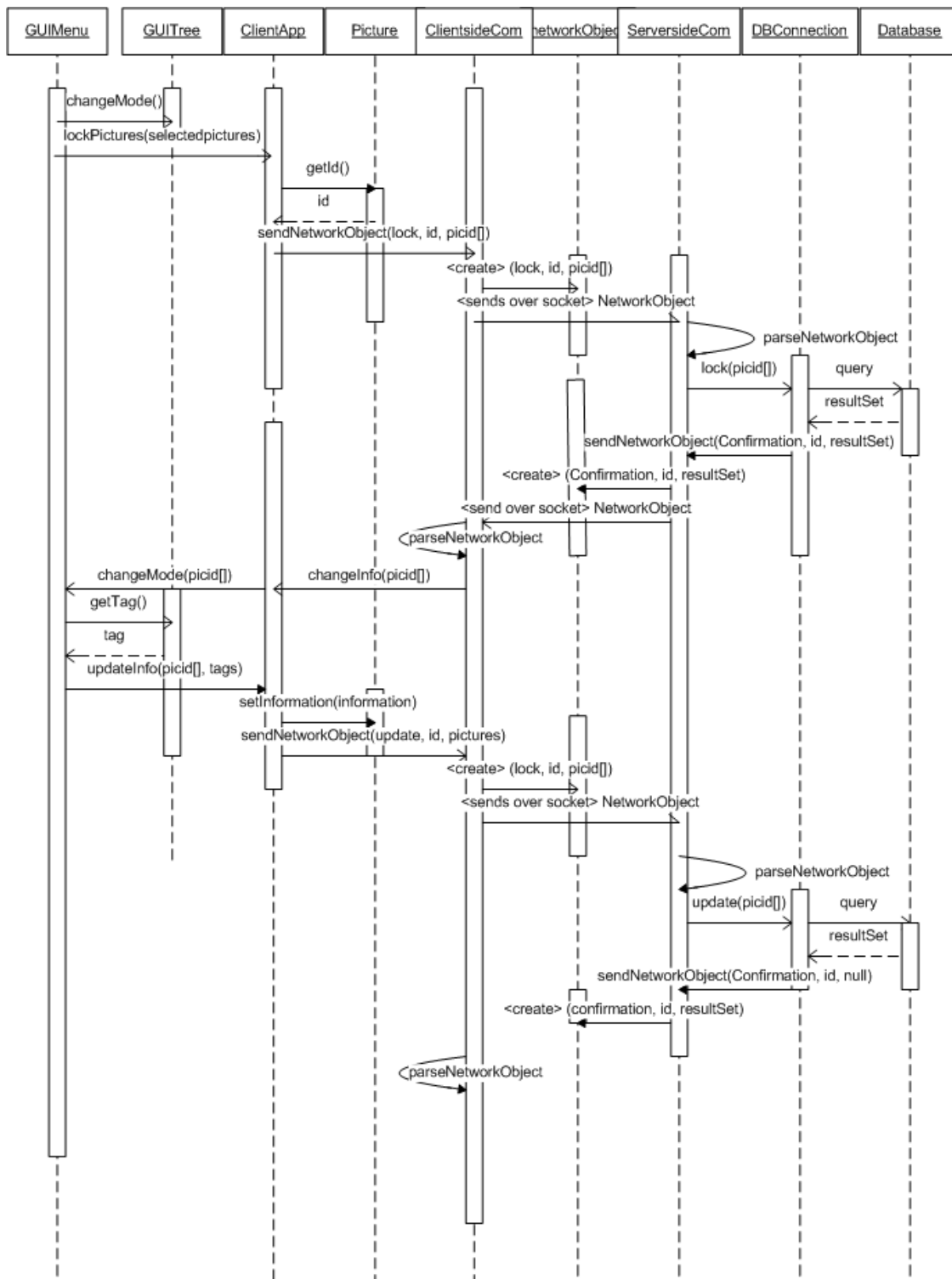


Import of pictures to KeyHole



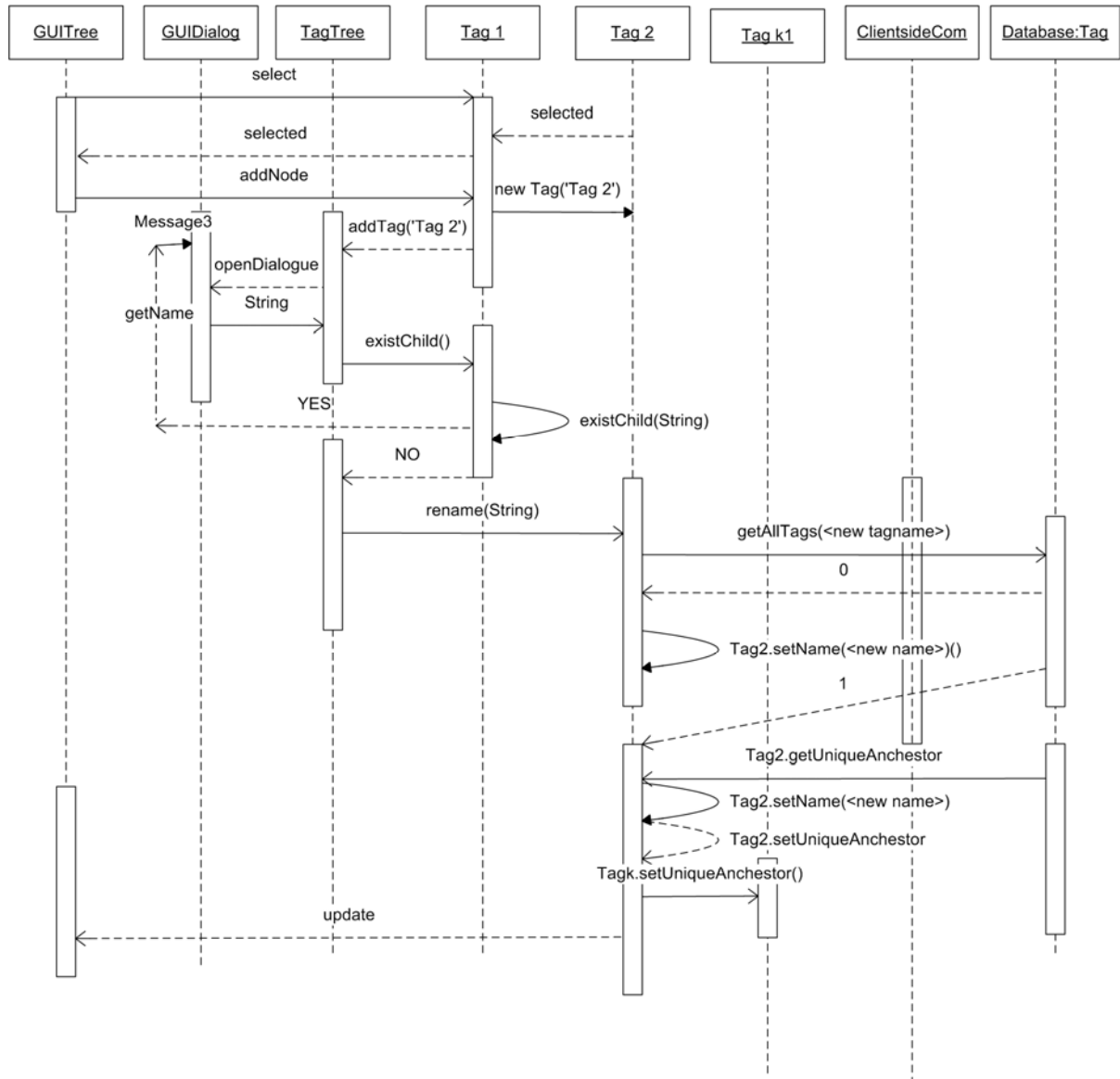
Mark a picture with a tag

References use case on page 25, Requirements Document (version 2).

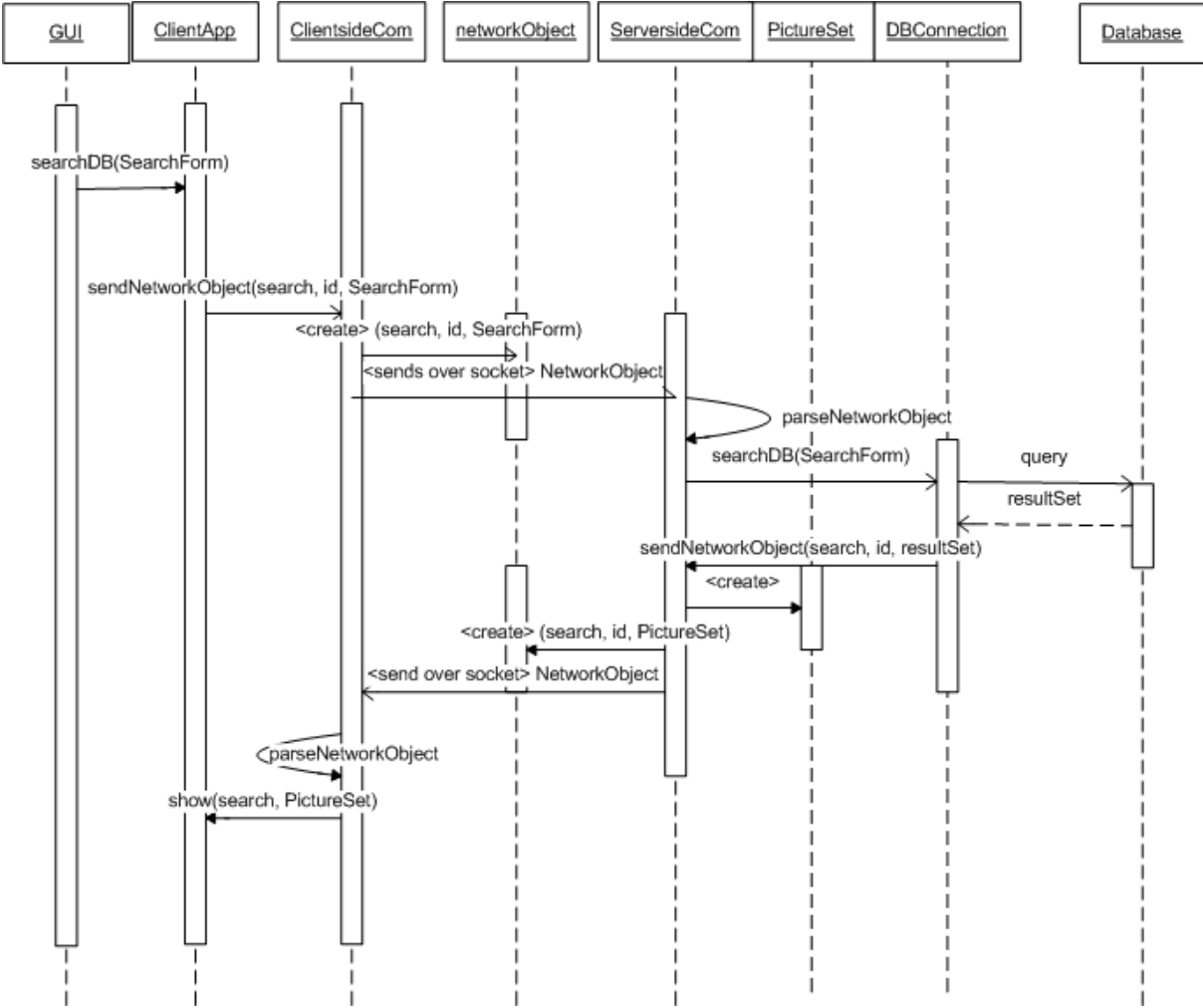


Add a new tag to an existing tag-tree

Here we have essentially skipped the socket communication; it is represented by "ClientsideCom". The focus is on client actions. References use case on page 27, Requirements Document (version 2).

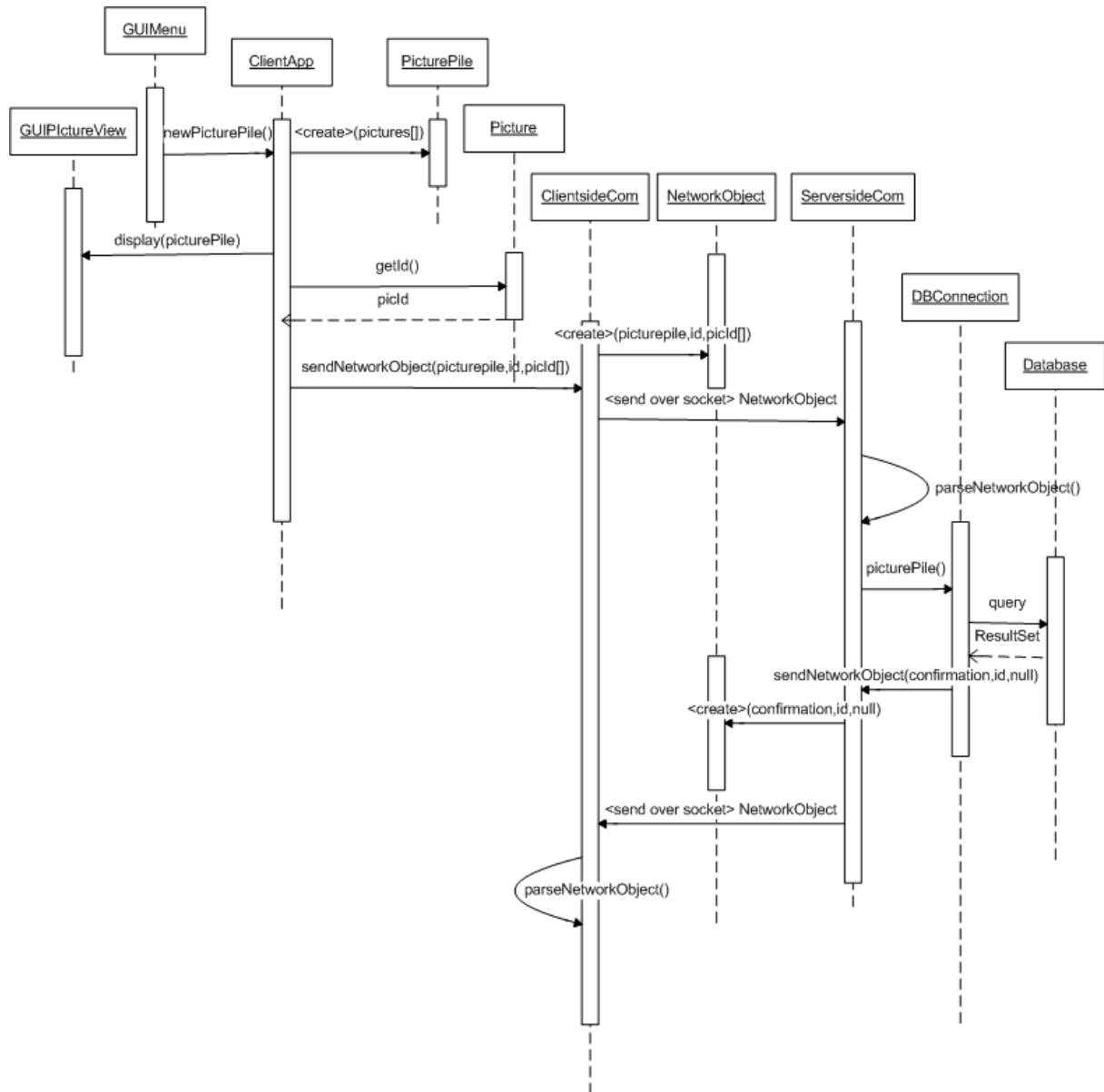


Search by means of tags

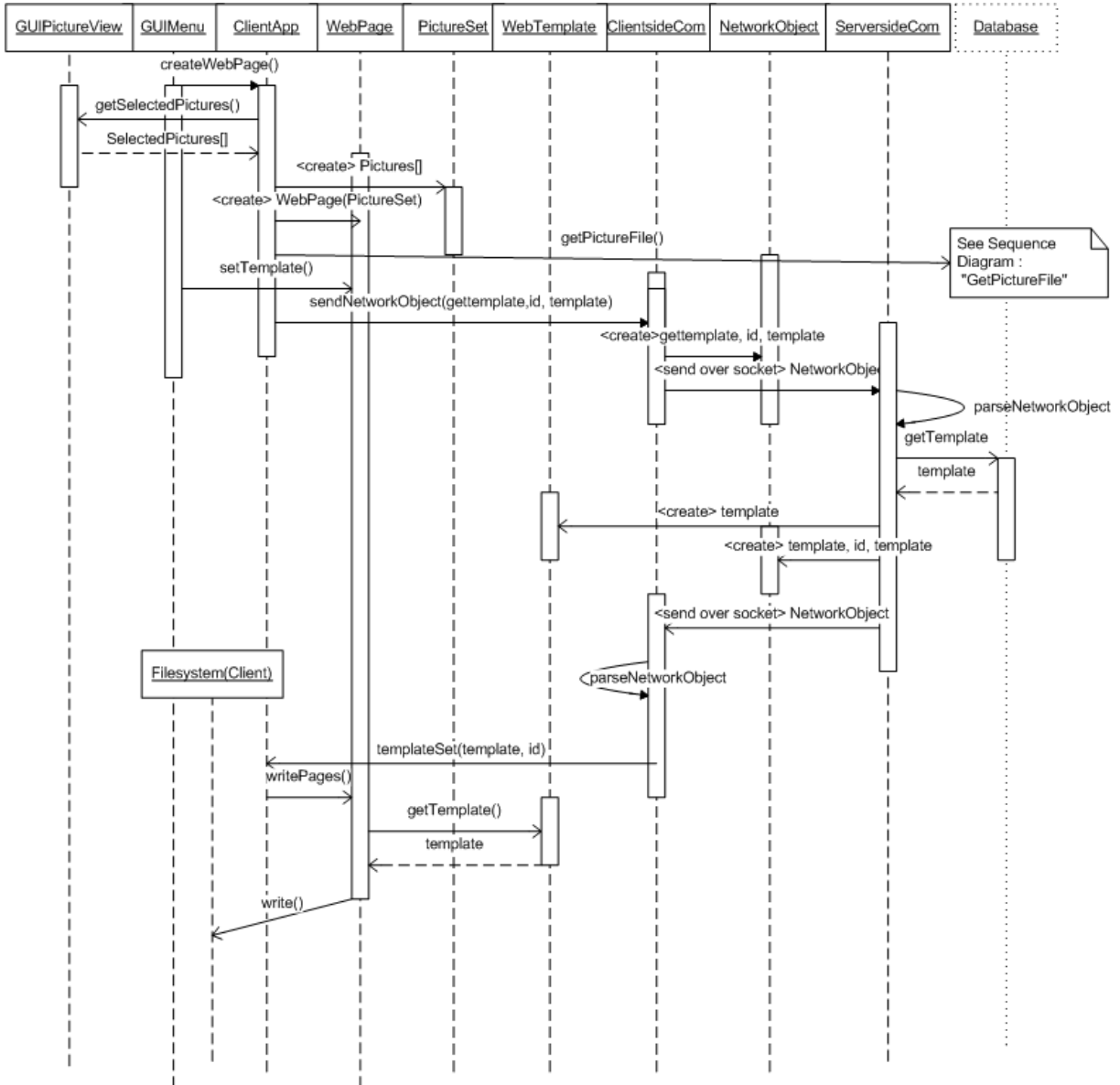


Create a picture pile

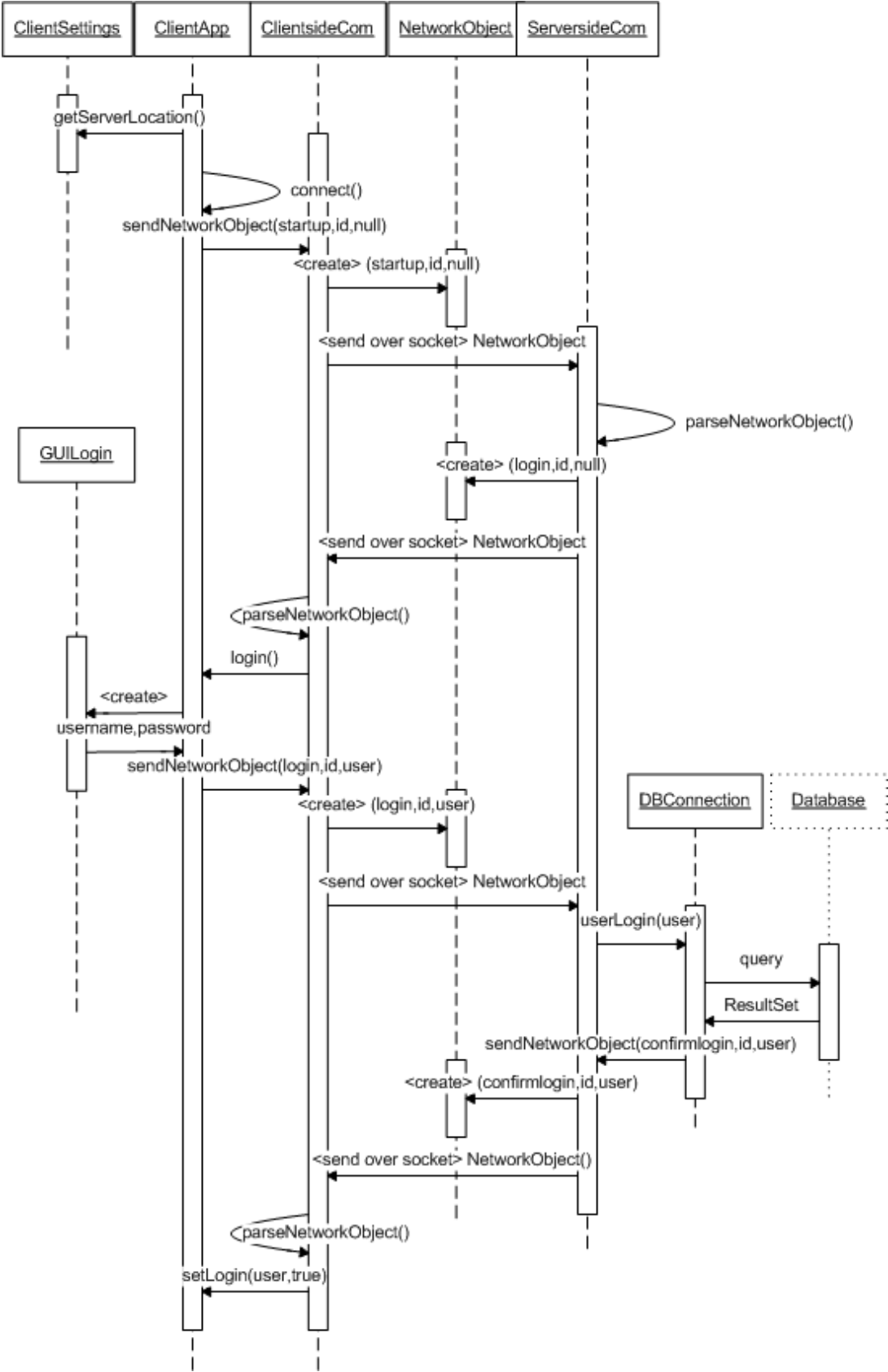
References use case on page 32, Requirements Document (version 2).



Creating a webpage



Starting KeyHole



5.5 Detailed Design

Server classes

Class ServerApp

ServerApp is the main class always running on the server machine. It starts the other classes.

FIELDS

Manipulator manipulator, used for manipulating pictures on the server.

ServerSettings serverSettings, keeps track of server settings.

DBConnection dBConnection, provides communication with the database.

ServerLogfile serverLogfile, logs all important actions.

ServersideCon serversideCon, communicates with the clients

METHODS

main()

Initiates server and listens for events.

Class ServerLogFile

Keeps track of the log using files on the server.

FIELDS

String path, the path to the current log file.

METHODS

boolean writeLog(String event)

Appends the string to the log file. Called by ServerApp. Returns true if operation succeeds.

Pre: File must exist and be writable.

Post: File has been updated with the new event appended.

boolean newFile(String name)

Creates a new log file. Sets the field 'path' to this file. This method should run automatically at the start of each day or on startup.

Pre: Application must have administrable rights on the machine.

Post: A new file has been created. ServerLogFile.path has been changed to this file.

Class ServerSettings

Keeps track of the server settings file.

FIELDS

String dBLoc, database location

String rootDir, file system root directory

int serverPort, server port number

String settingsFile, the path to the settings file.

METHODS

readSettings

boolean readSettings()

Reads the server settings file and import the values into the fields. Called by ServerApp.

Pre: File must exist and be readable.

Post: All server settings fields has been updated with values located in the file.

saveSettings

boolean saveSettings()

Saves the server settings file with the values currently in the class.

Pre: File must exist and be writable.

Post: Values in the file has been updated with the current settings.

Class DBConnection

Connects both ServersideCom and ServerApp with the database. Translates SearchForms and other requests to SQL-queries and then translates the ResultSets to objects before returning them.

FIELDS

none

METHODS

initDB

boolean initDB(String address)

Initiates the database and checks it for inconsistency.

Called by: ServerApp.

Pre: Database must be open and running at the specified address.

Post: Database has been checked and initialized.

searchDB

PictureSet searchDB(SearchForm searchForm)

Translates a SearchForm to a SQL query and sends it to the database. Then translates the corresponding resultset to a PictureSet. Called by ServerSideCom.

Pre: Database must be open for access.

Post: The relevant pictures has been acquired from the database and returned.

Called by: ServersideCom

References back to Requirements Document:

USE CASE: "AVANCERAD SÖKNING" p.29

USE CASE: "SÖKNING MED HJÄLP AV TAGGAR" p.29

userLogin

boolean userLogin(String userID, String password)

Tries to log in a user. UserID and password is checked with the database. Called by ServerSideCom. Returns true if login succeeds.

Pre: Database must be open for access.

Post: The user has been updated as "logged in" in the database.

Called by: ServersideCom

References back to Requirements Document:

USE CASE: "STARTA KEYHOLE" p.40

newPicture

boolean newPicture(Picture pic)

Creates a new picture in the database. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The picture has been added to the database and given a unique picture ID. The file has been copied to the server.

Called by: ServersideCom

References back to Requirements Document:

USE CASE: "INLÄSNING AV BILDER" p.21

USE CASE: "ÄNDRA BILDINFORMATION" p.36

USE CASE: "TILLDELNING AV TAGG FRÅN PRIMÄRA TAGGTRÄDET" p.25

USE CASE: "VÄLJ TAGGAR FRÅN ANDRA TAGGTRÄD" P.26

newPicturePile

boolean newPicturePile(PicturePile picturePile)

Creates a new picture pile. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The picture pile has been created in the database and been given a unique ID.

Called by: ServersideCom

References back to Requirements Document:

"ATT ARBETA MED FOTOHÖGAR" p.15

newAlbum

boolean newAlbum(Album album)

Creates a new album in the database. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The album has been created in the database and been given a unique ID.

Called by: ServersideCom

References back to Requirements Document:

"ATT ARBETA MED ALBUM" p.14

USE CASE: "SKAPA ETT ALBUM" p.23

USE CASE: "SPARA EN SÖKNING SOM ETT ALBUM" p.23

USE CASE: "ÄNDRA EGENSKAPER FÖR ETT ALBUM" p.24

USE CASE: "ÄNDRA SORTERINGEN AV BILDERNA I ETT ALBUM" p.24

USE CASE: "TA BORT BILDER UR ETT ALBUM" p.24

deleteObject

boolean deleteObject(long id)

Deletes entries in the database.

Pre: Database must be accessible. Object must exist.

Post: Object has been removed from database.

Called by: ServersideCom

References back to Requirements Document:

Use Case "RADERA EN BILD" p.22

Use Case "RADERA ETT ALBUM" p.25

Use Case "TA BORT EN TAGG UR TAGGTRÄDET" p.28

newTag

boolean newTag(Tag tag)

Creates a new tag in a tag tree. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The tag has been created in the database and been given a unique ID.

Called by: ServersideCom

References back to Requirements Document:

USE CASE: "LÄGGA TILL EN NY TAGG" p.27

newTagTree

boolean newTagTree(TagTree tagTree)

Creates a new tag tree. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The tag tree has been created in the database and been given a unique ID.

Called by: ServersideCom

newUser

boolean newUser(User user)

Creates a new user. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The user has been created in the database and been given a unique ID.

Called by: ServersideCom

newPhotographer

boolean newPhotographer(Photographer photographer)

Creates a new photographer. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The photographer has been created in the database and been given a unique ID.

Called by: ServersideCom

newAlbumTree

boolean newAlbumTree(AlbumTree albumTree)

Creates a new album tree. Returns true if operation succeeded.

Pre: Database must be accessible.

Post: The album tree has been created in the database and been given a unique ID.

Called by: ServersideCom

lock

boolean lock(long[] iD)

Tries to lock the pictures for writing. Checks with the database if it is already locked. Returns true if operation succeeded.

Pre: Database must be accessible, iD must exist.

Post: The object has received a lock if it hadn't.

Called by: ServersideCom

isLocked

boolean isLocked(long[] iD)

Sees if an object in the database is locked. Returns true if it is.

Pre: Database must be accessible, iD must exist.

Post: Nothing has changed.

Called by: ServersideCom

Class ServersideCom

FIELDS

Socket socket - The socket which connects to the client.

KHServer.DBConnection con - A connection object which connect to the database.

CONSTRUCTOR

ServersideCom(java.net.Socket socket)

Parameters:

socket - the socket which connects to the client.

METHODS

parseNetworkObject

public void parseNetworkObject(NetworkObject receivedObject)

Receives NetworkObjects from run() and determines its contents and what to do with it.

It will then call the suitable method in DBConnection for each incoming command.

Pre-conditions: A DBConnection must exist.

Called by: ServersideCom.run()

Calls: suitable methods in DBConnection.

sendNetworkObject

public boolean sendNetworkObject(java.lang.String command, int id, java.lang.Object result)

Creates a NetworkObject and sends it over the socket to the client.

Called By: DBConnection

Parameters:

command – What to do with the result.

id – An unique id for this transfer.

result – The object to execute the command on.

Returns:

true if sending object over socket succeeded.

run

public void run()

Establishes communication with the connecting client.

Receives and sends NetworkObjects over a socket connection.

Calls: ServersideCom.parseNetworkObject()

Specified by:

run in interface java.lang.Runnable

Overrides:

run in class java.lang.Thread

readFile

public java.awt.Image readFile(java.lang.String filepath)

Reading a picture from the servers file system.

The file must exist at the specified filepath.

Called By: parseNetworkObject()

Parameters:

filepath - the location of the file.

Returns:

the image

writeFile

Image writeFile(Picture pic)

Writing a picture to the servers file system.

Called By: parseNetworkObject()

Parameters:

Pic – the picture to be added or changed.

References back to Requirements Document:

Use Case "INLÄSNING AV BILDER" p.21

Client classes

Class FileManager

FileManager keeps track of files cached in the file system of the client.

FIELDS

none

METHODS

getFile

public void getFile(Picture picture)

If the picture is available in the cache it compares the local file timestamp with the timestamp at the server. If the server has a newer file it is retrieved from server, otherwise the file in the local cache is returned. If the picture is not available in the cache it is retrieved from the server, and placed in the cache.

Calls: ClientsideCom.sendNetworkObject, cacheFile

Called by: ClientApp

Parameters:

picture - Picture who's image file we want to retrieve.

getThumbnail

public void getThumbnail(Picture picture)

If the thumbnail is available in the cache it compares the local file timestamp with the timestamp at the server. If the server has a newer file it is retrieved from server, otherwise the file in the local cache is returned. If the thumbnail is not available in the cache it is retrieved from the server, and placed in the cache.

Calls: ClientsideCom.sendNetworkObject, cacheFile

Called by: ClientApp

Parameters:

picture - Picture whos thumbnail file we want to retrieve.

storeFile

public void storeFile(Picture picture, java.io.File file)

Sends the file to server.

Calls: ClientsideCom.sendNetworkObject

Parameters:

picture – The Picture object of the file.

file - the file to be stored in cache and on server

cacheFile

public void cacheFile(Picture picture, java.io.File file)

Writes image file to the local cache with the filename picture.getId().

Calls: ClientApp.setFile

Called by: ClientsideCom.parseNetworkObject

Parameters:

picture – The Picture object of the file.

file - the file to be stored in cache and on server

cacheThumbnail

public void cacheThumbnail(Picture picture)

Writes the thumbnail image file to the local cache with filename picture.getId() + ".thumbnail". Calls: ClientApp.setFile Called by: ClientsideCom.parseNetworkObject

Parameters:

picture –

Class ClientsideCom

FIELDS

Socket socket - The socket which connects to the server.

CONSTRUCTOR

ClientsideCom(java.lang.String serverloc, int port)

Creates a new ClientSideCom which will try to establish a connection with the server.

There must be a server running.

METHODS

parseNetworkObject

public void parseNetworkObject(NetworkObject receivedObject)

Receives NetworkObjects from run() and determines its contents and what to do with it.

It will then call the suitable method in the package KHClient for each incoming command.

Called by: ClientsideCom()

Calls: suitable methods in package KHClient.

Parameters:

receivedObject - the NetworkObject received from the server.

sendNetworkObject

public boolean sendNetworkObject(java.lang.String command, int id, java.lang.Object sendObject)

Creates and sends a NetworkObject over the socket to the client.

Called By: ClientApp

Parameters:

command – What to do with the object.

id – An unique id for this transfer.

sendObject – The object to execute the command on.

Returns:

true if sending object over socket succeeded.

Class ClientApp

FIELDS

ClientSettings clientSettings - the clients settings.

KHClient.ClientsideCom - clientsideCom the connection to the server.

User user - this users information.

boolean loggedin - shows if the user is logged in successfully or not.

boolean loggedindenied - shows if the user has been denied an acceptance or not.

METHODS

public boolean connect()

Creates a ClientsideCom.

Calls: ClientSettings.getServerLocation(), ClientsideCom(serverlocation, port)

References back to Requirements Document: USE CASE: "STARTA KEYHOLE"

Returns: true if connection has established. Shows dialog if not.

public void run()

runs

Calls: ClientApp(), connect(), login(), setupGUI()

Called by: main()

public void setupGUI(User user)

Initiates the GUI with the user settings. Executes the latest query.

Calls: ClientsideCom.sendNetworkObject()

Called by: run()

public void show(String command, Object object)

Determines how to show the result of the query.

Called by: ClientsideCom.parseNetworkObject().

Parameters:

command - What object to show and how.

object - The object to show.

public void setLogin(User user, boolean loggedin)

Sets weather the user could login or not.

Gets the userinformation if user logged in.

Called by: ClientsideCom.parseNetworkObject(),

Parameters:

user - The user information from the database. Null if not logged in.

loggedin - Weather the user is logged in or not.

public boolean login()

Creates and shows the login dialog.

Calls: ClientsideCom.sendNetworkObject()

Called by: run()

References back to Requirements Document: Use Case "STARTA KEYHOLE" p. 40

Returns: true if login successful.

public void lockPictures(PictureSet pictureSet)

Locks the pictures in the set if they are unlocked.

Calls: ClientsideCom.sendNetworkObject()

References back to Requirements Document: "USE CASE: FLERA ANVÄNDARE MODIFIERAR SAMMA BILD"

Parameters:

pictureSet - The pictures to lock.

public void updateInfo(PictureSet pictureSet)

Updates a set of pictures with information from GUIPicInfo form.

Calls: Picture.setInformation(), PictureSet.getId(), ClientsideCom.sendNetworkObject()

Parameters:

pictureSet – the set of pictures to update.

public void import(BufferedImage image)

Sends the image to the server.

Calls: ClientSideCom.sendNetworkObject()

Parameters:

image – the image to send.

References back to Requirements Document: "USE CASE: INLÄSNING AV BILDER TILL KEYHOLE" p.21

public static void main(java.lang.String[] args)

Calls run() . Sets up the GUI.

Class WebPage

FIELDS

kHClient.PictureSet pictureSet - The pictures to be included in the webpage.

WebTemplate template - The template to use.

CONSTRUCTOR

public WebPage(kHClient.PictureSet pictureSet)

When the user chooses to export a set of pictures to a webpage a Webpage object is created.

Called by: ClientApp

References back to Requirements Document: Use Case "Skapa Webbsida" p. 37

Parameters:

pictureSet - - the PictureSet to use when creating the webpage.

METHODS

getWebTemplate

public WebTemplate getWebTemplate()Called by: ClientApp

References back to Requirements Document: Use Case "Skapa Webbsida" p. 37

Returns:

templates - the selected template.

setWebTemplate

public void setWebTemplate(WebTemplate template)

Called by: ClientApp

References back to Requirements Document: Use Case "Skapa Webbsida" p. 37

Parameters:

template - an instance of the WebTemplate class.

writePicturesToFiles

public boolean writePicturesToFiles()

Gets the full-size pictures from the server's file system and saves them to the output folder in the local file system.

Validity Checks, Errors, and other Anomalous Situations: Must be able to write to the local file system.

Called by: createPages()

References back to Requirements Document: Use Case "Skapa Webbsida" p. 37

Returns: true if write succeeded.

createPages

public boolean createPages()

Creates the webpage files and saves them to the client's file system.

The files are created with the current settings.

The output is typically a set of html-files and a style sheet-file.

Validity Checks, Errors, and other Anomalous Situations:

Must be able to write to the local file system.

Calls: Manipulate.scale();

Called by: ClientApp

Returns:

true if write succeeded.

Class ClientSettings

FIELDS

String rootDirectory - The root directory that the client uses.

String serverLocation - Adress of the server.

String temporaryFile - Directory for temporary files.

METHODS

public boolean saveSettings()

Writes the current information to the settings file.

Called by: ClientApp

Returns:

true if operation succeeded.

public String getServerLocation()

Returns:

the serverLocation

public void setServerLocation(String serverLocation)

Parameters:

serverLocation - the serverLocation to set

Common classes

Class SearchForm

Provides an easy way of submitting a request for certain pictures from the database.

FIELDS

Tag[] tagList

String album

String photographer

String date

String modDate

String title

String filename

METHODS

SearchForm

SearchForm(Tag[] tagList, String album, String photographer, String date, String modDate, String title, String fileName)

Constructs a SearchForm, setting any unused fields to null.

Pre & post: none

Class Picture

FIELDS

long id – picture Id

Photographer photographer - The photographer of the picture.

Tag[] tags - Tags for the picture.

String filename – The file name of the picture.

String filepath – Location of the picture.

String changeDate – Date of last modification of the picture.

Int pixelHeight

Int pixelWidth

long mmHeight

long mmWidth

String caption – A caption of the picture.

String description – A description of the picture.

Boolean manipulated – True if the picture has been manipulated.

Int copywriteAgreement – The copy write agreement for the picture.

String diaphragm

long exposureTime

long focalLength

Boolean flash

BufferedImage thumbnail

Int thumbnailHeight

Int thumbnailWidth

METHODS

Boolean addTag(Tag t)

Post: the picture has received the tag

long getId()

Returns:

pictureId of this picture

Metadata getMetadata()

Returns: The metadata of this picture.

Photographer getPhotographer()

Returns: The photographer of this picture.

Tag[] getTags()

Returns: A list of tags for this picture.

Boolean hasTag(Tag t)

Parameters:

t - Tag to be added.

Returns: true if the picture has tag t, false otherwise.

Boolean removeTag(Tag t)

Parameters:

t - Tag to be removed.

Returns:

true if tag was removed.

Class PicturePile

FIELDS

long mainPictureId - the main picture pictureId

PictureSet ps – the pictures

METHODS

public long getMainPictureId()

Returns:

main picture pictureId

public void setMainPictureId(long mainPictureId)

Called by: ClientApp

Parameters:

mainPictureId - new main picture

public boolean addPicture(Picture picture)

Add picture to pile.

Calls: PictureSet.addPicture

Parameters:

picture - to be added.

Returns: true if picture was added.

public boolean removePicture(Picture picture)

Remove picture from pile.

Calls: PictureSet.removePicture

Parameters:

picture - to be removed.

Returns: true if picture was removed.

Class PictureSet

FIELDS

long id – the PictureSet ID

java.util.List<Picture> pictures - The pictures in the PictureSet

int sortOrder Picture sort order

METHODS

Boolean add(Picture picture)

Add picture to PictureSet.

long getId()

int getSortOrder()

Returns the way in which the PictureSet currently is sorted.

Boolean remove(Picture picture)

Remove picture from PictureSet.

Boolean setSortOrder(int sortOrder)

Set how the pictures in the album are sorted.

Class TagNode

A node of a hierarchical tree. There may be any number of children to a node.

FIELDS

Attributes corresponding to the attributes of the database table Tag:

string name – the word used as a tag

TagNode parent – corresponding to the parent id

string uniqueAncestor – the nearest unique tag-name towards the root

The name+uniqueAncestor must be unique, name may occur at several places in this or any tree.

List <TagNode> children - The set of children are found through recursive calls in the database tables. They are stored as a list attached to this TagNode.

CONSTRUCTOR

TagNode(name, parent, uniqueAncestor)

METHODS

setName

public void setName(String sName)

Simple set-method, without duplet check.

getName

public String getName()

Get the property.

setUniqueAncestor

public boolean setUniqueAncestor(String sName)

Set 'uniqueAncestor' property of this TagNode. This function involves possibly several database queries.

Called by:

Calls:

Argument: sName

Returns: true if successful

TagNode addNode(void)

Creates a new TagNode with the temporary name “new tag #” and with this current as parent. (# should be replaced by a number, thus making the tag unique.)

bool existChild(String sName)

Investigates the set of children to this current node, if there is any TagNode with the same name as the parameter. In that case it returns true.

Tag[] getChildren()

Returns the list of children.

bool rename(String newname)

This method tests if the parameter string makes a unique name or not. If not it will attempt to change the attribute “uniqueAncestor” in order to make the combination unique. This will engage several external method calls and ultimately make a series of queries to the database. If this effort fails (it may fail in some rare situation) the method returns false.

If the method is successful it returns true.

String getUniqueAncestor()

This will engage make a series of queries to the database. If this effort fails (it may fail in some rare situation) the method returns an empty string.

Called by this.setName().

Returns: If the method is successful it returns the name of an ancestor TagNode. If not it returns an empty string.

String askName()

Opens a GUI-dialogue that asks for a name. Returns a string.

Class TagTree

FIELDS

Attributes corresponding to the attributes of the database table TagTree:

String **name** – equals the name of the root TagNode.

TagNode **root** – the beginning of the tree.

TagNode **current** – currently selected

METHODS

none

Class Album

This class wraps a permanent set of picture called an album. The same picture may occur in several albums. An Album object may also be a node in the AlbumTree.

FIELDS:

String name – duplets allowed.

String uniqueAncestor – there must not exist duplets of the combination name and uniqueAncestor.

String description – may be null.

enumerical orderMethod – it can be ‘name’, ‘filename’, ‘date’, ‘custom’

PictureSet ps – may be null, e.g. an empty album or an album tree node organizing several albums.

Album parent – albums are organized in a hierarchical tree.

List <Album> children – is null for a leaf.

CONSTRUCTOR

public Album(string name, String description)

Creates an empty album.

Called by: ClientApp

public Album(Picture[] pictures)

Creates a new album from an array of pictures.

Called by: ClientApp

Parameters:

pictures -

public Album(long[] pictureIds)

Creates a new album from an array of pictureIds.

Called by: ServerApp

Parameters:

pictures -

METHODS

renumber(int n, int m)

Set the property “customSort” to the n:th picture argument. Adjusts all following AlbumPicture’s customSort one step.

public String getDescription()

Returns:

description Description of the album

public void setDescription(String description)

Parameters:

description - new description

public String getName()

Returns:

name of the album

public void setName(String name)

Parameters:

name - new name of album

public boolean addPicture(Picture picture)

Add picture to album.

Calls: PictureSet.addPicture

Parameters:

picture - to be added.

Returns: true if picture was added.

public boolean removePicture(Picture picture)

Remove picture from album.

Calls: PictureSet.removePicture

Parameters:

picture - to be removed.

Returns:

true if picture was removed.

public PictureSet getPictureSet()

Retrieve the PictureSet backing Album with current sorting order.

Returns: A PictureSet containing the pictures of the Album.

Class AlbumPicture

FIELDS

Picture picture – pointer to a picture

String text – There may be a unique text for this picture in this album.

int customSort

METHODS

addPicture(Picture)

addPicture(PicturePile)

Class AlbumTree

There is only one instance of this in the client. As opposed to the case with tags.

FIELDS

String name

Album root

METHODS

none

Class Manipulator

METHODS

public boolean rotate(Picture picture, Enum degree)

Rotates a picture with the given number of degrees (can only be 90, 180 or 270).

The picture at the filepath of the picture must exist and be of a supported format.

Post-conditions: The picture has been rotated in the cache.

Called by: ClientApp

References back to Requirements Document: Use Case "Roter bild" p. 36

Parameters:

picture - the picture that should be rotated.

degree - the number of degrees the picture should be rotated.

Returns: true if operation succeeded.

public boolean rotate(String filepath, Enum degree)

Rotates a picture with the given number of degrees (can only be 90, 180 or 270).

The picture at the given filepath must exist and be of a supported format.

Post-conditions: The picture has been rotated in the filesystem.

Called by: ServerApp

References back to Requirements Document: Use Case "Roter bild" p. 36

Parameters:

filepath - filepath to the picture that should be rotated.

degree - the number of degrees the picture should be rotated.

Returns: true if operation succeeded.

public boolean scale(Picture picture, int width, int height)

Scales the picture to the new given size.

The picture at the file path of the picture must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the cache.

Called by: User request to rotate, `WebPage.createWebPage()`

References back to Requirements Document: Use Case “Skala bild” p. 35

Parameters:

picture - the picture to be scaled.

width - the new width of the picture.

height - the new height of the picture.

Returns: true if operation succeeded.

public boolean scale(String filepath, int width, int height)

Scales the picture to the new given size.

The picture at the given filepath must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the filesystem.

Called by: `ServerApp`

References back to Requirements Document: Use Case “Skala bild” p. 35

Parameters:

filepath - the filepath of the picture to be scaled.

width - the new width of the picture.

height - the new height of the picture.

Returns:

true if operation succeeded.

public boolean scale(Picture picture, int percent)

Converts percent-information into width and height of the picture and calls

The picture at the filepath of the picture must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the cache.

Called by: User request to rotate, `WebPage.createWebPage()`.

Calls: `scale()`

References back to Requirements Document: Use Case “Skala bild” p. 35

Parameters:

picture - the picture to be scaled.

percent - percent to be scaled to.

Returns:

true if call to scale returns true.

public boolean scale(String filepath, int percent)

Converts percent-information into width and height of the picture and calls

function `scale(String filepath, int width, int height)`.

The picture at the given filepath must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the filesystem.

Called by: `ServerApp`.

Calls: scale()

References back to Requirements Document: Use Case "Skala bild" p. 35

Parameters:

filepath - the filepath of the picture to be scaled.

percent - percent to be scaled to.

Returns:

true if call to scale returns true.

public boolean crop(String filepath, int xUpleft, int yUpleft, int xDownright, int yDownright)

Cuts out a part of a picture. The picture at the given filepath must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the filesystem.

Called by: ServerApp.

Calls: scale()

References back to Requirements Document: Use Case "Beskära bild" p. 35

Parameters:

filepath - the filepath of the picture to be cropped.

xUpleft - xvalue at the top left corner.

yUpleft - yvalue at the top left corner.

xDownright - xvalue at the bottom left corner.

yDownright - yvalue at the top bottom corner.

Returns: true if operation succeeded.

public boolean crop(kHClient.Picture picture, int xUpleft, int yUpleft, int xDownright, int yDownright)

Cuts out a part of a picture.

The picture at the filepath of the picture must exist and be of a supported format.

Post-conditions: The picture is scaled to the new size in the cache.

Called by: ClientApp.

Calls: scale()

References back to Requirements Document: Use Case "Beskära bild" p. 35

Parameters:

picture - the filepath of the picture to be cropped.

xUpleft - xvalue at the top left corner.

yUpleft - yvalue at the top left corner.

xDownright - xvalue at the bottom left corner.

yDownright - yvalue at the top bottom corner.

Returns: true if operation succeeded.

Class Photographer

FIELDS

String name - The photographers name

int standardLicense - The photographer has a standard license agreement which defines his copyright preferences.

CONSTRUCTOR

```
public Photographer(java.lang.String name, int standardLicense)
```

Creates a new object with the photographers name and standard license agreement.

Called by: Picture(), ClientApp

References back to Requirements Document: Use Case "SÄTTA ETT AVTAL FÖR ANVÄNDNING AV EN BILD" p.35

Parameters:

name - - The name of the photographer

standardLicense - - the photographers standard license agreement

METHODS

getName

```
public java.lang.String getName()
```

Called by: Picture()

Returns:

the name

setName

```
public void setName(java.lang.String name)
```

Called by: Picture(), ClientApp()

Parameters:

name - the name to set

getStandardLicense

```
public int getStandardLicense()
```

Called by: Picture()

Returns:

the standardLicense

setStandardLicense

```
public void setStandardLicense(int standardLicense)
```

Called by: Picture(), ClientApp

Parameters:

standardLicense - the standardLicense to set

addToDB

public void addToDB()

Adds this photographer to the database.

Called by: ClientApp()

Calls: ClientsideCom.addPhotographer(this)

Class User

FIELDS

String username - The user's username;

String password - The user's password.

String latestSearch - The user's latest search string.

String latestFolder - The user's latest opened folder in the filetree.

String latestAction - The user's latest action.

String latestAlbum - The user's latest opened album in the albumtree.

String workdirectory - The user's work directory.

CONSTRUCTOR

User(String userName)

Creates a new object with the user's username

Called by: GUILogin

Calls: super

User(java.lang.String name, int standardLicence, java.lang.String workdirectory)

Creates a new object with the user's username, standard license agreement and workdirectory.

User(java.lang.String name, java.lang.String userName, int standardLicense)

METHODS

login

public boolean login(java.lang.String password)

Checks if the user is a legal user according to the database.

Converts the password to an md5-checksum.

Calls: ClientsideCom.login()

Called by: GUILogin

References back to Requirements Document: Use Case "Användaridentifiering" p.40

Parameters:

password -

Returns: true if login succeeded.

getLatestAction

```
public java.lang.String getLatestAction()
```

Returns:

the latestAction

setLatestAction

```
public void setLatestAction(java.lang.String latestAction)
```

Parameters:

latestAction - the latestAction to set

getLatestAlbum

```
public java.lang.String getLatestAlbum()
```

Returns: the latestAlbum

setLatestAlbum

```
public void setLatestAlbum(java.lang.String latestAlbum)
```

Parameters:

latestAlbum - the latestAlbum to set

getLatestFolder

```
public java.lang.String getLatestFolder()
```

Returns:

the latestFolder

setLatestFolder

```
public void setLatestFolder(java.lang.String latestFolder)
```

Parameters:

latestFolder - the latestFolder to set

getLatestSearch

```
public java.lang.String getLatestSearch()
```

Returns:

the latestSearch

setLatestSearch

```
public void setLatestSearch(java.lang.String latestSearch)
```

Parameters:

latestSearch - the latestSearch to set

getUserName

```
public java.lang.String getUserName()
```

Returns:

the userName

setUserName

```
public void setUserName(java.lang.String userName)
```

Parameters:

userName - the userName to set

getWorkdirectory

```
public java.lang.String getWorkdirectory()
```

Returns:

the workdirectory

setWorkdirectory

```
public void setWorkdirectory(java.lang.String workdirectory)
```

Parameters:

workdirectory - the workdirectory to set

Class NetworkObject

FIELDS

String typeOfCommand - There are different kinds of commands. These are import picture, write to file etc.

The commands will be interpreted in - ServersideCom.parseNetworkObject().

int id - Every network object has its unique id to be able to determine who asked for the result.

Object - contentsthe contents of the object.

CONSTRUCTOR

```
public NetworkObject(java.lang.String typeOfCommand,  
                    int id,  
                    java.lang.Object contents)
```

Creates a new network object to facilitate the communication over the network.

Parameters:

typeOfCommand - Defines the type of NetworkObject.

id - The unique id of the NetworkObject.

contents - The contents of the NetworkObject.

METHODS

getContents

public java.lang.Object getContents()

Returns:

the contents of the command.

getId

public int getId()Returns:

the id

getTypeOfCommand

public java.lang.String getTypeOfCommand()

Returns:

the typeOfCommand

Class WebTemplate

FIELDS

Color backgroundColor

Color borderColor

Color textColor

int pictureRows - Determines the number of rows

int pictureColumns - Determines the number of columns

CONSTRUCTOR

public WebTemplate()

Constructs a new template with the standard settings.

Called by: Webpage

METHODS

setColors

public void setColors(java.awt.Color background, java.awt.Color border, java.awt.Color text)

Sets the colors of the webpage.

setNumberOfThumbnails

public void setNumberOfThumbnails(int rows, int columns)

Sets the number of rows and columns of the thumbnails in the output webpage.

Parameters: rows, columns – The number of thumbnails displayed on each page.

getPictureColumns

public int getPictureColumns()

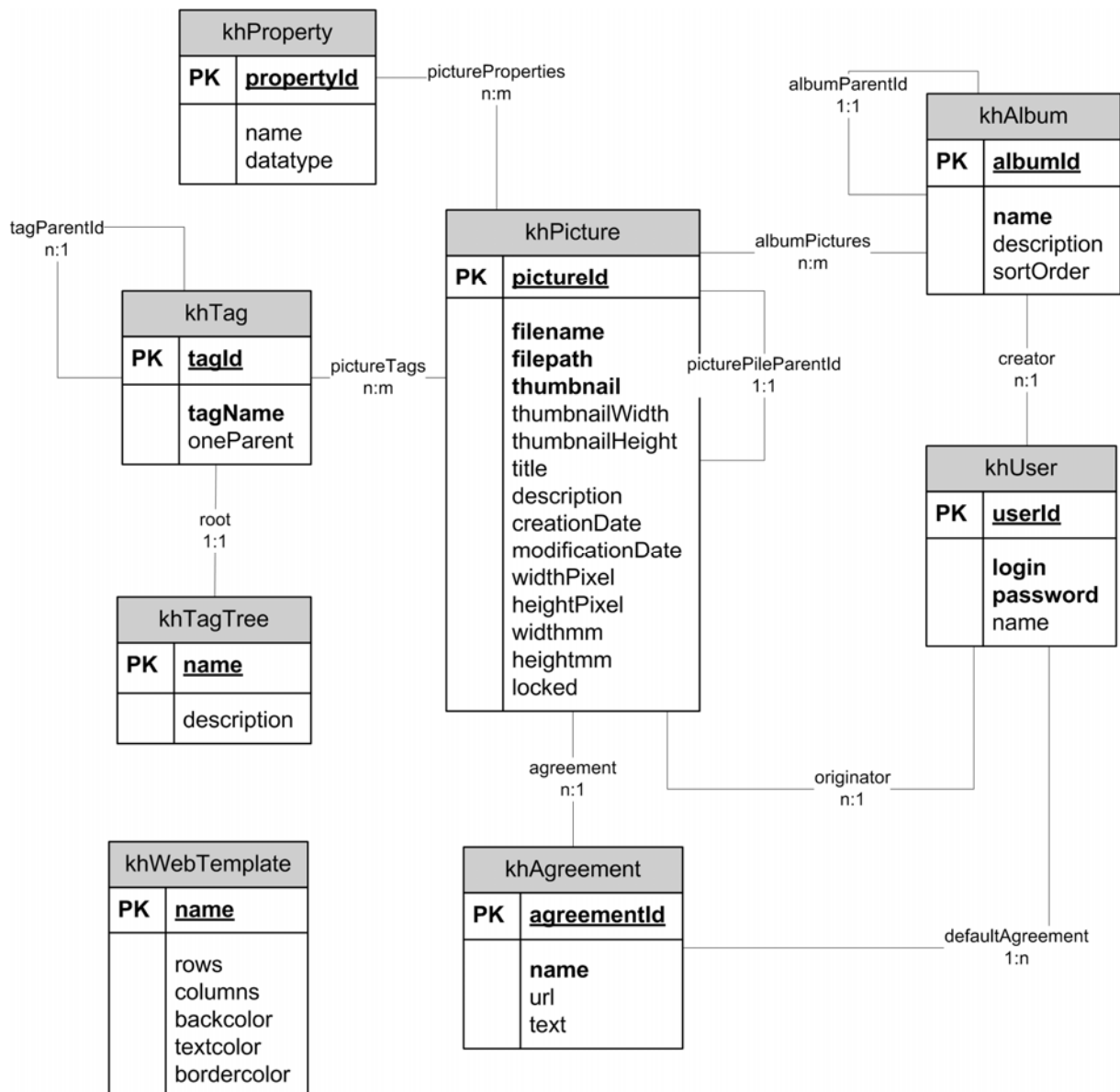
Returns: the pictureColumns

getPictureRows

public int getPictureRows()

Returns: the pictureRows

Database model



T-MATRIX:

Type	Name	I-term	E-term
Obj	khPicture	pictureId	filename, filepath, thumbnail, thumbnailWidth, thumbnailHeight, title, description, creationDate, modificationDate, widthPixel, heightPixel, widthmm, heightmm, locked
	khAlbum	albumId	name, description, sortOrder
	khUser	userId	login, password, name
	khAgreement	agreementId	name, url, text
	khTag	tagId	tagName, oneParent
	khTagTree	name	description
	khProperty	propertyId	name, datatype
	khWebTemplate	name	rows, columns, bgcolor, textcolor, bordercolor
Rel	albumPictures	albumId, pictureId	customOrder, description
	Creator	albumId, userId	private
	pictureProperties	pictureId, propertyId	value
	pictureTags	pictureId, tagId	
	Agreement	pictureId, agreementId	
	Originator	pictureId, userId	
	defaultAgreement	agreementId, userId	
	root	tagId, name	
	tagParentId	tagId, tagId	
	albumParentId	albumId, albumId	
	picturePileParentId	pictureId, pictureId	

LOGICAL DATABASE MODEL:

khPicture	((pictureId), filename, filepath, thumbnail, title, description, creationDate, modificationDate, widthPixel, heightPixel, widthmm, heightmm, locked, agreementId, userId, pictureId)
khAlbum	((albumId), name, description, sortOrder, userId, private, albumId)
khUser	((userId), login, password, name)
khAgreement	((agreementId), name, url, text)
khTag	((tagId), tagName, oneParent, tagId)
khTagTree	((name), description, tagId)
khProperty	((propertyId), name, datatype)
khWebTemplate	((name), rows, columns, bgcolor, textcolor, bordercolor)
albumPictures	((albumId, pictureId), customOrder, description)
pictureProperties	((pictureId, propertyId), value)

albumTags	((albumId, tagId))
pictureTags	((pictureId, tagId))

Requirements document cross references

Here we list, for each use case, some of the methods it will call during a run. Since the GUI is not included in this part of the design document, most references are to the server side.

Class ServerApp

main

Class ServerLogFile

writeLog

newFile

Class ServerSettings

readSettings

saveSettings

Class DBConnection

initDB

searchDB

USE CASE: "AVANCERAD SÖKNING" p.29

USE CASE: "SÖKNING MED HJÄLP AV TAGGAR" p.29

userLogin

USE CASE: "STARTA KEYHOLE" p.40

newPicture

USE CASE: "INLÄSNING AV BILDER" p.21

USE CASE: "ÄNDRA BILDINFORMATION" p.36

USE CASE: "TILLDELNING AV TAGG FRÅN PRIMÄRA TAGGTRÄDET"

p.25

USE CASE: "VÄLJ TAGGAR FRÅN ANDRA TAGGTRÄD" P.26

newPicturePile

"ATT ARBETA MED FOTOHÖGAR" p.15

USE CASE: "SKAPA EN FOTOHÖG" p.32

USE CASE: "LÄGG TILL EN BILD I EN FOTOHÖG" p.33

USE CASE: "BYT HUVUDBILD I EN FOTOHÖG" p.33

USE CASE: "TA UR EN BILD UR EN FOTOHÖG" p.34

newAlbum

"ATT ARBETA MED ALBUM" p.14

USE CASE: "SKAPA ETT ALBUM" p.23

USE CASE: "SPARA EN SÖKNING SOM ETT ALBUM" p.23
USE CASE: "ÄNDRA EGENSKAPER FÖR ETT ALBUM" p.24
USE CASE: "ÄNDRA SORTERINGEN AV BILDERNA I ETT ALBUM" p.24
USE CASE: "TA BORT BILDER UR ETT ALBUM" p.24

deleteObject

USE CASE: "RADERA EN BILD" p.22

USE CASE: "RADERA ETT ALBUM" p.25

USE CASE: "TA BORT EN TAGG UR TAGGTRÄDET" p.28

USE CASE: "DELA UPP EN FOTOHÖG" p.34

newTag

USE CASE: "LÄGGA TILL EN NY TAGG" p.27

newTagTree

newUser

USE CASE: "SKAPA EN NY ANVÄNDARE" p.31

newPhotographer

newAlbumTree

lock

isLocked

Class FileManager

getFile

getThumbnail

storeFile

cacheFile

cacheThumbnail

Class SearchForm

SearchForm

Class Picture

addTag

getMetadata

getPhotographer

getTags

hasTag

removeTag

Class PicturePile

getMainPictureId

setMainPictureId

addPicture
removePicture

Class PictureSet

add
getId
getSortOrder
remove
setSortOrder

Class TagNode

setName
getName
setUniqueAncestor
addNode
existChild
getChildren
rename
getUniqueAncestor
askName

Class TagTree

Class Album

renumber
getDescription
setDescription
getName
setName
addPicture
removePicture
getPictureSet

Class AlbumPicture

addPicture
addPicture

Class AlbumTree

Class ClientApp

connect

USE CASE: "STARTA KEYHOLE" p.40

run

setupGUI

USE CASE: "STARTA KEYHOLE" p.40

show

setLogin

login

USE CASE: "STARTA KEYHOLE" p. 40

lockPictures

"USE CASE: FLERA ANVÄNDARE MODIFIERAR SAMMA BILD"

updateInfo

import

"USE CASE: INLÄSNING AV BILDER TILL KEYHOLE" p.21

main

Class ClientSettings

saveSettings

getServerLocation

USE CASE: "STARTA KEYHOLE" p.40

setServerLocation

Class Manipulator

rotate

USE CASE: "ROTERA BILD" p. 36

scale

USE CASE: "SKALA BILD" p. 35

crop

USE CASE: "BESKÄRA BILD" p. 35

Class Photographer

Photographer

USE CASE: "SÄTTA ETT AVTAL FÖR ANVÄNDNING AV EN BILD" p.35

getName

setName

getStandardLicense

setStandardLicense

addToDB

Class User

login

USE CASE: "ANVÄNDARIDENTIFIERING" p.40

getLatestAction

setLatestAction

getLatestAlbum

setLatestAlbum

getLatestFolder

setLatestFolder

getLatestSearch

setLatestSearch

getUserName

setUserName

getWorkdirectory

setWorkdirectory

Class ClientsideCom

parseNetworkObject

sendNetworkObject

Class NetworkObject

getContents

getId

getTypeOfCommand

Class ServersideCom

parseNetworkObject

sendNetworkObject

run

readFile

writeFile

USE CASE: "INLÄSNING AV BILDER" p.21

USE CASE: "FLYTТА EN BILD" p.42

Class WebPage

getWebTemplate

USE CASE: "SKAPA WEBBSIDA" p. 37

setWebTemplate

USE CASE: "SKAPA WEBBSIDA" p. 37

writePicturesToFiles

USE CASE: "SKAPA WEBBSIDA" p. 37

createPages

Class WebTemplate

setColors

setNumberOfThumbnails

getPictureColumns

getPictureRow

5.6 Package Diagram

All GUI-classes, FileManager, ClientSettings and ClientsideCom are exclusive client classes.

ServerSettings, DBConnection, ServersideCom are exclusive server classes.

The rest are used by both applications.

Package khServerApplication

ServerApp

ServersideCom

ServerSettings

DBConnection

Package khClientApplication

ClientApp

ClientsideCom

ClientSettings

FileManager

GUI

GUITree

GUIPictureView

GUIMenu

GUISearch

GUIPictinfo

GUILogin

GUIUserinfo

Package khCommon

NetworkObject

User

Photographer

Picture

PicturePile

PictureSet

Manipulator

Tag

TagTree

Album

AlbumTree

6. Functional Test Cases

Import

Function being tested:

Import of pictures without entering information.

Use case:

“Inläsning av bilder till Keyhole” p.21

Initial system state:

The KH-Client has successfully connected to the KH-Server.

Input:

One or more pictures from the client file system.

Expected output:

The pictures have been uploaded to the server and placed in the directory “<KH-root>/photographer/unknown/year/month/day” at the server file system (where year/month/day is the import date). Information about the pictures has been stored in the database. The KH-Client has opened a file view (“filvy”) of the imported pictures.

Instructions:

1. Choose “import pictures” from the main menu.
2. Choose the pictures to be imported from the local file system.
3. Verify that the pictures have been imported.

Delete a picture

Function being tested:

Editor (“bildredaktören”) or the photographer successfully deleting a picture from the system.

Use case:

“Radera en bild” p.22

Initial system state:

There is at least one picture in the system of which the current user is the photographer (or the current user is an editor with full privileges to delete the picture).

Input:

A picture in the system.

Expected output:

The chosen picture and any references to it have been removed from the database. The current client view has been updated and does not show the deleted picture.

Instructions:

1. Choose a picture.
2. Choose delete the picture.
3. Answer yes to confirm the delete.
4. Verify that the picture is no longer in the system.

Delete a picture – insufficient rights

Function being tested:

Failure to delete a picture from the system due to missing privileges.

Use case:

“Radera en bild” p.22

Initial system state:

There is at least one picture in the system. The picture being deleted has not been photographed by the current user or the current user is not an editor.

Input:

A picture in the system.

Expected output:

The client display a dialog explaining that the picture could not be deleted. The picture and any relating information has not been altered at the server.

Instructions:

1. Choose a picture.
2. Choose delete the picture.
3. Answer yes to confirm the delete.
4. Verify that the picture still is in the system.

Delete a picture – locked by another user

Function being tested:

Failure to delete, move or modify a picture from the system due to picture being locked by another client.

Use case:

“Radera en bild” p.22

”Organisera om bilder”, p.31

”Skala bild”, p.34

”Beskära bild förlustfritt”, p.34

”Roter bild”, p.35

“Flera användare modifierar samma bild”, p.42

Initial system state:

There is at least one picture in the system. The picture being deleted, moved or modified is currently locked by another client.

Input:

A picture in the system.

Expected output:

The client displays a dialog explaining that the operation could not be performed. The picture or any relating information has not been altered at the server.

Instructions:

1. Choose a picture.
2. Choose delete, move or modify the picture.
3. Answer yes to confirm the operation.

4. Click ok in the dialog explaining that the operation could not be performed.
5. Verify that the picture has not been deleted, moved or modified.

Create an album

Function being tested:

Creating a new empty album.

Use case:

“Skapa ett nytt album” p.22

Initial system state:

Input:

Album name (and optionally a description).

Expected output:

An empty album and its location in the album tree has been stored in the database. The album tree view has been updated at the client.

Instructions:

1. Right click on an album in the album tree view at the level where the new album should be placed.
2. Choose “New Album”.
3. Enter album name and optionally a description.
4. Verify that the album has been created at the correct location in the album tree.

Create an album on a search result set

Function being tested:

Creating a new album from a search result.

Use case:

“Spara en sökning som ett album” p.23

Initial system state:

Input:

Album name (and optionally a description).

Expected output:

An album containing the chosen pictures and its location in the album tree has been stored in the database. The album tree view has been updated at the client.

Instructions:

1. Right click on an album in the album tree view at the level where the new album should be placed.
2. Choose “New Album”.
3. Enter album name and optionally a description.
4. Verify that the album has been created at the correct location in the album tree containing the chosen pictures.

Change album properties

Function being tested:

Successfully changing album properties.

Use case:

“Ändra egenskaper för album” p.23

Initial system state:

There is an album available.

Input:

Valid album name (non-empty and no other album with the same name at the same level in the tree) and description.

Expected output:

The album properties have changed in the database and the client has updated the album tree view to show the new name.

Instructions:

1. Right click on an album in the album tree view.
2. Choose “Change Album Properties”
3. Change the album name and description in the form.
4. Confirm the changes.
5. Verify that the album name has changed and that the new name is visible in the album tree view.

Change album name

Function being tested:

Failure to change album name to same name of other album at the current place in the album tree.

Use case:

“Ändra egenskaper för album” p.23

Initial system state:

There is more than one album available at the current place in the album tree.

Input:

Album name.

Expected output:

The client displays a dialog explaining that the album could not be renamed. The album information has not been altered at the server.

Instructions:

1. Right click on an album in the album tree view.
2. Choose “Change Album Properties”
3. Change the album name to same name as another album at the current place in the album tree.
4. Confirm the changes.
5. Click ok in the dialog explaining that the operation could not be performed.
6. Verify that the album name has not changed and the album is still available in the album tree view.

Changing album name to invalid name

Function being tested:

Changing album name to invalid name.

Use case:

“Ändra egenskaper för album” p.23

Initial system state:

There is an album available.

Input:

None.

Expected output:

The client displays a dialog explaining that the album could not be renamed. The album information has not been altered at the server.

Instructions:

1. Right click on an album in the album tree view.
2. Choose "Change Album Properties"
3. Remove the album name.
4. Confirm the changes.
5. Click ok in the dialog explaining that the operation could not be performed.
6. Verify that the album name has not changed and the album is still available in the album tree view.

Changing sort order for an album

Function being tested:

Changing sort order for an album.

Use case:

"Ändra sorteringen av bilderna i ett album" p.24

Initial system state:

There is an album available with pictures.

Input:

None.

Expected output:

The client has updated the view of the album to show the pictures in the new sort order. The new sort order for the album has been stored in the database.

Instructions:

1. Open the album.
2. Choose (one at a time) "Sortera efter titel" (sort by title), "Sortera efter filnamn" (sort by filename), "Sortera efter skapelsedatum" (sort by creationtime), "Sortera på fotograf" (sort by photographer), "Egen ordning" (user defined order)
3. Verify that the pictures are being displayed in the new sort order.
4. Repeat until all sortings have been tested.
5. Choose "Egen ordning" and try to reorder the miniatures by drag and drop.
6. Choose some other way to order the pictures.
7. Choose "Egen ordning" again.
8. Verify that the latest custom order now is applied.

Removing pictures from an album

Function being tested:

Removing pictures from an album.

Use case:

“Ta bort bilder ur ett album” p.25

Initial system state:

There is an album available with pictures.

Input:

None.

Expected output:

The pictures have been removed from the album in the database and the client has updated the view of the album without the removed pictures. The pictures have only been removed from the album, not from the system.

Instructions:

1. Open the album.
2. Choose one or more pictures and choose “Remove from album”.
3. Confirm the action.
4. Verify that the pictures are no longer in the album.

Deleting an album

Function being tested:

Deleting an album.

Use case:

“Radera ett album” p.24

Initial system state:

There is an album available.

Input:

None.

Expected output:

The client has updated the album tree view and does no longer show the removed album. The server has removed the album from the database.

Instructions:

1. Right click on the album.
2. Choose “Delete album”.
3. If the album contained pictures answer yes to confirm the removal of the album.
4. Verify that the album is no longer in the album tree.
5. Verify that all pictures still exists in the file system.

Assigning a tag to a picture

Function being tested:

Adding a tag from the tag tree to a picture

Use case:

“Tilldelning av tagg från det primära taggträdet” p.25

Initial system state:

One or many pictures have been imported to the system and are selected by the user.

Input:

A tag and one or many pictures.

Expected output:

The pictures have been updated with the tag information in the database. The tag has been added to the pictures information form on the client side.

Instructions:

1. Choose "Ändra bildinformation"
2. Choose a tag in the current tag tree.
3. Choose "Välj"
4. Verify that the tag has been added to the pictures information form.

Adding a new tag to the tag tree

Function being tested:

Adding a new tag to the tag tree.

Use case:

"Lägga till en ny tagg" p.27

Initial system state:

The KH-Client has successfully connected to the KH-Server. There is a tag tree available.

Input:

A tag name that does not exist on the selected level of the tree.

Expected output:

The tag has been added to the tag tree on the chosen level in the tree. A new tagnode has been created and the database has been updated with the tag.

Instructions:

1. Right click on a tag in the current tag tree.
2. Choose "Ny tagg"
3. Enter the new tag name
4. Confirm
5. Verify that the tag is in the tree.

Adding a new tag to the tag tree – expected failure

Function being tested:

Failure to adding a new tag to the tag tree that already exists on that level.

Use case:

"Lägga till en ny tagg" p.27

Initial system state:

The KH-Client has successfully connected to the KH-Server. There is a tag tree available.

Input:

A tag name that already exists on the selected level of the tree.

Expected output:

The tag has not been added to the tag tree on the chosen level in the tree. A dialog is shown requesting the user to change name of the tag.

Instructions:

1. Right click on a tag in the current tag tree.
2. Choose "Ny tagg"
3. Enter the new tag name
4. Confirm
5. Verify that the tag is not in the tree.

Removing a tag from the tag tree

Function being tested:

Removing a tag from the tag tree.

Use case:

"Ta bort en tagg i taggträdet" p.29

Initial system state:

The KH-Client has successfully connected to the KH-Server. There is a tag tree available with at least one tag.

Input:

The tag to remove.

Expected output:

The tag has been removed from the tag tree. All tags on the levels under the removed tag moves up one step. All affected tags have been updated in the database with its new parent. Every picture marked with the tag is still marked with the tag but now as a text string.

Instructions:

1. Right click on a tag in the current tag tree.
2. Choose "Ta bort tagg"
3. Confirm

Searching for pictures using the tag tree

Function being tested:

Searching for pictures using the tag tree.

Reference to RD:

"Att söka bilder med hjälp av taggar", p 19, and "Att söka bilder i Keyhole" p.30

Initial system state:

There must be some pictures in the system marked with one or more tags. There must be some tags in more than one tag tree.

Input:

The tags to use in the search from the tag trees.

Expected output:

All pictures marked with the tag (one tag search) or that fits an expression will be shown as a result of the search. A bad formed expression should result in an error message. It should be possible to edit the expression and try again.

Instructions:

1. Left click on a tag in the current tag tree.
2. Verify that the tag is added to the search expression.
3. Search and verify the expected result.
4. Keep the former search expression. Choose another tag from the same tree and left click.
5. Verify that the new tag is added with an “ELLER” (OR) conjunction.
6. Search and check the tags of the pictures in the resulting set.
7. Keep the former search expression. Choose another tag tree and choose a tag in the new tree, left click.
8. Verify that the tag is added to the search expression with an “OCH” (AND) conjunction.
9. And verify that the first two have been grouped with a pair of parenthesis.
10. Search and verify the expected result.
11. Keep the former search expression. Try to edit it. You could change “OCH” to “ELLER” or move the parenthesis.
12. Try a bad formed expression.
13. Verify the error message.
14. Vary the expressions and try several times.

Searching for pictures using the search form

Function being tested:

Searching for pictures using the search form.

Reference to RD:

”Avancerad sökning”, p 19 and “Att söka bilder i Keyhole” p.29

Initial system state:

There must be some pictures in the system. The pictures must have been assigned attributes that is about to be tested. There must also exist picture which has not the attributes set.

Input:

Search expressions for applicable fields: file name, title, photographer, creation date, modification date

Expected output:

All pictures marked with the search criteria will be shown as miniatures as a result of the search.

Instructions:

1. Choose the form for “Sökning”.
2. Enter a reasonable value in one text field.
3. Execute the search. Verify the result set.
4. Enter an unreasonable value.
5. Execute the search. Verify that entering illegal values result in error messages. Verify that entering a string value and then delete the letters, thus creating an empty string, is treated as if the field has not been used not as if you search for pictures with this field empty.
6. Repeat with some different reasonable and unreasonable values for each possible search field.

7. Then try combined search with values in two field.
8. Try to combine search for a date or title combined with a tag expression.
9. Be attentive to what is not found. Consider the possibility there could be hits that should show up but that fail for some reason.
10. Note queries that seem to take a long time. Is long execution time correlated to complicated expressions or to big result sets?

Big result set

Function being tested:

Behaviour on very big result sets.

Reference to RD:

“Frågestrategi vid stora svarsmängder”, RD p 44

Initial system state:

There must exist a reasonably big number of pictures – say at least 500. They should be selectable by some search query. It could be all pictures created by one and the same photographer.

Input:

Using the search form – search for all pictures created by one and the same photographer.

Expected result:

The complete result set should be browsable within a reasonable time.

Instructions:

1. All the pictures should quickly, within few seconds, be represented by name and an empty miniature image.
2. Verify that you can browse back and forth in the set.
3. All the pictures should finally, within one or two minutes, be represented with miniature.
4. Verify that you can browse back and forth in the set.

Creating a new user

Function being tested:

Creating a new user

Use Case:

“Skapa en ny användare” p.32

Initial system state:

The KH-Client has successfully connected to the KH-Server. The user must be an administrator.

Input:

Unique values for username and password. Privileges.

Expected output:

A new user has been added to the database. A dialog is shown to the user verifying that the new user has been added.

Instructions:

1. Choose “Skapa ny användare”
2. Enter a unique username, password and privileges.

3. Confirm the existence of the new user.
4. Logg out and try to logg in as the new user.
5. Test a function that the new user should not be allowed to use (try to create a new user). Verify the expected error message. Notice if an action is erroneously allowed.

Creating a new user – expected failure

Function being tested:

Failure to create a new user due to the username already being used.

Use Case:

“Skapa en ny användare” p.32

Initial system state:

The KH-Client has successfully connected to the KH-Server. The user must be an administrator.

Input:

Not unique value for username. Password. Privileges.

Expected output:

No new user has been added to the database. The system shows a dialog requesting the user to enter a new username.

Instructions:

1. Choose “Skapa ny användare”
2. Enter a not unique username, a password and privileges.
3. Confirm
4. Verify the expected error message.

Create a picture pile

Function being tested:

Create a picture pile

Use Case:

“Skapa en fotohög” p.32

Initial system state:

There must be at least two pictures in the system.

Input:

Two pictures.

Expected output:

The chosen pictures will be joined together in a picture pile. Only the main picture will be shown directly. The database will be updated with information of what picture pile contains which pictures.

Instructions:

1. Select a set of pictures to be included in the picture pile. The pictures may be single pictures, a whole album or another picture pile.
2. Right click and choose “Skapa fotohög”.
3. Choose which of the pictures that will be the main picture for this picture pile.
4. Choose not to give the main pictures information to the rest of the pictures.

Add a picture to a picture pile

Function being tested:

Add a picture to a picture pile.

Use Case:

“Lägg till en bild i en fotohög” p.33

Initial system state:

There must exist a picture pile and a picture that is not part of the picture pile.

Input:

Picture, picture pile

Expected output:

The chosen picture will be joined together to the picture pile. It will no longer be shown on the screen. Only the main picture will be shown from the picture pile. The database will be updated with information that the picture is part of the picture pile.

Instructions:

1. Select a picture to be included in the picture pile.
2. Right click and choose add to picture pile.
3. Browse your way to the picture pile you want to add the picture to.
4. Choose whether the picture will get the main picture's information or not.
5. Confirm

Assigning a new main picture of a picture pile

Function being tested:

Change the main picture of a picture pile.

Use Case:

“Byt huvudbild i en fotohög” p.33

Initial system state:

There must exist a picture pile in the system containing two or more pictures.

Input:

Picture pile, picture

Expected output:

The chosen picture will be the new main picture of the picture pile. The new main picture will be shown on the screen. The database will be updated with information of what picture is the main picture of the picture pile.

Instructions:

1. Select the picture pile.
2. Choose “Öppna”.
3. Verify that the pictures in the picture pile are shown.
4. Right click on the picture you want to have as the new main picture.
5. Choose “Gör till huvudbild” (assign as main picture) from the pop up menu.

Inherit properties from the main picture of a picture pile

Function being tested:

Get the main picture of a picture piles information copied to another of the pictures in the picture pile.

Use Case:

“Lägg till en bild i en fotohög” p.33 and ”Byt huvudbild i en fotohög” p.33

Initial system state:

There must exist a picture pile in the system with different information in the main picture and another picture.

Input:

Picture, picture pile

Expected output:

The picture in the picture pile will have the main pictures information. The database will copy the information from the main picture to the selected picture.

Instructions:

1. Select a picture pile.
2. Select one or many pictures that are not the main picture.
3. Choose “Tilldela huvudbildens information”.
4. Confirm
5. Verify by selecting the altered pictures, that it has new information.

Remove a picture from a picture pile

Function being tested:

Remove a picture from a picture pile.

Use Case:

“Ta ur en bild ur en fotohög” p.34

Initial system state:

There must exist a picture pile in the system. It must have more than one picture.

Input:

Picture, picture pile

Expected output:

The selected picture will no longer be a part of the picture pile. The picture will be removed from the picture pile in the database.

Instructions:

1. Select a picture pile.
2. Right click on a picture that is not the main picture.
3. Choose “Ta bort från fotohög”.
4. Confirm
5. Verify that the picture is removed from the picture pile by opening the picture pile.

Dissolve a picture pile

Function being tested:

Dissolve a picture pile.

Use Case:

“Dela upp en fotohög” p.34

Initial system state:

There must exist a picture pile in the system. It must have more than one picture.

Input:

Picture pile

Expected output:

The pictures in the picture pile will no longer be a part of the picture pile. The pictures inside the picture pile will be shown as miniatures. All the pictures that were collected in the picture pile will now appear as free pictures in their own right.

Instructions:

1. Select a picture pile.
2. Right click on the main picture.
3. Choose “Dela upp fotohög” (dissolve the picture pile).
4. Confirm
5. Verify that the picture pile has been dissolved by checking that all the pictures inside the picture pile are shown as miniatures.

Scaling a picture - percent

Function being tested:

Scaling a picture using the scale factor percent.

Use Case:

“Skala bild” p.35

Initial system state:

There must exist a picture in the system.

Input:

Which picture to scale and the scaling factor in percent.

Expected output:

The picture will be scaled on the server file system and shown scaled to the client.

Instructions:

1. Open a picture.
2. Choose “skala” (scale).
3. Enter the scale factor.
4. Confirm
5. Verify that the picture has been scaled as much as you chose. Estimate the quality of the scale compared to an advanced picture processing tool as Photoshop.
6. Verify that the picture is marked as changed and not saved.
7. Choose “spara” (save), and verify that the mark “changed” disappears.

Scaling a picture - pixel

Function being tested:

Scaling a picture using the scale factor number of pixels.

Use Case:

“Skala bild” p.35

Initial system state:

There must exist a picture in the system.

Input:

Which picture to scale and the new size of the picture.

Expected output:

The picture will be scaled on the client file system and shown scaled to the client.

Instructions:

1. Open a picture.
2. Choose “skala”.
3. Enter the scale factor.
4. Confirm
5. Verify that the picture has been scaled as much as you chose. Estimate the quality of the scale compared to an advanced picture processing tool as Photoshop.
6. Do not save the picture. Select another folder and go back to the first folder.
7. Verify that the file still is marked as changed but not saved.
8. Leave the folder.
9. Open another client. Verify that the picture that was scaled by the first client is locked.
10. Shut down the first client.
11. Try again to open the scaled picture. It should now appear unlocked and in its original size.

Automatic scaling

Function being tested:

Automatic scaling.

Use Case:

“Skala bild” p.35

Initial system state:

There must exist a picture that is larger than the screen in its original size in the system.

Input:

A picture that is larger than the screen in its original size.

Expected output:

The picture will be shown scaled to the client. Nothing will happen to the picture on the server file system.

Instructions:

1. Open a picture.
2. Verify that the picture has been scaled to fit your screen. Estimate the quality of the scale compared to an advanced picture processing tool as Photoshop.

Crop a picture

Function being tested:

Crop a picture.

Use Case:

“Beskära bild förlustfritt” p.35

Initial system state:

There must exist a picture in the system.

Input:

A picture to crop.

Expected output:

The picture will be cropped on the server file system and shown cropped to the client.

Instructions:

1. Open a picture.
2. Choose “Beskär” (crop)
3. Mark an area of the picture using the mouse.
4. Choose “Ok”
5. Verify that the picture has been cropped by looking at it.
6. Verify that the picture is marked as changed but not saved.
7. Choose “Spara”
8. Verify that the picture has been cropped on the server file system.
9. Try another picture. Now choose “spara som” (save as).
10. Verify that the original has not been changed and isn't marked as changed, and that there is a new picture added to the system.

Rotate a picture

Function being tested:

Rotate a picture.

Use Case:

“Roter bild” p.36

Initial system state:

There must exist a picture in the system.

Input:

A picture to rotate.

Expected output:

The picture will be rotated on the server file system and shown rotated to the client.

Instructions:

1. Open a picture.
2. Click to rotate (left/right).
3. Verify that the picture has been rotated by inspection.

Change picture information

Function being tested:

Change picture information.

Use Case:

“Ändra bildinformation” p.36

Initial system state:

There is at least one picture in the system. Pictures are shown as miniatures in the form PictureBox.

Input:

One or many pictures. Picture information to change.

Expected output:

The picture information will be updated in the database.

Instructions:

1. Select one or many pictures.
2. Choose “Ändra bildinformation”.
3. Verify that the form PicInfo is editable.
4. Enter the picture information you want to change.
5. Choose “Uppdatera information”.
6. Verify that the picture information has been updated.

Change picture information - lock failure

Function being tested:

Failure to change picture information due to pictures being locked.

Use Case:

“Ändra bildinformation” p.36 and “Flera användare modifierar samma bild”, p 43

Initial system state:

There is at least one picture in the system. Pictures are shown as miniatures in the form PictureBox. The pictures have been locked by another client. At least two clients must be running.

Input:

One or many pictures. Picture information to change.

Expected output:

A message box is shown telling the user to wait because the pictures are locked.

Instructions:

1. Select one or many pictures that have been locked by another client.
2. Choose “Ändra bildinformation”.
3. Verify that the form PicInfo is not editable.

Change copyright agreement – user rights failure

Functions being tested:

Change copyright agreement.

Failure to change copyright agreement due to not having privileges.

Use Case:

“Sätta ett avtal för användning av en bild” p.37

Initial system state:

There is at least two pictures in the system. Pictures are shown as miniatures in the form PictureBox. One of the pictures has been created by the current user and the other by different photographer. The user is not an administrator.

Input:

One or many pictures.

Expected output:

The dropdown list used to change privileges is not editable.

Instructions:

1. Select one or many pictures which are created by the current user.
2. Choose “Ändra bildinformation”.
3. Chose another agreement from the dropdown combo box used to change copyright agreement.
4. Verify that the agreement has been changed. Leave the program and verify the change on returning to the picture after restarting the client program.
5. Verify the link to the complete agreement text.
6. Select one or many pictures which are created by another photographer.
7. Choose “Ändra bildinformation”.
8. Verify that the dropdown list used to change copyright agreement is not editable.

Create a webpage

Function being tested:

Creating a webpage.

Use case:

“Skapa webbsida” p.36

Initial system state:

There is at least one picture and a web template available.

Input:

The pictures to be on the web page. Web page template and customization options.

Expected output:

A webpage has been created using the chosen template and it has stored on the client file system together with the pictures.

Instructions:

1. Choose “Export to webpage” from the menu.
2. Choose template and other settings.
3. Choose where the webpage will be saved.
4. Choose export.
5. Verify that the generated web page is working.

Logging in

Function being tested:

Logging in

Use Case:

“Starta Keyhole” p.40

Initial system state:

Input:

Valid username and password.

Expected output:

The KH-Client has successfully connected to the KH-Server. The client starts and shows the users last known view.

Instructions:

1. Start the KeyHole client.
2. Enter a valid username and password.
3. Verify that the program starts and show the last shown view.

Incorrect username or password

Function being tested:

Failure to log in due to incorrect username or password.

Use Case:

“Starta Keyhole” p.40

Initial system state:

Input:

Invalid username or password.

Expected output:

The log in dialog is shown again telling the user to try again because the username or password was invalid.

Instructions:

1. Start the KeyHole client.
2. Enter an invalid username or password.
3. Verify that the program asks the user for username and password again.

First start up of a client

Function being tested:

Missing client settings file.

Use Case:

“Starta Keyhole” p.40

Initial system state:

There is no client settings file (for example if the client is being run for the first time). In testing purpose the file must be manually deleted before starting the KH-client.

Input:

Server address, valid username and password.

Expected output:

The client displays a dialog asking the user for the server location. If the client is successful in connecting to and log in on the server the server location is saved in the client settings file for future use. The client displays a file view of the KH root.

Instructions:

1. Start the KeyHole client.
2. A dialogue window should open.
3. Enter a valid server location in the dialogue.
4. Enter valid username and password when asked.
5. Verify that the client successfully connects to the entered server and that the server location is written to the client settings file.

Moving pictures

Function being tested:

Moving pictures (due to changed information on photographer or photography date).

Use case:

“Flytta en bild” p.42

Initial system state:

There is at least one picture available.

Input:**Expected output:**

The chosen pictures have been moved to their new location on the server file system. All references to the old file location in database has been updated.

Instructions:

1. Select one or many pictures.
2. Choose “Ändra bildinformation”.
3. Enter new photographer or photo date.
4. Choose “Uppdatera information”.
5. Verify that the pictures have changed location to reflect the new photographer or photo date.

Lost communication

Function being tested:

Lost communication between server and client.

Reference to RD:

“Kommunikationen avbruten” p.40

Initial system state:

Client successfully connected to the server but client has had no response from server for more than 30sec.

Input:**Expected output:**

Client displays a dialog message explaining that it has lost the connection to the server and asks the user if it should continue to try to connect or abort.

Instructions:

1. Shut down the KH-server.
2. Verify the behavior of the client application.
3. Restart the server.
4. Verify the behavior of the client application.
5. Disconnect network cable.
6. Verify the behavior of the client application.
7. Reconnect the network cable.
8. Verify the behavior of the client application. Does it reconnect?

minimal required hardware**Function being tested:**

Testing minimal required hardware.

Reference to RD:

”Operativsystem” p.43

“Nätverket” p.43

“Minne” p.44

”Minimikrav” Appendix I

Initial system state:

Computer with minimal required hardware.

Input:**Expected output:**

Test cases ended successfully.

Instructions:

1. Do all the test cases on minimal requirements hardware.

References

Diagramming technique references

UML , Unified Modeling Language.

1. “Unified Modeling Language (UML) Tutorial”, Kennesaw State University, CSIS 4650 - Spring 2001, by David Braun, Jeff Sivils, Alex Shapiro, Jerry Versteegh,
http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm.
2. Randy Miller, ”Practical UML™ A Hands-On Introduction for Developers”,
<http://dn.codegear.com/article/31863>

Class diagram, UML

Statechart, UML

Interaction Diagram, UML

Entity-Relation-model: A diagramming convention to describe relational databases. (Actually we have learned to draw it, as we have done here, in the course Databasteknik, here at CSC-KTH.) “This approach to modeling was first proposed in the mid-1970s by Chen (Chen, 1976); several variants have been developed since then (Codd, 1979; Hammer and McLeod, 1981; Hull and King, 1987), all with the same basic form.” (Ian Somerville, “Software Engineering”, edition 8, Addison-Wesley 2007

Other references

“Code Conventions for the Java™ Programming Language”,
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

“The HotDoc Computation Archives”, <http://www.legalcs.com/hotdocs/0134.htm>

Stefan Hetzl, “The .bmp file format”, <http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html>

Appendix I

The bitmap .bmp file format

Introduction:

The .bmp file format (sometimes also saved as .dib) is the standard for a Windows 3.0 or later DIB(device independent bitmap) file. It may use compression (though I never came across a compressed .bmp-file) and is (by itself) not capable of storing animation. However, you can animate a bitmap using different methods but you have to write the code which performs the animation. There are different ways to compress a .bmp-file, but I won't explain them here because they are so rarely used. The image data itself can either contain pointers to entries in a color table or literal RGB values (this is explained later).

Basic structure:

A .bmp file contains of the following data structures:

```
BITMAPFILEHEADER  bmfh;  
BITMAPINFOHEADER  bmih;  
RGBQUAD          aColors[];  
BYTE              aBitmapBits[];
```

bmfh contains some information about the bitmap file (about the file, not about the bitmap itself). bmih contains information about the bitmap such as size, colors,... The aColors array contains a color table. The rest is the image data, which format is specified by the bmih structure.

Exact structure:

The following tables give exact information about the data structures and also contain the settings for a bitmap with the following dimensions: size 100x100, 256 colors, no compression. The start-value is the position of the byte in the file at which the explained data element of the structure starts, the size-value contains the number of bytes used by this data element, the name-value is the name assigned to this data element by the Microsoft API documentation. Stdvalue stands for standard value. There actually is no such a thing as a standard value but this is the value Paint assigns to the data element if using the bitmap dimensions specified above (100x100x256). The meaning-column gives a short explanation of the purpose of this data element.

THE BITMAPFILEHEADER:

start	size	name	stdvalue	purpose
1	2	bfType	19778	must always be set to 'BM' to declare that this is a .bmp-file.
3	4	bfSize	??	specifies the size of the file in bytes.
7	2	bfReserved1	0	must always be set to zero.
9	2	bfReserved2	0	must always be set to zero.
11	4	bfOffBits	1078	specifies the offset from the beginning of the file to the bitmap data.

THE BITMAPINFOHEADER:

start	size	name	stdvalue	purpose
15	4	biSize	40	specifies the size of the BITMAPINFOHEADER structure, in bytes.
19	4	biWidth	100	specifies the width of the image, in pixels.
23	4	biHeight	100	specifies the height of the image, in pixels.
27	2	biPlanes	1	specifies the number of planes of the target device, must be set to zero.
29	2	biBitCount	8	specifies the number of bits per pixel.
31	4	biCompression	0	Specifies the type of compression, usually set to zero (no compression).
35	4	biSizeImage	0	specifies the size of the image data, in bytes. If there is no compression, it is valid to set this member to zero.
39	4	biXPelsPerMeter	0	specifies the the horizontal pixels per meter on the designated target device, usually set to zero.
43	4	biYPelsPerMeter	0	specifies the the vertical pixels per meter on the designated target device, usually set to zero.
47	4	biClrUsed	0	specifies the number of colors used in the bitmap, if set to zero the number of colors is calculated using the biBitCount member.
51	4	biClrImportant	0	specifies the number of color that are 'important' for the bitmap, if set to zero, all colors are important.

Note that biBitCount actually specifies the color resolution of the bitmap. The possible values are: 1 (black/white); 4 (16 colors); 8 (256 colors); 24 (16.7 million colors). The biBitCount data element also decides if there is a color table in the file and how it looks like. In 1-bit mode the color table has to contain 2 entries (usually white and black). If a bit in the image data is clear, it points to the first palette entry. If the bit is set, it points to the second. In 4-bit mode the color table must contain 16 colors. Every byte in the image data represents two pixels. The byte is split into the higher 4 bits and the lower 4 bits and each value of them points to a palette entry. There are also standard colors for 16 colors mode (16 out of Windows 20 reserved colors (without the entries 8, 9, 246, 247)). Note that you do not need to use this standard colors if the bitmap is to be displayed on a screen which support 256 colors or more, however (nearly) every 4-bit image uses this standard colors. In 8-bit mode every byte represents a pixel. The value points to an entry in the color table which contains 256 entries (for details see Palettes in Windows. In 24-bit mode three bytes represent one pixel. The first byte represents the red part, the second the green and the third the blue part. There is no need for a palette because every pixel contains a literal RGB-value, so the palette is omitted.

THE RGBQUAD ARRAY:

The following table shows a single RGBQUAD structure:

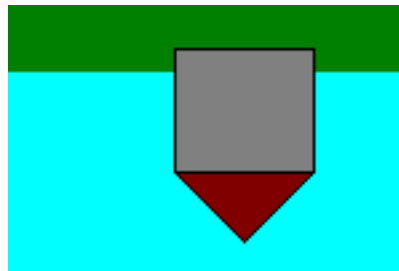
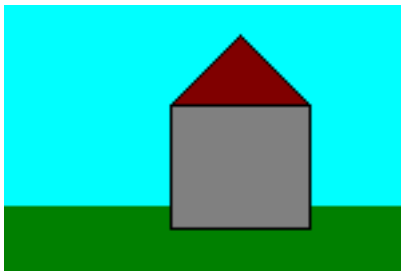
start	size	name	stdvalue	purpose
1	1	rgbBlue	-	specifies the blue part of the color.
2	1	rgbGreen	-	specifies the green part of the color.
3	1	rgbRed	-	specifies the red part of the color.
4	1	rgbReserved	-	must always be set to zero.

Note that the term palette does not refer to a RGBQUAD array, which is called color table instead. Also note that, in a color table (RGBQUAD), the specification for a color starts with the blue byte. In a palette a color always starts with the red byte. There is no simple way to map the whole color table into a LOGPALETTE structure, which you will need to display the bitmap. You will have to write a function that copies byte after byte.

THE PIXEL DATA:

It depends on the BITMAPINFOHEADER structure how the pixel data is to be interpreted (see above).

It is important to know that the rows of a DIB are stored upside down. That means that the uppermost row which appears on the screen actually is the lowest row stored in the bitmap, a short example:



Pixels displayed on screen pixels stored in .bmp-file

You do not need to turn around the rows manually. The API functions which also display the bitmap will do that for you automatically.

Another important thing is that the number of bytes in one row must always be adjusted to fit into the border of a multiple of four. You simply append zero bytes until the number of bytes in a row reaches a multiple of four, an example:

6 bytes that represent a row in the bitmap:

```
A0 37 F2 8B 31 C4
```

must be saved as:

```
A0 37 F2 8B 31 C4 00 00
```

to reach the multiple of four which is the next higher after six (eight). If you keep these few rules in mind while working with .bmp files it should be easy for you, to master it.

Copyright 1998 Stefan Hetzl. If you have questions or comments or have discovered an error, send mail to hetzl@teleweb.at. You may forward this document or publish it on your webpage as long as you don't change it and leave this notice at the end.

Stefan Hetzl "The .bmp file format", <http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html>

Index

- abstract, 9
- agreement, 106
- album**, 8
 - change properties, 91
 - create, 91
 - delete, 94
 - invalid name, 92
 - remove pictures, 93
 - sort order, 93
- Album
 - class, 40, 70
- AlbumPicture
 - class, 40, 71
- AlbumTree
 - class, 41
- audience, 7
- big result set, 98
- bitmap picture**, 8
- bmp**, 8
 - file format, 111
- caching
 - fileupload, 59
- Class diagram, 110
- Class Diagram, 42
- client, 7, 8
- client settings, 107
- ClientApp, 33
 - class, 61
- client-server architecture, 11
- ClientSettings, 33
 - class, 64
- ClientsideCom, 33
 - class, 60
 - methods, 60
- commenting conventions, 10
- communication
 - lost, 108
- conventions
 - naming and coding, 110
- copyright, 106
- crop, 104
- cross references, 82
- Database Management System, 8
- Database Model, 13
- database schema, 13
- DBConnection
 - class, 54
 - methods, 54
- DBMS, 7, 8
- Entity-Relation**, 110
- entity-relationship, 13
- entity-relationship model, 13
- event**, 8
- FileManager, 34
 - class, 59
- form**, 8
- graphical user interface, 19
- GUILogin, 35
- GUIPicInfo, 36
- GUIPictureView, 35
- GUITree, 36
- GUIUserInfoForm, 35
- import, 46
- Interaction Diagram, 45, 110
- Java doc, 10
- logging in, 107
- main picture**, 8
- manipulation
 - pictures, 16
- Manipulator
 - class, 41, 72
- message box**, 8
- metadata**, 8
- miniature**, 8
- moving pictures, 108
- naming conventions, 9

- NetworkObject
 - class, 41
- Package
 - diagram, 88
 - khClientApplication, 88
 - khCommon, 88
 - khServerApplication, 88
- password, 107
- photographer
 - changed, 108
- Photographer
 - class, 39
- photography date
 - changed, 108
- Picture
 - class, 38, 65
- picture information
 - change, 105
 - lock failure, 105
- picture pile**, 8
 - add a picture, 100
 - create, 50, 99
 - dissolve, 102
 - inherit, 101
 - new main picture, 100
- picture tree**, 8
- PicturePile
 - class, 39, 67
- PictureSet
 - class, 38, 67
- Requirements Document, 7
- Responsibility Collaborator, 33
- rotate, 104
- scaling
 - automatic, 103
 - percent, 102
 - pixel, 103
- Scope, 7
- search
 - form, 29
 - with tags, 49
- SearchForm
 - class, 65
- searching, 97
- selecting
 - pictures, 15
- server, 7
- ServerApp, 37
 - class, 53
 - methods, 53
- ServerLogfile, 38
- ServerLogFile
 - class, 53
- ServerSettings, 37
 - class, 54
 - methods, 54
- ServersideCom, 37
 - socket, 57
- socket
 - client, 60
 - server, 58
- start up
 - client, 44, 52
 - server, 43
- Statechart, 110
- tag**, 8
 - assigning, 94
 - remove from tree, 96
- tag tree**, 8
 - add new tag, 95
- TagNode, 40
 - class, 68
- tag-tree
 - add new tag, 48
- TagTree, 40
 - class, 69
- thread**, 8
- UML**, 110
- Unified Modeling Language, 110
- user

creating new, 98
User
class, 39
Web export, 11, 17
webpage
create, 30, 51, 106

form, 30
WebPage, 34
class, 63
WebTemplate, 38
version history, 7
wild card, 9