Semester Work

WS 1998/99

Personal Assistant for Internet OMS

An Intelligent Adaptive Caching Agent

Beat Signer, IIIC
March 1, 1999

Institute for Information Systems
Swiss Federal Institute of Technology (ETHZ)

Advisor:
Antonia Erni
Responsible Professor:
Prof. M. C. Norrie

Abstract

To improve the performance of internet database services for regular users, the *Internet OMS* framework was extended by the *personal assistant*, a middle layer caching agent. On the one hand, the user can explicitly define queries to be permanently cached, on the other hand, the personal assistant monitors all user actions and prefetches the data most likely to be used in the near future from the database agent into his local persistent cache. Cache consistency is ensured through cooperation between the personal assistant's caches and the database agent. The usage of the personal assistant's caching framework will increase the overall cache hit rate and therefore reduce response times, e.g. over slower network connections.

Contents

Chapter 1

Problem Formulation

SEMESTER-WORK **Beat Signer**

Personal Assistant for Internet OMS

Internet OMS is an agent based Internet interface to the object oriented database system, OMS, both developed in the group for Global Information Systems. A combination of server and client agents provides performance gains to users through active caching mechanisms.

Casual users may access the information service directly via the usual WWW services. The server agent controls all access to the database and general access is improved by the caching strategy of the server agent. All results of queries frequently requested by clients are stored in a global server cache, thereby reducing response time and database load.

The objective of this semester work is to extend the Internet OMS architecture by an additional agent - the personal assistant. The personal assistant should help regular users to reduce response times in prefetching often required data in the users local cache. Cooperation between the server and client agents ensures that only new and updated information is transferred from the server, while other information is accessed via the local cache. Prefetching of data can be based on user profiles which enable user information requirements to be downloaded into the cache in order that it is available locally when required.

The front end agent should be extended with functionalities for changing and updating the user profile, if a personal assistant is installed on the clients local machine.

The goal of the semester work is to have an extension of Internet OMS, such that a personal assistant can be downloaded over the Internet and installed on the clients local machine. Configurations and changes to the user profile should all be possible over the Internet.

The main stages of the work are:

- Learn the Internet OMS architecture
- Development of a personal assistant as an extension of the generic agent
- Extension of the front end agent with functionalities for regular users in case a personal assistant is installed on the clients local machine

Start Date:	Monday, 19th of October 1998
Environment:	Internet OMS, Java, OMS/Prolog, Sun, Unix
Supervision:	Antonia Erni, IFW D47.1

Chapter 2

Introduction

Based on the *Internet OMS* architecture [EN97, Rus97, ENK98] – an agent based internet interface for the object oriented database system *OMS* [Wue95] – the personal assistant, a new middle layer caching agent, was developed. It's main task is to reduce response times for queries requested by regular users, using different cooperative caches.

Each user can tell his preferences to the personal assistant by explicitly defining queries he wishes to be permanently cached. This is an effective way to increase the cache hit rate for users exactly knowing their preferences, i.e. knowing which data they will regularly use in the future.

Because it is an illusion to assume each user knows exactly which queries he will regularly request in the future and also is willing to investigate the necessary time to add them to the personal assistant's cache, there exists a second, adaptive caching component. The personal assistant monitors all user actions and maintains statistics about user behaviour. Based on this statistics, it autonomously will perform prefetching of the data most likely to be used in the near future. Over slower network connections, e.g. internet connections using analogue modems, the active prefetching will lead to a better utilization of the available bandwidth. After receiving an answer, a user naturally analyzes the result for a certain time. Without prefetching, no data will be transmitted during this time and the modem will be idle. Unfortunately when connecting to the internet using a modem, you have to pay for the whole online-time, including the time the modem is idle. The personal assistant will therefore use the idle time to perform active prefetching of data, resulting in a permanent workload on the modem.

The usage of this two cooperative caching mechanisms complemented by a third short time session cache should adapt to each individual users behaviour and therefore improve the overall cache hit rate. As a result of the higher cache hit rate, the mean query response times will get smaller.

The price we have to pay for shorter response times is the overhead caused by the management of the caching framework and by maintaining cache consistency. As a result of active prefetching, the database agent will have to handle an increased number of queries. An optimal tradeoff between the reduction of response times and the resulting overhead trying to improve the caching hit rate had to be found.

In chapter 3 we describe the architecture of the personal assistant and his different caches. The user interfaces of the personal assistant, the database agent and the extended frontend agent will be the content of chapter 4. Finally, the object signing mechanism necessary to give applets additional power will be discussed in chapter 5.

Chapter 3

Architecture

3.1 Overview

There are three different "subcaches" building up the personal assistant's caching framework: The *session cache*, the *personal cache* and a *prefetching cache* (see figure 3.1). These three caches are distinct, i.e. the same information will be cached at most in one of the caches. From section 3.2 to section 3.4 we will describe the architecture of the three subcaches an their tasks.

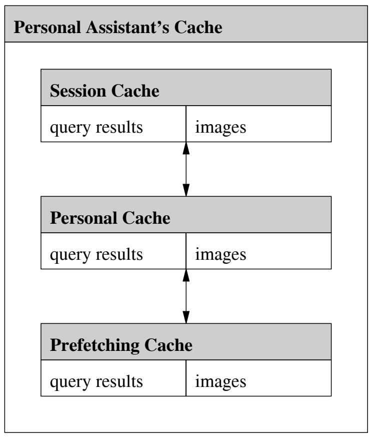


Figure 3.1: Personal assistant's caching framework

Each of the three caches is separated in a part which will cache query results and an image caching part. We have chosen this approach because the image sizes are large relative to the query results and often the same image is required by different queries. Due to this separation, an image will always be cached at most once, even if it is needed by several queries. The caching space saved by this strategy, i.e. by eliminating image redundancy, is used to cache additional queries. Every time a user removes a query from the cache (e.g. from the *personal cache*), the corresponding images should also be deleted. By using the separation strategy, we always have to check if an image is still required by other queries before removing it from the cache, i.e. we had to introduce a reference counter (indicating the number of queries using the image) to process *garbage collection.*

The personal assistant will not cache any references to HTML pages and multimedia files different to images (sound, movies etc.). Caching of this data will be done by the internal cache of the browser running the application.

3.2 Session Cache

The session cache, as its name suggests, acts as a short time cache and will only cache queries recently used. Therefore the caching strategy used by the session cache is a simple least recently used (LRU) strategy (for further information about caching strategies see [Tan92, SG94, Fra96]). By using an LRU replacement strategy, the session cache will profit from the locality of a single session.

3.3 Personal Cache

All the queries explicitly defined by the user to be permanently cached will be managed by the personal cache. It is made persistent every time the personal assistant is shut down, i.e. it will be written to disk and restored when the personal assistant is restarted next time. The personal cache is optimal for so-called "entry-queries" (a small amount of queries, which can be chosen from the start page) and to permanently cache information the user often requires.

To be sure the data cached by the personal cache is always up to date, every time a new entry is added to the personal cache, the personal assistant will register it on the database agent. Whenever data changes on the database agent, he will check if queries registered by the personal assistant will be affected. If so, the database agent will send a cache invalidation message to the personal assistant, forcing him to update his cache to maintain cache consistency. After removing a query from one of the personal assistant's caches, the query will also be unregistered on the database agent. Further, every time the personal assistant is started, he has to register the whole contents of his persistent caches on the server.

The personal cache doesn't have a caching strategy, i.e. if there is no more place in the cache, the cache manager won't select a victim and replace it by the new entry. If the user tries to add a new query to the personal cache and there is no more place in it, the personal assistant will inform him that there is no more place in the personal cache and that he either has to manually remove some entries from the cache or to increase the personal cache size. This "caching strategy" makes sense because the user wants to be sure that all the queries he added to the personal cache will be present in the future and not being deleted by the cache manager without his knowledge.

From the personal assistant's three subcaches, the personal cache has the highest priority because the user explicitly defined the queries to be permanently cached. If an entry is added to the personal cache and is either present in the session or the prefetching cache, it will be removed from these caches to avoid cache redundancy. Further a query won't be added neither to the session nor to the prefetching cache if it is already present in the high priority personal cache.

3.4 Prefetching Cache

In addition to the short time *session cache* and the user defined *personal cache*, the intelligent *prefetching cache* will analyze user behaviour and try to predict the information the user will need in the near future. It dynamically adapts to a users preferences without any interaction.

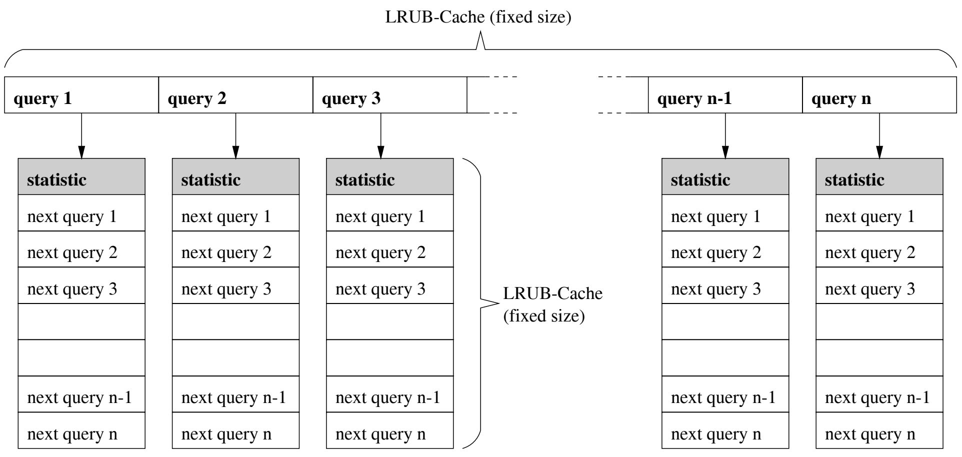


Figure 3.2: Structure of the prefetching statistic

For each query requested by the user (*query 1* to *query n* in figure 3.2), the prefetching cache maintains a statistic of the succeeding queries and the number of times they were required immediately after the current query (*next query 1* to *next query n* in figure 3.2). The statistic is represented by a cache of limited size. So for each query only a limited amount of possible succeeding queries – the queries which will most likely follow the current query – are considered in the statistic. The simplest replacement strategy for these statistic caches would be an LRU strategy. The problem is, that these caches should act as long time caches, i.e. the statistic will grow over a long period. Therefore we should also consider the number of accesses for each cache entry, when selecting a victim. As a result, the LRU strategy is extended by the number of times a cache entry was requested. The number of cache hits acts as a bonus for entries which where often used in the past but hadn't been accessed during a longer period (LRUB cache). Considering these facts, we suggest the following weighting formula for cache entry i:

$$weight_i = \frac{\alpha + (1-\alpha)\log(\text{number of hits})}{\log(\text{time since last access})} \qquad 0 \leq \alpha \leq 1 \tag{3.1}$$

Every time an element has to be added to a full cache, the cache entry with the smallest weight will be selected as a victim and removed from the cache. The memory of the cache can be adjusted by changing the α value in equation 3.1. The smaller the value of α, the more the cache will memorize the number of cache hits (by setting $\alpha = 1$ we have a normal LRU cache).

The queries with their statistic are also maintained in a cache of limited size using the same LRUB caching strategy. Therefore only for the queries most likely being used in the future, a statistic of the succeeding queries will be present in the prefetching statistic. Every time a user requests a new query, the personal assistant will check if there exists a statistic for it. If he finds a corresponding statistic, he will select the queries with the highest weights (the queries which will most likely succeed the current query) from it. These queries finally will be prefetched into the prefetching cache.

At the moment, the number of queries to be prefetched (prefetching width) is set to three, i.e. for each query at most three queries will be prefetched. By increasing the prefetching width, the cache hit rate should also increase but as a negative effect the database agent will have to handle an additional amount of queries.

The prefetching cache will profit from the fact that in the internet based interface for the OMS database, users often browse the same way, i.e. there will be chains of queries often requested in the same order. The great advantage of this new prefetching mechanism over a “static” strategy globally caching the most frequently used queries is, that the prefetching cache, like a sliding window, always focuses the current query. As a result the prefetching cache only contains the data most likely required in the near future and therefore needs much less memory than a global strategy trying to achieve the same hit rate!

3.5 Startup Process

The applications startup process was redesigned and simplified giving the applets more network power by using object signing (see chapter 5). Each time a personal assistant is started, he will send the IP-address and the port he is listening for clients to the database agent. When a new frontend agent is started within the browser, he will first connect to the database agent, sending him the IP-address of the user's machine and asking for the IP-address and port he has to connect to. The database agent will check if there exists a registered personal assistant with the same IP-address as the address received from the frontend agent. If a personal assistant is present, he will send the IP-address and port of the personal assistant to the frontend agent, otherwise he will send his own IP-address and port (direct connection to the database agent). For a user starting the applet, the process will look the same whether he has installed a personal assistant or not.

Chapter 4

User Interface

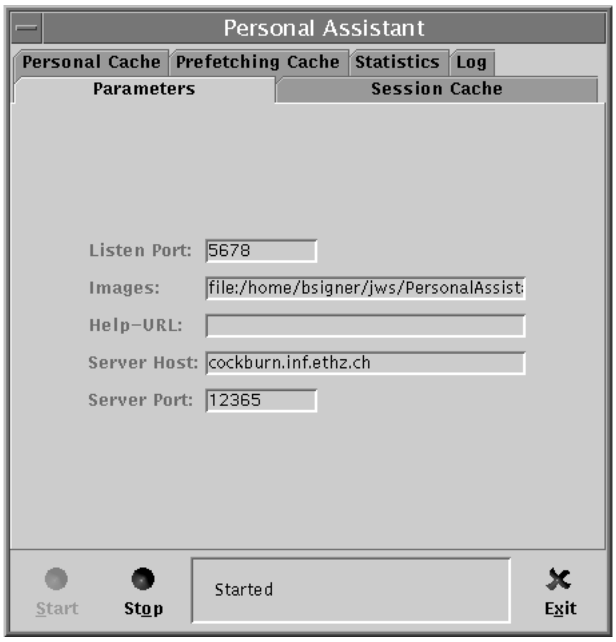
The main task of this work was to develop a personal assistant. After implementing the personal assistant, we noticed that it would be nice to reuse the caching framework for the database agent as well as for the frontend agent. Further this allowed us to use a visualization of the database and frontend agent caches similar to the one used for the personal assistant. In the next three sections we will describe the user interfaces of the personal assistant, the database agent and the frontend agent.

4.1 Personal Assistant

After starting up the personal assistant, the agent's main window containing six tab folders will appear. The control panel on the bottom of the application allows to start, stop or exit the personal assistant. Further all important messages will be shown in the center of the control panel. From section 4.1.1 to section 4.1.6 we will describe the content of the different tab folders. The additional power added to the default *Internet OMS* user interface will be discussed in section 4.3.3.

4.1.1 Parameters

The parameter tab folder contains parameters relevant for connecting to other agents, locations of the file caches etc. as shown in figure 4.1. First we have to specify the location of the desired database agent. The `Server Host` field contains the IP-address of the host the database agent is located whereas the `Server Port` defines the port the database agent is listening for clients. Further we have to assign a unique port to the personal assistant itself on which he will listen for potential clients (`Listen Port`). `Help-URL` is the location of the applications help files, if present. A base directory for the storage of the file caches containing images and queries has further to be defined in the field `Images`.



Listen Port: 5678
Images: file:/home/bsigner/jws/PersonalAssist
Help-URL:
Server Host: cockburn.inf.ethz.ch
Server Port: 12365
Started

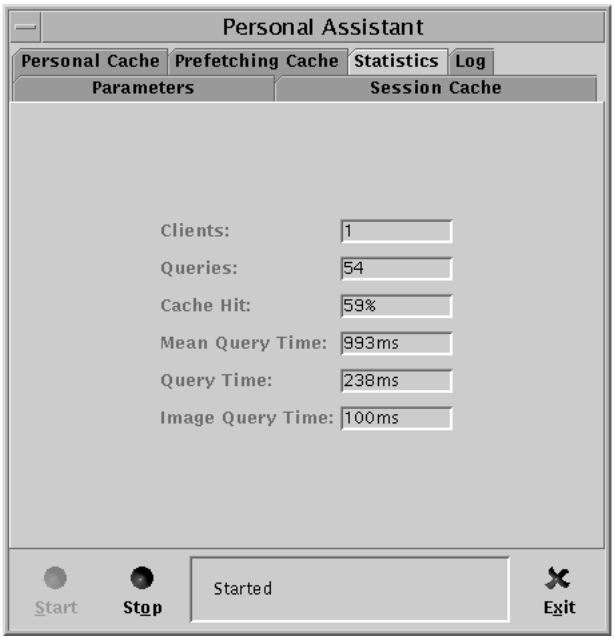
Figure 4.1: Personal assistant parameters

4.1.2 Statistics

The statistics tab folder shows information concerning the overall performance of the caching framework (see figure 4.2). The total number of clients currently connected to the personal assistant is shown in the `Clients` field. `Queries` indicates the number of queries handled during the session. An interesting parameter is the `Cache Hit`, indicating the overall cache hit rate of the three subcaches building up the personal assistant's caching framework. A second important parameter is the mean time the personal assistant has to handle a query (`Mean Query Time`). It is calculated by the mean time to handle a query and the mean time to handle an image request. `Query Time` is the time from the reception of a query by a client to the end of the transmission of the answer. Finally, `Image Query Time` is the time between the reception of an image request and the transfer of the local image address to the client, i.e. in particular it includes the time to transfer the image from the database agent to the personal assistant's cache.

4.1.3 Log View

Figure 4.3 shows the log view containing additional information to the messages shown in the control panel at the bottom of the application. If the application shows an abnormal behaviour, it is a good idea to inspect the log view and to search for any warnings or errors. Further the log view shows additional information about processed queries, clients connecting to



Clients:	1
Queries:	54
Cache Hit:	59%
Mean Query Time:	993ms
Query Time:	238ms
Image Query Time:	100ms

Started

Figure 4.2: Personal assistant statistics

the agent and other network specific details. All information presented in the log view is stored in a log file (`Log.txt`) in the base caching directory and therefore also can be inspected after terminating the personal assistant by using a standard text editor.

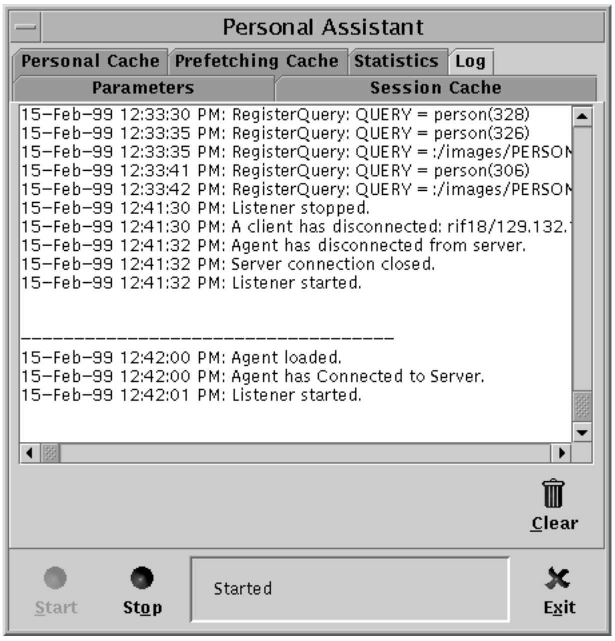
4.1.4 Session Cache

The cache view described in this section will be similar for all the following cache views (e.g. the prefetching and personal cache views). As shown in figure 4.4 each cache view is separated in two main parts to show the two subcaches (query cache and image cache, respectively). The views of the subcaches are equivalent, so it suffices to describe one of them only.

By pressing the `Delete` button, the user can delete the content of the whole cache (see figure 4.6). After confirming the delete message box, all cache elements will be removed, i.e. in case of a file cache, the files will be deleted from the local disk.

To resize a cache press the `Resize` button and enter the desired new cache size (see figure 4.7). If the new cache size is smaller than the size of all entries currently in the cache, the cache manager will select victims and remove them until the overall size of the cache entries fits the new cache size.

By pressing the `View` button you can visualize the cache content as shown in figure 4.5 for the prefetching query cache. Each cache entry is referenced

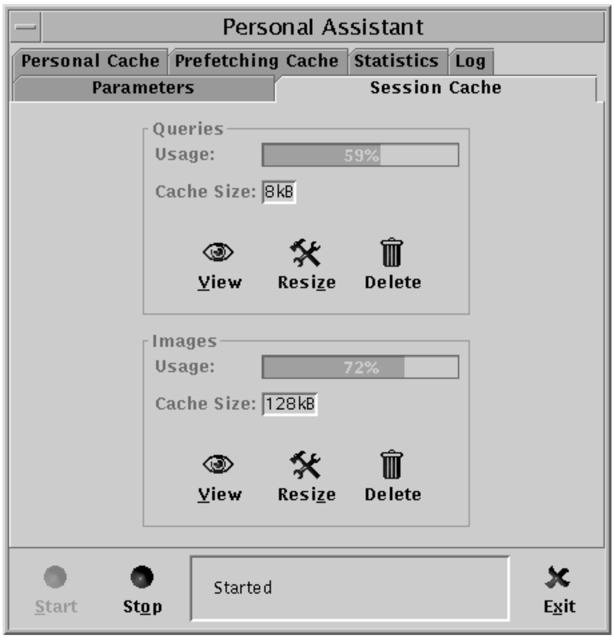


```
15-Feb-99 12:33:30 PM: RegisterQuery: QUERY = person(328)
15-Feb-99 12:33:35 PM: RegisterQuery: QUERY = person(326)
15-Feb-99 12:33:35 PM: RegisterQuery: QUERY = :/images/PERSON
15-Feb-99 12:33:41 PM: RegisterQuery: QUERY = person(306)
15-Feb-99 12:33:42 PM: RegisterQuery: QUERY = :/images/PERSON
15-Feb-99 12:41:30 PM: Listener stopped.
15-Feb-99 12:41:30 PM: A client has disconnected: rif18/129.132.
15-Feb-99 12:41:32 PM: Agent has disconnected from server.
15-Feb-99 12:41:32 PM: Server connection closed.
15-Feb-99 12:41:32 PM: Listener started.


-----------------------------------
15-Feb-99 12:42:00 PM: Agent loaded.
15-Feb-99 12:42:00 PM: Agent has Connected to Server.
15-Feb-99 12:42:01 PM: Listener started.
```

Started

Figure 4.3: Log View



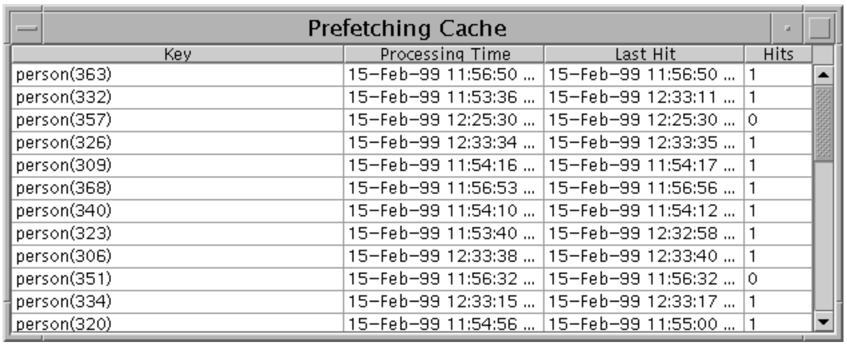
Queries
Usage: 59%
Cache Size: 8kB

Images
Usage: 72%
Cache Size: 128kB

Started

Figure 4.4: Personal assistant session cache

by `Key` which in the case of the *Internet OMS* framework is always equivalent to the query. The time the query was processed by the database agent is



Prefetching Cache

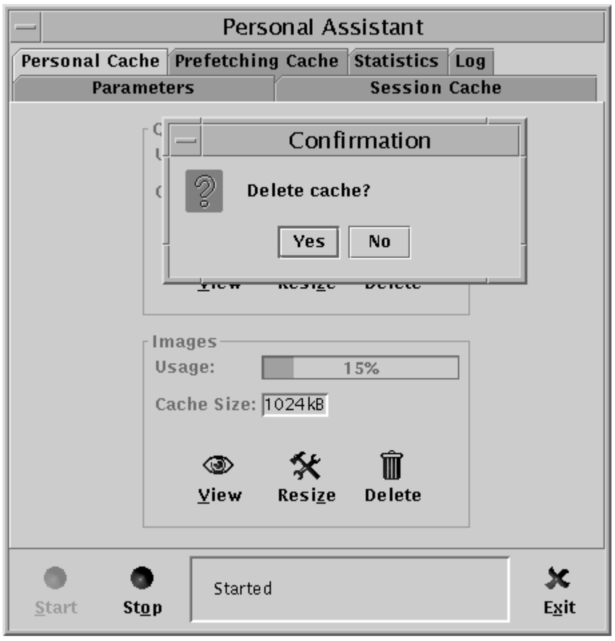
Key	Processing Time	Last Hit	Hits
person(363)	15–Feb–99 11:56:50 ...	15–Feb–99 11:56:50 ...	1
person(332)	15–Feb–99 11:53:36 ...	15–Feb–99 12:33:11 ...	1
person(357)	15–Feb–99 12:25:30 ...	15–Feb–99 12:25:30 ...	0
person(326)	15–Feb–99 12:33:34 ...	15–Feb–99 12:33:35 ...	1
person(309)	15–Feb–99 11:54:16 ...	15–Feb–99 11:54:17 ...	1
person(368)	15–Feb–99 11:56:53 ...	15–Feb–99 11:56:56 ...	1
person(340)	15–Feb–99 11:54:10 ...	15–Feb–99 11:54:12 ...	1
person(323)	15–Feb–99 11:53:40 ...	15–Feb–99 12:32:58 ...	1
person(306)	15–Feb–99 12:33:38 ...	15–Feb–99 12:33:40 ...	1
person(351)	15–Feb–99 11:56:32 ...	15–Feb–99 11:56:32 ...	0
person(334)	15–Feb–99 12:33:15 ...	15–Feb–99 12:33:17 ...	1
person(320)	15–Feb–99 11:54:56 ...	15–Feb–99 11:55:00 ...	1

Figure 4.5: Cache content

shown in the `Processing Time` field. This value is essential to check the validity of a cache entry. Each time a cache hit occurs, the time of the last access (`Last Hit`) and the number of cache hits (`Hits`) will be updated.

4.1.5 Personal Cache

The personal cache user interface (see figure 4.6) looks the same as the session cache interface discussed in section 4.1.4. The personal cache is the



Personal Assistant

Confirmation

Delete cache?

Images
Usage: 15%
Cache Size: 1024kB

Started

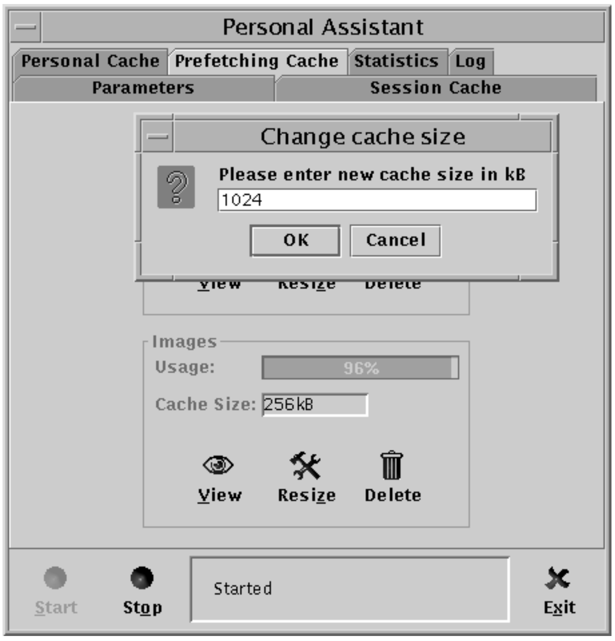
Figure 4.6: Delete a personal cache

only cache, the content of which can be directly manipulated by the user. If a regular user has running a personal assistant and starts the frontend agent, the menu of the agent will be automatically extended by some additional commands, allowing him to manipulate the contents of the personal cache as described in section 4.3.

When resizing the personal cache, it shows a different behaviour than the other caches. If you try to resize either the personal query or image cache to a size smaller than the entries currently in the cache, you will be informed that the new size is too small. You first have to manually remove elements from the cache as long as the size is larger than the desired new cache size. Why such a complicated mechanism in case of reducing the size of the personal cache to a size smaller than the elements currently in the cache? The cache managers of the personal caches don't remove any element for you. This makes sense, because you explicitly defined the queries of the personal cache to be permanently cached and therefore they shouldn't be removed by a cache manager without your control.

4.1.6 Prefetching Cache

Also the prefetching cache (see figure 4.7) looks identical to the session cache described in section 4.1.4. There is one important additional detail



Please enter new cache size in kB: 1024

Usage: 96%

Cache Size: 256kB

Started

Figure 4.7: Changing the prefetching cache size

concerning the prefetching cache. If you delete either the prefetching query or the prefetching image cache, only the caches will be deleted but not the statistics the actual prefetching is based on. If you want not only to delete the prefetching cache but also to reset the prefetching statistics, you have to delete the `stat.idx` file located in the prefetching query cache directory and the prefetching image cache directory, respectively.

4.2 Database Agent

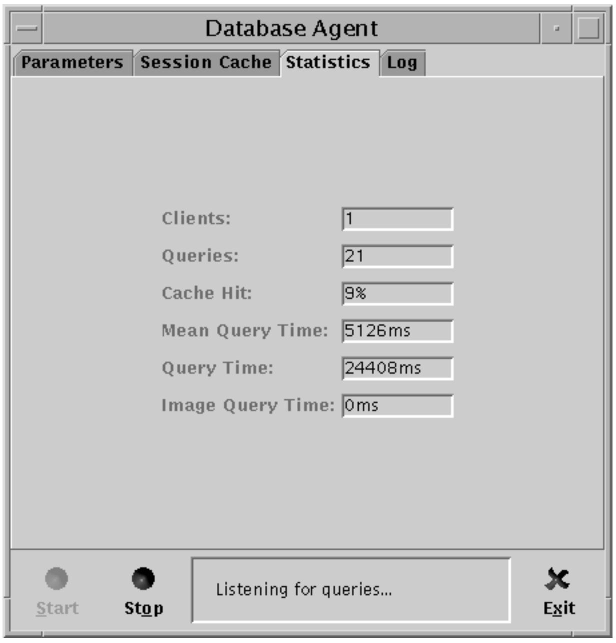
Most of the database agent's user interface has the same functionality as the user interface of the personal assistant. Therefore we only describe the parts different to the personal assistant's user interface. For further information about the personal assistant's user interface see section 4.1.

4.2.1 Parameters

In contrast to the personal assistant, `Images` indicates where all the images used by the OMS database are located.

4.2.2 Statistics

The statistics looks exactly the same as the statistics of the personal assistant (see figure 4.8). The only difference to the personal assistant's statistics



Clients: 1
Queries: 21
Cache Hit: 9%
Mean Query Time: 5126ms
Query Time: 24408ms
Image Query Time: 0ms

Listening for queries...

Figure 4.8: Database agent statistics

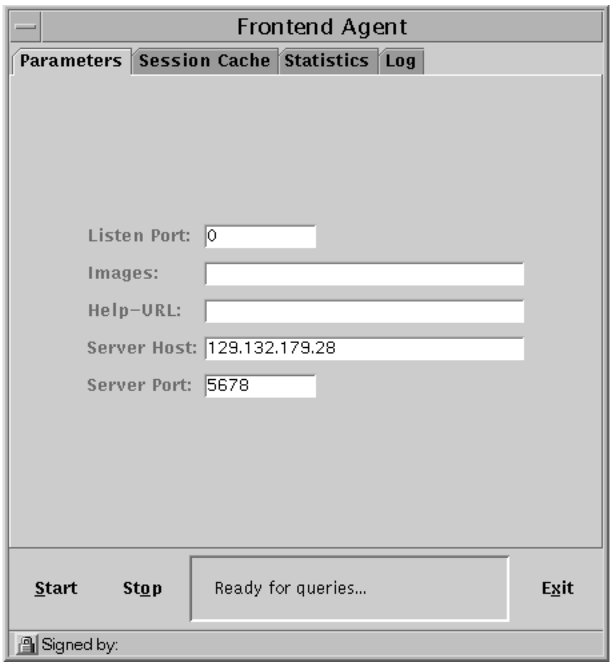
is the interpretation of the `Image Query Time`. The value of this field informs you about the time the database agent had to send the address of the local image to his client, i.e. it doesn't include any transmission time for images because the database agent never has to fetch images. Therefore this value will be relatively constant and small compared to the equivalent parameter of the personal assistant.

4.3 Frontend Agent

The frontend agent consists of two user interfaces: a user interface allowing to control the cache and the *Internet OMS* user interface used to browse the database. While the former looks similar to the caching interfaces described in section 4.1 and section 4.2, the latter was extended by functionalities to add queries to the personal cache.

4.3.1 Parameters

The `Images` field normally used to set the base directory for images and caching, doesn't influence the frontend agent, because all the caches of the frontend agent are memory caches which will not be made persistent and stored on disk when leaving the agent. The `Listen Port` will always be zero indicating that the frontend agent doesn't listen for any clients (see figure 4.9).



Listen Port: 0
Images:
Help-URL:
Server Host: 129.132.179.28
Server Port: 5678
Ready for queries...

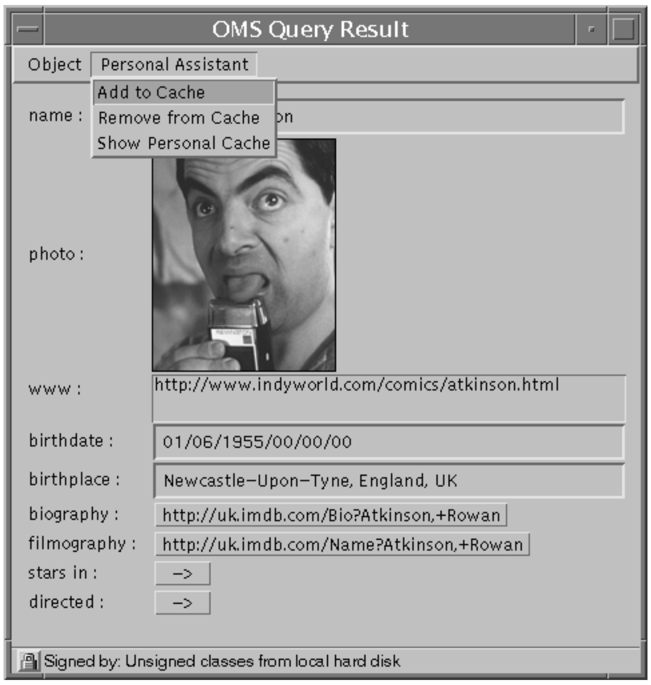
Figure 4.9: Frontend agent parameters

4.3.2 Statistics

The statistics looks the same as the personal assistant's statistics (see figure 4.2). `Query Time` and `Image Query Time` represent the time passed between sending a query to the server and receiving the corresponding answer from the server.

4.3.3 Personal Cache

The *Internet OMS* user interface was extended by the facility to add a query result to the personal assistant's personal cache (see figure 4.10). A



name : on
photo :
www : http://www.indyworld.com/comics/atkinson.html
birthdate : 01/06/1955/00/00/00
birthplace : Newcastle–Upon–Tyne, England, UK
biography : http://uk.imdb.com/Bio?Atkinson,+Rowan
filmography : http://uk.imdb.com/Name?Atkinson,+Rowan
stars in : –>
directed : –>
Signed by: Unsigned classes from local hard disk

Figure 4.10: Frontend agent menu

regularly used query can be added to the personal cache by selecting the `Add to cache` entry from the new `Personal Assistant` menu. If you want to add a new entry to the cache but there is no more place in the personal cache, the new entry wont be added and you will be informed that you either have to manually remove some entries from the cache or to increase the size of the personal cache.

To remove a query from the cache, the user has to select the `Remove from Cache` entry in the `Personal Assistant` menu of the query to be deleted. Because it isn't very user friendly always having to browse to the query you want to remove, there exists a second possibility to remove an entry from the personal cache which will be described in the next paragraph.

By selecting the `Show Personal Cache` entry from the `Personal Assistant` menu, a window with the contents of the personal cache as shown in figure 4.11 will be displayed. If you don't know the information behind a

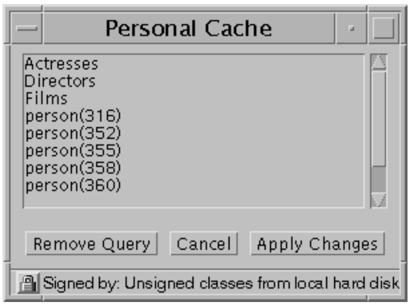


Figure 4.11: Personal cache entries

query, you can view the corresponding data by double clicking the query in the list of cache entries – it is possible to browse your personal cache. To remove a query from the personal cache, highlight the corresponding query in the list of cached queries and press the `Remove Query` button. The query will be deleted from the list of cached queries but not yet from the cache itself until you press the `Apply Changes` button. So if you unfortunately removed queries from the cache you didn't want to remove, by pressing the `Cancel` button the entries won't be deleted from the cache.

Chapter 5

Object Signing

5.1 Relaxing the Sandbox Restrictions

By default, Java applets are contained within a “sandbox”, i.e. they underlie a lot of security restrictions. Within the *Internet OMS* framework, it is necessary the applet can open network connections to locations different from where it came from (e.g. it has to communicate with the local personal assistant). Therefore we had to relax the sandbox restrictions to give the applet additional capabilities. In this chapter, we will describe the essential steps to give applets more power by digitally signing them.

Fortunately recent browsers[1] have provisions to give trusted applets the ability to work outside the sandbox . For this power to be granted to your applet, you will need to use Netscape-specific Java classes called the “Netscape Capabilities API”[2]. You will have to add additional code to the applet, requesting permission to do any “dangerous” actions and then digitally sign it with an unforgeable digital ID (private key)

Everybody is able to identify the signer of a digitally signed object and further can determine whether the object has been tampered since it was signed. Instead of requiring users to allow software either unlimited or no access at all, object signing allows the user to control the degree of access.

Each time a user runs a signed applet requesting some form of access to the local system, the browser will check its granted privileges for the corresponding signer. If the user already granted the requested additional power to the signer during an earlier session, the browser allows the access without interrupting the user’s activities; otherwise the browser will show a Java security dialog box asking if the user will trust the developer the applet was signed by.

[1]All the explanations in this chapter refer to the Netscape Communicator. Using Microsoft Internet Explorer object signing works quite different.

[2]http://developer.netscape.com/docs/manuals/signedobj/capsapi_classes.zip

5.2 Digital Signatures

Public key cryptography always involves a pair of keys (public key and private key) associated with a person wanting to send secure messages or digitally sign messages. The public key is published, whereas the private key is kept secret. A message encrypted with the private key can only be decrypted with the corresponding public key and analogous, a message encrypted with the public key can only be decrypted with the corresponding private key.

To obtain a pair of valid keys, you have to request a certificate from a certification authority (CA). After generating a private key and the corresponding public key (by using a crypto-tool), you have to send the public key to a certification authority which – after verifying your identity – will issue a certificate. This certificate is a digital document, binding a particular public key to a specific person.

To digitally sign an object, you have to encrypt it with your private key. Because public key cryptography isn't very fast, not the data itself but a small hash value generated by a *one-way hash function* applied to the corresponding data will be encrypted. A *one-way hash function* generates hash numbers of fixed length with the following characteristics:

- The hash values can be computed *efficiently* using the one-way hash function $f : A \rightarrow B$.
- For a given $f(x)$ (where $x \in A$) it is too difficult to find an $x' \in A$ so that $f(x') = f(x)$ (one-way function).
- The value of the hash is "unique" for the hashed data, i.e. any change in the data, even deleting or altering only a single character, results in a different hash value.

Figure 5.1 illustrates the process of signing an object. There are two items transferred from the sender (Alice) to the receiver of the signed object (Bob): the original data and the digital signature, which is a one-way hash value of the original data encrypted with Alice's private key. To validate the integrity of the data, Bob first uses Alice's public key to decrypt the one-way hash value. He then uses the same one-way hash function Alice used computing her hash value, and generates a new hash value of the received data. If the two hash values match, Bob can be sure that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer additionally requires a method to ensure the public key really belongs to the corresponding person, which is the task of a CA as described earlier.

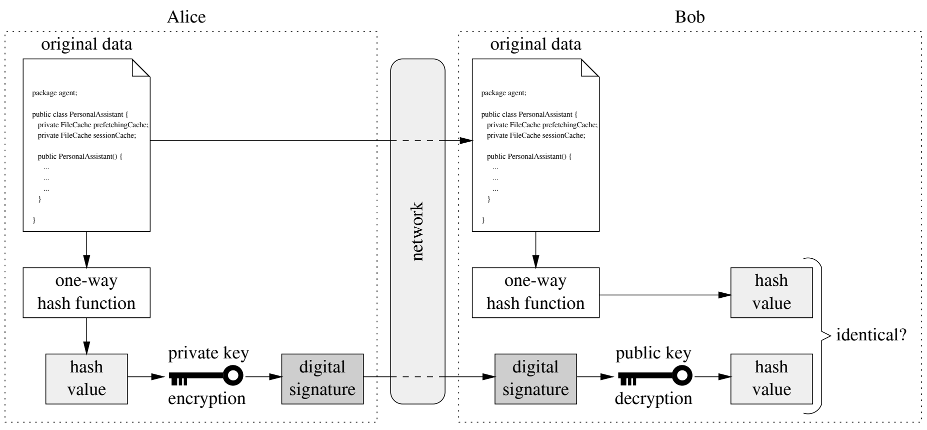


Figure 5.1: Object signing

5.3 Netscape Signing Tool

After this short theoretical introduction to object signing in the last section, we will explain now, how to use the Netscape *signtool* to digitally sign your code. Signing a file using the Netscape signing tool requires the following steps:

1. Assign a password to your Netscape private key database if not yet done.
2. Download the Netscape signtool[3].
3. If you don't yet have your own certificate, get one from a CA (like Verisign, Thawte etc.) or generate a test certificate using the signtool (`signtool -G CERT_NAME`). Before generating a test certificate using the signtool, make sure there are no instances of the Netscape Communicator currently running.
4. Create an empty directory: `mkdir SIGNDIR`
5. Put the files to be signed into the new directory.
6. Sign the directory by specifying the name of your object-signing certificate: `signtool -k CERT_NAME -Z OUTPUT_FILE.jar SIGNDIR`
7. Type the password of your private key database when asked.
8. Test the archive just created: `signtool -v OUTPUT_FILE.jar`

The generated JAR-file acts as a digital envelope for a compressed collection of files. To use the digitally signed code, you have to specify the

[3]http://developer.netscape.com/software/signedobj/jarpack.html

JAR-file in the HTML-code containing the corresponding applet using the `archive` tag as follows:

```
<applet archive="OUTPUT_FILE.jar" code="APPLET_NAME.class">
</applet>
```

If you are not sure which certificates are available for code signing, you can use `signtool -L` to get a list of all object-signing certificates marked by an asterisk.

Communicator's certificate and key databases are portable among all platforms. If you obtained your object-signing certificate while running Communicator on a system different from the system on which you intend to sign files, you need to copy your certificate and private key files to the new system. Locate the files `key3.db` and `cert7.db` and copy them on the system where you intend to sign pages (on unix machines this will be the `/.netscape` directory. It is useful to always keep copies of the `key3.db` and `cert7.db` files somewhere separate, ensuring that you won't lose your certificates if you accidentally damage them using the *signtool* executable.

To import a new CA in the Netscape Communicator, make sure your web server knows the MIME-type application/x-x509-ca-cert. To arrange this, your system administrator has to associate this MIME-type with the file extension `.cacert`. Depending on your web server, this may involve editing a configuration file or using an administration tool. When opening now the `X509.cacert` within the Netscape Communicator, it will ask you, if you want to import the new CA.

Because during the development of a new application it is very time consuming always have to sign your code after changing it, there exists a possibility to get more privileges for applets without signing them, which will be discussed in the next section.

5.4 Codebase Signing

When developing secure Java code it is often quite useful to bypass the signing stage during the development cycle. For this reason, developers may add two additional lines to their Netscape Communicator's preference file allowing the codebase of the applet (file-URL or http-URL) to function as a principal for enabling privileges. As long as the applet stays at the same location, you can run it after changing it's code without having to perform any security procedures. To activate codebase principals you have to edit the file `preferences.js` which is located in the Netscape directory. Before editing the file, make sure that all instances of the Netscape Communicator are shut down. Then add the following two lines to the `preferences.js` file:

```
user_pref("signed.applets.codebase_principal_support",true);
user_pref("signed.applets.local_classes_have_30_powers",true);
```

This change to the preferences file is meant for development use only. After activating codebase principals, you have to be very careful browsing the internet!

Appendix A

API Reference

A.1 The agent Package

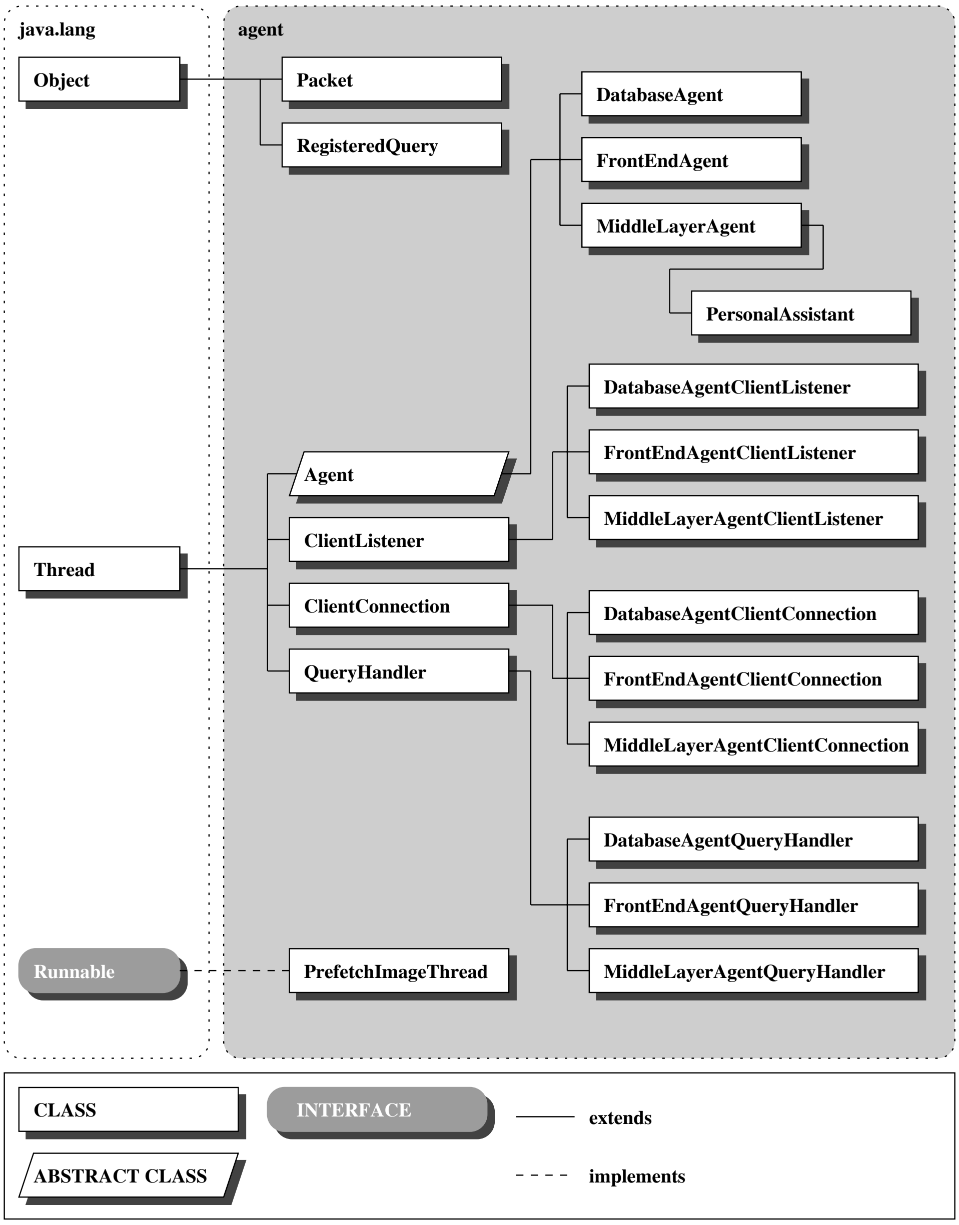

java.lang
Object
Thread
Runnable
agent
Packet
RegisteredQuery
DatabaseAgent
FrontEndAgent
MiddleLayerAgent
PersonalAssistant
DatabaseAgentClientListener
FrontEndAgentClientListener
MiddleLayerAgentClientListener
Agent
ClientListener
ClientConnection
QueryHandler
DatabaseAgentClientConnection
FrontEndAgentClientConnection
MiddleLayerAgentClientConnection
DatabaseAgentQueryHandler
FrontEndAgentQueryHandler
MiddleLayerAgentQueryHandler
PrefetchImageThread
CLASS
INTERFACE
extends
ABSTRACT CLASS
implements

Figure A.1: The agent package

Class agent.Agent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.Agent
```

---

public abstract class **Agent**
extends Thread

The Agent is the basic component which builds up the agent-framework. The Agent was developped by P. Ruser. Improvements and changes have been done by Antonia Erni. Redesign has been done by Beat Signer. The Agent class provides the key functionality which is common to all agents: user-interface, communication via sockets, query registration, messaging, client and server tasks, client and server communication, cache handling, query handling, error handling, etc. All agents using the Agent-Framework derive from the base-class "Agent".

Variables

● **NO_VALUE**

```
public static final int NO_VALUE
```

● **NO_DATA**

```
public static final String NO_DATA
```

● **SIMPLE_QUERY**

```
public static final int SIMPLE_QUERY
```

● **ANSWER**

```
public static final int ANSWER
```

● **COMMAND_OK**

```
public static final int COMMAND_OK
```

● **GET_IMAGE_BASE**

```
public static final int GET_IMAGE_BASE
```

● **IMAGE_BASE**

```
public static final int IMAGE_BASE
```

● **GET_IMAGE**

```
public static final int GET_IMAGE
```

● **SERVER_SHUTDOWN**

```
public static final int SERVER_SHUTDOWN
```

● **CACHE_INVALIDATION**

```
public static final int CACHE_INVALIDATION
```

● **REGISTER_QUERY**

```
public static final int REGISTER_QUERY
```

● **UNREGISTER_QUERY**

```
public static final int UNREGISTER_QUERY
```

● **REGISTER**

`public static final int REGISTER`

● **GET_HELP_URL**

`public static final int GET_HELP_URL`

● **HELP_URL**

`public static final int HELP_URL`

● **GET_START_ADDRESS**

`public static final int GET_START_ADDRESS`

● **START_ADDRESS**

`public static final int START_ADDRESS`

● **ADD_TO_PERSONAL_CACHE**

`public static final int ADD_TO_PERSONAL_CACHE`

● **REMOVE_FROM_PERSONAL_CACHE**

`public static final int REMOVE_FROM_PERSONAL_CACHE`

● **GET_CACHED_QUERIES**

`public static final int GET_CACHED_QUERIES`

● **CACHED_QUERIES**

`public static final int CACHED_QUERIES`

● **FETCH**

`public static final int FETCH`

● **FETCH_ANSWER**

`public static final int FETCH_ANSWER`

● **PREFETCH**

`public static final int PREFETCH`

● **PREFETCH_ANSWER**

`public static final int PREFETCH_ANSWER`

● **INITIATE_PREFETCH**

`public static final int INITIATE_PREFETCH`

● **VIRTUAL_QUERY**

`public static final int VIRTUAL_QUERY`

● **VIRTUAL_IMAGE**

`public static final int VIRTUAL_IMAGE`

● **CACHING**

`public static final int CACHING`

● **NO_CACHING**

`public static final int NO_CACHING`

● **SYNTAX_ERROR**

`public static final int SYNTAX_ERROR`

● **FATAL_ERROR**

```
public static final int FATAL_ERROR
```

● **TIMEOUT_ERROR**

```
public static final int TIMEOUT_ERROR
```

● **NO_LINK_ERROR**

```
public static final int NO_LINK_ERROR
```

● **NO_PLACE_ERROR**

```
public static final int NO_PLACE_ERROR
```

● **COMMAND_OK_PACKET**

```
public static final Packet COMMAND_OK_PACKET
```

● **GET_IMAGE_BASE_PACKET**

```
public static final Packet GET_IMAGE_BASE_PACKET
```

● **SERVER_SHUTDOWN_PACKET**

```
public static final Packet SERVER_SHUTDOWN_PACKET
```

● **GET_HELP_URL_PACKET**

```
public static final Packet GET_HELP_URL_PACKET
```

● **GET_CACHED_QUERIES_PACKET**

```
public static final Packet GET_CACHED_QUERIES_PACKET
```

● **SYNTAX_ERROR_PACKET**

```
public static final Packet SYNTAX_ERROR_PACKET
```

● **FATAL_ERROR_PACKET**

```
public static final Packet FATAL_ERROR_PACKET
```

● **TIMEOUT_ERROR_PACKET**

```
public static final Packet TIMEOUT_ERROR_PACKET
```

● **NO_LINK_ERROR_PACKET**

```
public static final Packet NO_LINK_ERROR_PACKET
```

● **NO_PLACE_ERROR_PACKET**

```
public static final Packet NO_PLACE_ERROR_PACKET
```

● **NULL_PACKET**

```
public static final Packet NULL_PACKET
```

● **DELIMITER**

```
public static final String DELIMITER
```

● **BUFFER_SIZE**

```
public static final int BUFFER_SIZE
```

● **CHAR_SIZE**

```
public static final int CHAR_SIZE
```

● **helpURL**

```
public String helpURL
```

● **imageBase**

```
protected String imageBase
```

● **serverImageBase**

```
public String serverImageBase
```

● **host**

```
protected String host
```

● **port**

```
protected int port
```

● **listenPort**

```
protected int listenPort
```

● **PrefURL**

```
protected String PrefURL
```

● **cacheDir**

```
protected String cacheDir
```

● **registeredQueries**

```
protected Hashtable registeredQueries
```

● **currentQueries**

```
protected Hashtable currentQueries
```

● **currentQueriesPacket**

```
protected Hashtable currentQueriesPacket
```

● **lastQuery**

```
protected long lastQuery
```

● **server**

```
public Socket server
```

● **listen_socket**

```
public ServerSocket listen_socket
```

● **stream_in**

```
protected DataInputStream stream_in
```

● **stream_out**

```
protected PrintWriter stream_out
```

● **clientConnections**

```
public Vector clientConnections
```

● **disconnectAllways**

```
protected boolean disconnectAllways
```

● **clientListener**

```
protected ClientListener clientListener
```

serverListener

```
protected ServerListener serverListener
```

middleLayerAgentPorts

```
protected Hashtable middleLayerAgentPorts
```

showMainWindow

```
protected boolean showMainWindow
```

mainWindow

```
protected AgentView mainWindow
```

preferences

```
protected Properties preferences
```

Constructors

Agent

```
public Agent(boolean showMainWindow,
             boolean disconnectAllways,
             String path)
```

Methods

init

```
public void init()
```

sendNoLinkError

```
public void sendNoLinkError()
```

Sends a NO_LINK_ERROR to all waiting clients. This error will occure when the server was shut down.

increaseClients

```
public void increaseClients()
```

decreaseClients

```
public void decreaseClients()
```

registerClient

```
public void registerClient(ClientConnection client)
```

Registers a new client on the agent which keeps a list of all connected clients and the links it has to these clients. This is done to be able to send a shutdown-message to all the connected clients.

Parameters:
client - Client to be registered.

sendSimpleNonCachingQuery

```
public void sendSimpleNonCachingQuery(String query)
```

Send a simple query non cached query.

openMainWindow

```
protected abstract void openMainWindow(boolean showMainWindow)
```

Opens the main window.

● **readLog**

```
protected void readLog()
```

Reads the log from a file.

● **writeLog**

```
public void writeLog()
```

Writes the log to a file.

● **readPreferences**

```
protected void readPreferences()
```

Reads the preferences from a file.

● **writePreferences**

```
public void writePreferences()
```

Writes the preferences to a file.

● **updatePreferences**

```
public void updatePreferences()
```

Updates the preference variables if the user changed them on the graphical agent window.

● **printToLog**

```
public void printToLog(String text)
```

Prints a message to the log.

Parameters:
text - Text to be printed to the log.

● **startServerConnection**

```
public abstract boolean startServerConnection()
```

Tries to connect to another agent which is listening on a specific port. If successful, a server task is created which listens to messages from the server agent.

Returns:
TRUE if connected sucessfully, FALSE otherwise.

● **stopServerConnection**

```
public void stopServerConnection(boolean propagate)
```

Stops the connection to the server agent and ends the task which is listening for messages from the server.

● **startClientConnection**

```
public abstract boolean startClientConnection()
```

Tries to create a server socket and a task which listens for queries from other agents on a specific port.

Returns:
TRUE if connected sucessfully, FALSE otherwise.

● **stopClientConnection**

```
public void stopClientConnection()
```

Stops the connection to all client agents and terminates the task listening for queries from other agents.

● **updateImageBase**

```
public void updateImageBase()
```

● **unregisterClient**

```
public void unregisterClient(ClientConnection client)
```

Unregisters a client.

Parameters:
client - Client to be unregistered

● **registerQuery**

```
public boolean registerQuery(String query,
                             ClientConnection client,
                             long time)
```

Registers a query. All the queries cached in a client's cache are registered on the server. If a query is invalidated, an invalidation-message is sent to all the clients registered for the query. The Registration is stored in a hashtable. Each entry stores all the clients registered for that query, since more than one client can register for the same query.

Parameters:
query - Query to be registered.
client - Client registering for the query.
time - Processing time.
Returns:
TRUE if registered, FALSE otherwise.

● **unregisterQuery**

```
public void unregisterQuery(String query,
                            ClientConnection client)
```

Unregisters a query. If a query is purged from a clients cache, it will be unregistered from this agent. If a client disconnects, he will unregister all his queries.

Parameters:
query - Query to be unregistered.
client - Client unregistering the query.

● **invalidateCache**

```
public void invalidateCache(String query)
```

This method is called to send an invalidation-message to all the clients which are registered for a specific query. The invalidation-strategy depends on the type of agent which keeps track of the queries. The concrete agents have to overwrite the method to update the invalidated entry.

Parameters:
query - Query to be invalidated.

● **sendRegisterQuery**

```
public void sendRegisterQuery(String query,
                              long time)
```

Registers a query on a server agent.

Parameters:
query - Query to be registered on the server agent.

● **sendUnregisterQuery**

```
public void sendUnregisterQuery(String query)
```

Unregisters a query from a server agent

Parameters:
query - Query to be unregistered from the server agent

● **handleQuery**

```
public boolean handleQuery(Packet query,
                                QueryHandler source)
```

This method handles the queries received from a client agent. Depending on the query-type, the appropriate query handling takes place (Query processing on the server, query registering, query un-registering, error handling)

Parameters:
query - Query to be processed
source - Client which asked for the query to be processed

● **addPackage**

```
public synchronized void addPackage(Packet query,
                                    QueryHandler source)
```

If a query needs to be processed by a server agent, it is send to the server agent. The query is then stored in a queue on this agent where it will remain until the server agent has processed the query.

Parameters:
query - Query to be processed
source - Client which asked for the query to be processed

● **handleError**

```
public void handleError(int error)
```

HandleError should be overwritten. This method will invoke the front-end windows and data visualization in the front-agent.

Parameters:
error - Error received by agent.

● **handleStatusLine**

```
public void handleStatusLine(String text)
```

HandleStatusLine should be overwritten. This method will invoke the front-end windows and data visualization in the front-agent.

Parameters:
error - text received by agent.

● **handleAnswer**

```
public void handleAnswer(Packet query,
                         Packet answer)
```

HandleAnswer should be overriden. This method will invoke the front-end windows and data visualization in the front-agent.

Parameters:
answer - Answer received by agent.

● **updateStatistics**

```
public void updateStatistics(long beginTime,
                             long endTime,
                             boolean cacheHit,
                             boolean query)
```

● **closeWindows**

```
public abstract void closeWindows()
```

Closes all the Windows opened during the session.

● **killAgent**

```
public void killAgent()
```

KillAgent is called if the user clicks on the Exit-button in the interface or if he closes the window. This method writes the parameters, the cache and the log to a file. Then the Application is terminated by 'exit'.

Class agent.ClientConnection

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientConnection
```

---

public class **ClientConnection**
extends Thread

If a new client agent connects to this agent, a new thread is created. This thread - the ClietConnection - will wait for queries sent from the client agent. The ClientConnection initiates the handling of the query and waits for the next query sent by the client agent.

Version:
v1.0
Author:
Philip Ruser, '97

Variables

client

```
public Socket client
```

client_address

```
public InetAddress client_address
```

internetAgent

```
public int internetAgent
```

Constructors

ClientConnection

```
public ClientConnection()
```

ClientConnection

```
public ClientConnection(Agent parent,
                        Socket client_socket)
```

Constructor of ClientConnection

Parameters:
parent - Parent of the ServerListener thread (Agent)
client_socket - Socket on which the communication will take place

Methods

run

```
public void run()
```

Loop "forever" and wait for a query. The handling of the query will be initiated, and the ClientConnection waits for the next query to arrive.

Overrides:
run in class Thread

● **send**

```
public void send(Packet answer)
```

Sends an answer to a query back to the client agent.

● **closeConnection**

```
public void closeConnection()
```

Close the connection to the client agent.

Class agent.ClientListener

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientListener
```

---

public class **ClientListener**
extends Thread

The ClientListener is a task which listens for new client agents connecting to this agent. If a new client agent connects, a new thread is created which handles the client's queries.

Version:
v1.0
Author:
Philip Ruser, '97

Constructors

ClientListener

```
public ClientListener()
```

ClientListener

```
public ClientListener(Agent parent)
```

Constructor of the ClientListener

Parameters:
parent - Parent of the ServerListener thread (Agent)

Methods

run

```
public void run()
```

Loop "forever" to accept new clients, start the client thread and wait for the next client.

Overrides:
run in class Thread

Class agent.DatabaseAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.Agent
                   |
                   +----agent.DatabaseAgent
```

---

public class **DatabaseAgent**
extends Agent

Constructors

DatabaseAgent

```
public DatabaseAgent(boolean showMainWindow,
                     boolean disconnectAllways,
                     String path)
```

Methods

readPreferences

```
public void readPreferences()
```

Reads the preferences from a file.

Overrides:
readPreferences in class Agent

openMainWindow

```
protected void openMainWindow(boolean showMainWindow)
```

Starts up the interface.

Overrides:
openMainWindow in class Agent

closeWindows

```
public void closeWindows()
```

Closes all the Windows opened during the session.

Overrides:
closeWindows in class Agent

updatePreferences

```
public void updatePreferences()
```

Updates the preference variables if the user changed them on the graphical agent window.

Overrides:
updatePreferences in class Agent

startServerConnection

```
public boolean startServerConnection()
```

Tries to connect to another agent which is listening on a specific port. If successful, a server task is created which listens to messages from the server agent.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startServerConnection in class Agent

● **startClientConnection**

```
public boolean startClientConnection()
```

Tries to create a server socket and a task which listens for queries from other agents on a specific port.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startClientConnection in class Agent

● **getServerImage**

```
public synchronized String getServerImage(String destinationFile,
                                          String sourceAddress)
```

● **handleQuery**

```
public boolean handleQuery(Packet query,
                           QueryHandler source)
```

This method handles the queries received from a client agent.

Overrides:
handleQuery in class Agent

● **updateCache**

```
public void updateCache(String query,
                        String answer)
```

● **checkCache**

```
public Packet checkCache(String key)
```

CheckCache checks if a specific query was already processed before. If so, the query and the answer will be in the cache. The cache entry is fetched from the file and is returned immediately. All the cache statistics are then calculated and updated.

Parameters:
query - Query to look for in the cache.
Returns:
Answer to the specified query if this was found in the cache

● **invalidateCache**

```
public void invalidateCache(String query)
```

Updates invalidated queries.

Parameters:
query - Query which has been invalidated.
Overrides:
invalidateCache in class Agent

● **killAgent**

```
public void killAgent()
```

KillAgent is called if the user clicks on the Exit-button in the interface or if he closes the window.

Overrides:
killAgent in class Agent

● **getSessionCache**

```
public FileCache getSessionCache()
```

Class agent.DatabaseAgentClientConnection

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientConnection
                   |
                   +----agent.DatabaseAgentClientConnection
```

public class **DatabaseAgentClientConnection**
extends ClientConnection

Constructors

DatabaseAgentClientConnection

```
 public DatabaseAgentClientConnection(DatabaseAgent parent,
                                      Socket client_socket)
```

Methods

run

```
 public void run()
```

Loop "forever" and wait for a query. The handling of the query will be initiated, and the ClientConnection waits for the next query to arrive.

Overrides:
run in class ClientConnection

Class agent.DatabaseAgentClientListener

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientListener
                   |
                   +----agent.DatabaseAgentClientListener
```

---

public class **DatabaseAgentClientListener**
extends ClientListener

Constructors

DatabaseAgentClientListener

```
 public DatabaseAgentClientListener(DatabaseAgent parent)
```

Methods

run

```
 public void run()
```

Loop "forever" to accept new clients, start the client thread and wait for the next client.

Overrides:
run in class ClientListener

Class agent.DatabaseAgentQueryHandler

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.QueryHandler
                   |
                   +----agent.DatabaseAgentQueryHandler
```

---

public class **DatabaseAgentQueryHandler**
extends QueryHandler

Constructors

DatabaseAgentQueryHandler

```
 public DatabaseAgentQueryHandler(Packet query,
                                  DatabaseAgent parent,
                                  ClientConnection client)
```

Methods

processAnswer

```
 public void processAnswer(Packet answer)
```

Overrides:
processAnswer in class QueryHandler

Class agent.FrontEndAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.Agent
                   |
                   +----agent.FrontEndAgent
```

---

public class **FrontEndAgent**
extends Agent

Variables

parentApplet

```
public Applet parentApplet
```

specificQueries

```
protected Hashtable specificQueries
```

imageAddress

```
public String imageAddress
```

Constructors

FrontEndAgent

```
public FrontEndAgent(String host,
                     int port,
                     boolean showMainWindow,
                     boolean disconnectAllways,
                     boolean middleLayerAgent)
```

Methods

init

```
public void init()
```

Overrides:
init in class Agent

readPreferences

```
public void readPreferences()
```

Reads the preferences from a file.

Overrides:
readPreferences in class Agent

openMainWindow

```
protected void openMainWindow(boolean showMainWindow)
```

Starts up the interface.

Overrides:
openMainWindow in class Agent

● **closeWindows**

```
public void closeWindows()
```

Closes all the Windows opened during the session.

Overrides:
closeWindows in class Agent

● **killAgent**

```
public void killAgent()
```

KillAgent is called if the user clicks on the Exit-button in the interface or if he closes the window.

Overrides:
killAgent in class Agent

● **startServerConnection**

```
public boolean startServerConnection()
```

Tries to connect to another agent which is listening on a specific port. If successful, a server task is created which listens to messages from the server agent.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startServerConnection in class Agent

● **startClientConnection**

```
public boolean startClientConnection()
```

Tries to create a server socket and a task which listens for queries from other agents on a specific port.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startClientConnection in class Agent

● **sendQuery**

```
public void sendQuery(String query)
```

SendQuery is used by the Front-agent to feed a query into the system.

Parameters:
query - Query to send.

● **handleAnswer**

```
public void handleAnswer(Packet queryPacket,
                         Packet answerPacket)
```

HandleAnswer should be overriden.

Overrides:
handleAnswer in class Agent

● **sendSpecificQuery**

```
public void sendSpecificQuery(String query,
                              String specificID)
```

FrontSendSpecificQuery is used by the Front-agent to feed a query into the system, and to specially handle the answer.

Parameters:
query - Query to send.
specificID - ID used to recognize the answer to be handled specially.

● **sendSpecificNoCachingQuery**

```
public void sendSpecificNoCachingQuery(String query,
                                       String specificID)
```

FrontSendSpecificNoCachingQuery is used by the Front-agent to feed a query which shouldn't be cached into the system, and to specially handle the answer.

Parameters:
query - Query to send.
specificID - ID used to recognize the answer to be handled specially.

● **handleSystemViewer**

```
public void handleSystemViewer(String operator,
                               String text)
```

HandleSystemViewer should be overriden. This method will invoke the front-end windows and data visualization in the front-agent.

Parameters:
answer - Answer received by agent.

● **updateCache**

```
public void updateCache(String query,
                        String answer)
```

● **checkCache**

```
public String checkCache(String query)
```

CheckCache checks if a specific query was already processed before. If so, the query and the answer will be in the cache. The cache entry is fetched from the file and is returned immediately. All the cache statistics are then calculated and updated. If a query is found in the cache, the liveTime of the appropriate cache entry is increased. The liveTime of all the other cache entries is decreased. This liveTime-Algorithm is used to avoid queries residing in the cache "forever".

Parameters:
query - Query to look for in the cache.
Returns:
Answer to the specified query if this was found in the cache

● **requestImage**

```
public void requestImage(String name)
```

● **writePreferences**

```
public void writePreferences()
```

Writes the preferences to a file. On the frontend agent nothing is written.

Overrides:
writePreferences in class Agent

● **writeLog**

```
public void writeLog()
```

Writes the log to a file. On the frontend agent nothing is written.

Overrides:
writeLog in class Agent

● **getImage**

```
public Image getImage(String attrVal,
                      Applet parent)
```

● **updatePersonalCacheView**

```
public void updatePersonalCacheView(String data)
```

● **handleSpecificAnswer**

```
public void handleSpecificAnswer(String answer,
                                 String query,
                                 String specificID)
```

● **invalidateCache**

```
public void invalidateCache(String query)
```

Updates the specified query.

Parameters:
query - Query which has been invalidated.
Overrides:
invalidateCache in class Agent

● **getSessionCache**

```
public MemoryCache getSessionCache()
```

● **getSessionImageCache**

```
public MemoryCache getSessionImageCache()
```

Class agent.FrontEndAgentClientConnection

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientConnection
                   |
                   +----agent.FrontEndAgentClientConnection
```

---

public class **FrontEndAgentClientConnection**
extends ClientConnection

Constructors

FrontEndAgentClientConnection

```
 public FrontEndAgentClientConnection(FrontEndAgent parent,
                                      Socket client_socket)
```

Methods

run

```
 public void run()
```

Loop "forever" and wait for a query. The handling of the query will be initiated, and the ClientConnection waits for the next query to arrive.

Overrides:
run in class ClientConnection

Class agent.FrontEndAgentClientListener

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientListener
                   |
                   +----agent.FrontEndAgentClientListener
```

---

public class **FrontEndAgentClientListener**
extends ClientListener

Constructors

FrontEndAgentClientListener

```
 public FrontEndAgentClientListener(FrontEndAgent parent)
```

Methods

run

```
 public void run()
```

Loop "forever" to accept new clients, start the client thread and wait for the next client.

Overrides:
run in class ClientListener

Class agent.FrontEndAgentQueryHandler

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.QueryHandler
                   |
                   +----agent.FrontEndAgentQueryHandler
```

---

public class **FrontEndAgentQueryHandler**
extends QueryHandler

Constructors

FrontEndAgentQueryHandler

```
 public FrontEndAgentQueryHandler(Packet query,
                                  FrontEndAgent parent,
                                  ClientConnection client)
```

Class agent.MiddleLayerAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.Agent
                   |
                   +----agent.MiddleLayerAgent
```

public class **MiddleLayerAgent**
extends Agent

Constructors

MiddleLayerAgent

```
public MiddleLayerAgent(boolean showMainWindow,
                        boolean disconnectAllways)
```

Methods

openMainWindow

```
protected void openMainWindow(boolean showMainWindow)
```

Starts up the interface.

Overrides:
openMainWindow in class Agent

closeWindows

```
public void closeWindows()
```

Closes all the Windows opened during the session.

Overrides:
closeWindows in class Agent

startServerConnection

```
public boolean startServerConnection()
```

Tries to connect to another agent which is listening on a specific port. If successful, a server task is created which listens to messages from the server agent.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startServerConnection in class Agent

startClientConnection

```
public boolean startClientConnection()
```

Tries to create a server socket and a task which listens for queries from other agents on a specific port.

Returns:
TRUE if connected sucessfully, FALSE otherwise.
Overrides:
startClientConnection in class Agent

getImage

```
public InputStream getImage(String requestedImage)
```

● **sendRegistry**

```
protected boolean sendRegistry()
```

Registers the middle layer agent on the server with his port and IP-address.

● **updateCache**

```
public void updateCache(String query,
                        String answer)
```

● **handleFetchAnswer**

```
public void handleFetchAnswer(Packet query,
                              Packet answer,
                              ClientConnection client)
```

● **handlePrefetchAnswer**

```
public void handlePrefetchAnswer(Packet query,
                                 Packet answer)
```

Class agent.MiddleLayerAgentClientConnection

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientConnection
                   |
                   +----agent.MiddleLayerAgentClientConnection
```

---

public class **MiddleLayerAgentClientConnection**
extends ClientConnection

Constructors

MiddleLayerAgentClientConnection

```
 public MiddleLayerAgentClientConnection(MiddleLayerAgent parent,
                                         Socket client_socket)
```

Methods

run

```
 public void run()
```

Loop "forever" and wait for a query. The handling of the query will be initiated, and the ClientConnection waits for the next query to arrive.

Overrides:
run in class ClientConnection

Class agent.MiddleLayerAgentClientListener

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.ClientListener
                   |
                   +----agent.MiddleLayerAgentClientListener
```

---

public class **MiddleLayerAgentClientListener**
extends ClientListener

Constructors

● **MiddleLayerAgentClientListener**

```
public MiddleLayerAgentClientListener(MiddleLayerAgent parent)
```

Methods

● **run**

```
public void run()
```

Loop "forever" to accept new clients, start the client thread and wait for the next client.

Overrides:
run in class ClientListener

Class agent.MiddleLayerAgentQueryHandler

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.QueryHandler
                   |
                   +----agent.MiddleLayerAgentQueryHandler
```

---

public class **MiddleLayerAgentQueryHandler**
extends QueryHandler

Constructors

MiddleLayerAgentQueryHandler

```
 public MiddleLayerAgentQueryHandler(Packet query,
                                     MiddleLayerAgent parent,
                                     ClientConnection client)
```

Methods

processAnswer

```
 public void processAnswer(Packet answer)
```

Overrides:
processAnswer in class QueryHandler

Class agent.Packet

```
java.lang.Object
   |
   +----agent.Packet
```

---

public class **Packet**
extends Object

Unique format for packets sent from one agent to another agent.

Version:
    0.1, Dec 1998
Author:
    Beat Signer

Constructors

Packet

```
 public Packet(long id,
               int type,
               int caching,
               long time,
               String data)
```

Costructs a new packet.

Parameters:
- id - Packet ID.
- type - Type of the packet.
- caching - Caching strategy used by the packet.
- time - Processing time of the packet.
- data - Data field of the packet.

Methods

toPacket

```
 public static Packet toPacket(String packetString)
```

Transforms a string format used to send the packets over the network to the packet format.

Returns:
    Packed parsed from the string.

getID

```
 public long getID()
```

Returns the ID of the packet.

Returns:
    ID of the packet.

getType

```
 public int getType()
```

Returns the type of the packet.

Returns:
    Type of the packet.

● **getCaching**

```
public int getCaching()
```

Returns the caching strategy of the packet.

Returns:
Caching strategy of the packet.

● **getTime**

```
public long getTime()
```

Returns the processing time of the packet.

Returns:
Processing time of the packet.

● **getData**

```
public String getData()
```

Returns the data field of the packet.

Returns:
Data field of the packet.

● **toString**

```
public String toString()
```

Returns a string representation of the packet.

Returns:
String representation of the packet.
Overrides:
toString in class Object

Class agent.PersonalAssistant

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.Agent
                   |
                   +----agent.MiddleLayerAgent
                           |
                           +----agent.PersonalAssistant
```

---

public class **PersonalAssistant**
extends MiddleLayerAgent

Middle Layer component for regular users allowing them to improve response times by using intelligent caching strategies supported by dynamicly generated user profiles. The personal assistant contains three different caches: an LRU session cache for short time caching, a personal cache which can be defined by the user and a long time prefetching cache which dynamically generates a user profile and tries to predict the data the user will need in the near future.

Version:
0.1, Nov 1998
Author:
Beat Signer

Variables

● **prefetchingImageCache**

```
protected FileCache prefetchingImageCache
```

● **prefetchingImageStatistic**

```
protected MemoryCache prefetchingImageStatistic
```

Constructors

● **PersonalAssistant**

```
public PersonalAssistant()
```

Constructs a new personal assistant.

Methods

● **readPreferences**

```
public void readPreferences()
```

Reads the preferences like ports, directories etc.

Overrides:
readPreferences in class Agent

● **openMainWindow**

```
protected void openMainWindow(boolean showMainWindow)
```

Starts up the caching interface.

Parameters:
showMainWindow - TRUE if the caching view should be shown, FALSE otherwise.
Overrides:

openMainWindow in class MiddleLayerAgent

● **handleQuery**

```
public boolean handleQuery(Packet query,
                           QueryHandler source)
```

Handles the queries received from a client

Parameters:
query - Query to be processed.
source - Client asking for the query to be processed.
Overrides:
handleQuery in class Agent

● **updateCache**

```
public void updateCache(String query,
                        String answer)
```

Puts a query result into the short time session cache.

Parameters:
query - Query which should be added to the session cache.
answer - Answer for the corresponding query which should be added to the session cache
Overrides:
updateCache in class MiddleLayerAgent

● **getFromCache**

```
public Packet getFromCache(Packet query)
```

Checks is a specific query is in one of the three caches. If so, the answer will be fetched from the cache.

Parameters:
query - Query to look for in the cache.
Returns:
Answer for the specified query in case of a cache hit, FALSE otherwise.

● **isInCache**

```
public boolean isInCache(String query)
```

Checks if a query is cached.

Parameters:
query - Query the cache has to be checked for.
Returns:
TRUE if the query is in the cache, FALSE otherwise.

● **isInImageCache**

```
public boolean isInImageCache(String query)
```

Checks if an image is cached.

Parameters:
query - Image query the cache has to be checked for.
Returns:
TRUE if the image is in the cache, FALSE otherwise.

● **killAgent**

```
public void killAgent()
```

Stores all parameters before leaving the agent.

Overrides:
killAgent in class Agent

● **handleFetchAnswer**

```
public void handleFetchAnswer(Packet query,
                                       Packet answer,
                                       ClientConnection client)
```

Overrides:
handleFetchAnswer in class MiddleLayerAgent

● **handlePrefetchAnswer**

```
public void handlePrefetchAnswer(Packet query,
                                 Packet answer)
```

Handles an answer to a prefetch which is used for active prefetching of queries.

Parameters:
query - Prefetch query sent to the server.
answer - Answer received from the server.
Overrides:
handlePrefetchAnswer in class MiddleLayerAgent

● **prefetch**

```
public void prefetch(String query)
```

Prefetches query results for a defined query. The number of queries to be prefetched is defined by 'PREFETCH_WIDTH'.

Parameters:
query - Query for which the prefetching has to be done.

● **prefetchImage**

```
public void prefetchImage(String query)
```

Prefetches images for a defined image query. The number of images to be prefetched is defined by 'PREFETCH_WIDTH'.

Parameters:
query - Image for which the prefetching has to be done.

● **invalidateCache**

```
public void invalidateCache(String query)
```

Updates the specified query in the cache.

Parameters:
query - Query which has been invalidated.
Overrides:
invalidateCache in class Agent

● **getSessionCache**

```
public FileCache getSessionCache()
```

Returns the session cache.

Returns:
Session cache.

● **getSessionImageCache**

```
public FileCache getSessionImageCache()
```

Returns the session image cache.

Returns:
Session image cache.

● **getPersonalCache**

```
public FileCache getPersonalCache()
```

Returns the personal cache.

Returns:

Personal cache.

● **getPersonalImageCache**

```
public ReferenceFileCache getPersonalImageCache()
```

Returns the personal image cache.

Returns:
Personal image cache.

● **getPrefetchingCache**

```
public FileCache getPrefetchingCache()
```

Returns the prefetching cache.

Returns:
Prefetching cache.

● **getPrefetchingImageCache**

```
public FileCache getPrefetchingImageCache()
```

Returns the prefetching image cache.

Returns:
Prefetching image cache.

● **getPrefetchImageWidth**

```
public int getPrefetchImageWidth()
```

Returns the prefetching image width indicating how much images have to be prefetched.

Returns:
Number of images to be prefetched.

Class agent.PrefetchImageThread

```
java.lang.Object
   |
   +----agent.PrefetchImageThread
```

---

public class **PrefetchImageThread**
extends Object
implements Runnable

Thread for prefetching of images.

Version:
  0.1, Jan 1999
Author:
  Beat Signer

Constructors

PrefetchImageThread

```
 public PrefetchImageThread(String query,
                            PersonalAssistant parent)
```

Constructs a new prefetching thread.

Parameters:
  query - Query for which the prefetching has to be done.
  parent - Agent which initiated the prefetching.

Methods

run

```
 public void run()
```

Prefetches the images for the specified query and adds them to the prefetching cache.

Class agent.QueryHandler

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----agent.QueryHandler
```

---

public class **QueryHandler**
extends Thread

The Query Handler is a thread which is created to manage a query. If the query arrives from the client, the QueryHandler-thread is created. It then places the query into the agents queue of queries to be processed. After doing this the thread will suspend its execution until the answer arrives from the agent. If the answer is here, the QueryHandler wakes up and returns the answer to the client agent.

Version:
v1.0
Author:
Philip Ruser, '97

Variables

query

```
protected Packet query
```

parent

```
protected Agent parent
```

client

```
public ClientConnection client
```

Constructors

QueryHandler

```
public QueryHandler(Packet query,
                    Agent parent,
                    ClientConnection client)
```

Constructor of the QueryHandler

Parameters:
packet - Query to be processed by the agent
parent - Agent which processes the packet
client - Client which waits for the packet

Methods

run

```
public void run()
```

Execution of the QueryHandler. The QueryHandler places the packet into the agents queue of queries to be processed. After doing this the thread will suspend its execution until the answer arrives from the agent. If the answer is here, the QueryHandler wakes up and returns the answer to the client agent.

Overrides:
run in class Thread

● **answerIsHere**

```
public void answerIsHere(Packet answer)
```

As soon as the answer is returned by the agent, the QueryHandler resumes its execution.

Parameters:
answer - Answer returned by the Agent

● **processAnswer**

```
public void processAnswer(Packet answer)
```

Class agent.RegisteredQuery

```
java.lang.Object
   |
   +----agent.RegisteredQuery
```

---

public class **RegisteredQuery**
extends Object

Stores the clients registered for a query.

Version:
 0.1, Feb 1999
Author:
 Beat Signer

Constructors

RegisteredQuery

```
 public RegisteredQuery()
```

Constructs a new registered query.

RegisteredQuery

```
 public RegisteredQuery(ClientConnection client,
                        long time)
```

Constructs a new registered query.

Parameters:
 client - ClientConnection to be registerd.
 time - Processing time.

Methods

setTime

```
 public synchronized void setTime(long time)
```

Sets the processing time of the query.

Parameters:
 time - New processing time.

getTime

```
 public long getTime()
```

Returns the processing time for the query.

Returns:
 Processing time for the query.

add

```
 public synchronized boolean add(ClientConnection client,
                                 long time)
```

Adds a new client to the query.

Parameters:
 client - ClientConnection to be registered.

time - Processing time.

Returns:

TRUE if client could be added, FALSE otherwise.

● **remove**

```
public synchronized void remove(ClientConnection client)
```

Removes a clients registration for the query.

Parameters:

client - Client to be unregistered.

● **clients**

```
public Enumeration clients()
```

Returns all the clients registered for this query.

Returns:

All the clients registered for this query.

● **size**

```
public int size()
```

Returns the number of clients registered for the query.

Returns:

Number of clients registered for the query.

A.2 The cache Package

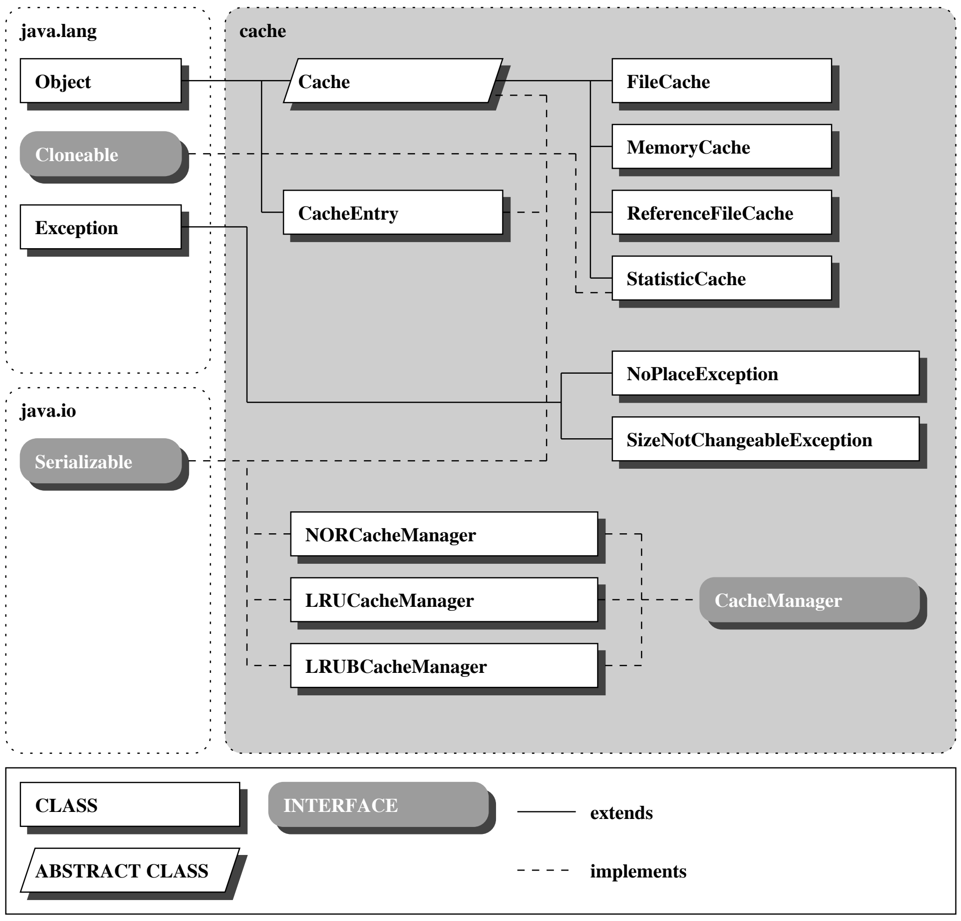


Figure A.2: The cache package

Class cache.Cache

```
java.lang.Object
   |
   +----cache.Cache
```

---

public abstract class **Cache**
extends Object
implements Serializable

Cache allowing to use different caching strategies.

Version:
    0.1, Nov 1998
Author:
    Beat Signer

Variables

currentSize

```
 protected int currentSize
```

maxSize

```
 protected int maxSize
```

Constructors

Cache

```
 public Cache(int size,
              CacheManager cacheManager)
```

Constructs a new cache.

Parameters:
    size - Maximun size of the cache (in kB).
    cacheManager - Cache manager which should be used for this cache.

Methods

getCacheManager

```
 public CacheManager getCacheManager()
```

Returns the used cache manager.

Returns:
    Used cache manager.

containsKey

```
 public abstract boolean containsKey(String key)
```

Checks if an element with a certain key is in the cache.

Parameters:
    key - Key for which the cache has to be checked for.
Returns:
    TRUE, if element with the corresponding key is in the cache, FALSE otherwise.

● **isEmpty**

```
public abstract boolean isEmpty()
```

Checks if the cache is empty.

Returns:
TRUE if the cache is empty, FALSE otherwise.

● **add**

```
public void add(Object data,
                String key) throws NoPlaceException
```

Adds an element to the cache.

Parameters:
data - The data to be cached.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
There is no more place in the cache and no element can be removed.

● **get**

```
public abstract Object get(String key)
```

Returns the element with the corresponding key from the cache. If the element is in the cache, the element is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key if it is in the cache, NULL otherwise.

● **remove**

```
public abstract void remove(String key)
```

Removes the element with the corresponding key from the cache if present.

Parameters:
key - Key of the element which has to be removed.

● **reset**

```
public void reset()
```

Resets the cache, i.e. removes all the elements.

● **getProcessingTime**

```
public Date getProcessingTime(String key)
```

Returns the Time the query in the corresponding cache entry was processed.

Parameters:
key - Key for which the processing time has to be returned.
Returns:
Time the query in the corresponding cache entry was processed.

● **keys**

```
public abstract Enumeration keys()
```

Returns an enumeration of the keys of all the elements currently in the cache.

Returns:
Keys of all the elemets which are currently in the cache.

● **getKeys**

```
public Vector getKeys()
```

Returns a vector containing the keys of all the elements currently in the cache.

Returns:
Vector containg all the elemets currently in the cache.

● **setMaxSize**

```
public void setMaxSize(int maxSize) throws SizeNotChangeableException
```

Changes the maximum space (in kB) the cache is allowed to use. If the new size is smaller than the overall size of the elements currently in the cache, some elements will be removed.

Parameters:
maxSize - Maximum size (in kB) the cache is allowed to use.
Throws: SizeNotChangeableException
Not possible to change the size of the cache.

● **getMaxSize**

```
public int getMaxSize()
```

Returns the maximum space (in Bytes) the cache is allowed to use

Returns:
Maximum size (in Bytes) the cache is allowed to use.

● **setSize**

```
protected void setSize(int size)
```

Updates the current size of the cache.

Parameters:
size - New size of the cache (in Bytes).

● **getSize**

```
public int getSize()
```

Returns the currently used space by the cache.

Returns:
Currently used space by the cache (in Bytes).

● **getCacheEntry**

```
public abstract CacheEntry getCacheEntry(String key)
```

Special method to get an element for a correspondig key. This method doesnt't maintain the whole statistic of hits etc. and is only for internal usage. To get an element from the Cache use 'get()'.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key.

● **addChangeListener**

```
public void addChangeListener(ChangeListener listener)
```

Adds a new ChangeListener to the cache. The listener will be informed every time the data stored in the cache changes.

Parameters:
listener - ChangeListener to be added to the cache.

● **removeChangeListener**

```
public void removeChangeListener(ChangeListener listener)
```

Removes a ChangeListener listening for changes to the data stored in the cache.

Parameters:

listener - ChangeListener to be removed.

● **fireChange**

```
protected void fireChange()
```

Sends a 'ChangeEvent' to all the ChangeListeners listening for changes to the data stored in the cache.

● **getContent**

```
public Vector getContent()
```

Returns additional cache data like 'hitCount' etc. as a vector of cache entry vectors. Usefull for the representation in tables.

Returns:
Vector containing a vector for each cache entry with the following data: key, processingTime, lastHitTime and hitCount.

● **toString**

```
public String toString()
```

Returns a string representation of the class name and relevant attributes.

Returns:
String representation of the class name and relevant attributes.
Overrides:
toString in class Object

Class cache.CacheEntry

```
java.lang.Object
   |
   +----cache.CacheEntry
```

---

public class **CacheEntry**
extends Object
implements Serializable

A single cache entry.

Version:
 0.1, Nov 1998
Author:
 Beat Signer

Methods

getData

```
 public Object getData()
```

Returns the data of the cache entry.

Returns:
 Data of the cache entry.

getKey

```
 public String getKey()
```

Returns the key which is used for this entry in the lookup-table of the cache.

Returns:
 Key for lookup in the cache.

updateLastHit

```
 public void updateLastHit()
```

Sets the date of the lastHit to the current date.

getBirthdate

```
 public Date getBirthdate()
```

Returns the date and time the entry was inserted into the cache.

Returns:
 Date an time the entry was inserted into the cache.

getLastHit

```
 public Date getLastHit()
```

Returns the date and time of the last hit.

Returns:
 Date an time of the last hit.

getProcessingTime

```
 public Date getProcessingTime()
```

Returns the date and time the query was processed.

Returns:
Date and time the query was processed.

● **incHitCount**

```
public void incHitCount()
```

Increases the number of cache hits.

● **getHitCount**

```
public int getHitCount()
```

Returns the number of cache hits.

Returns:
Number of cache hits.

● **setSize**

```
public void setSize(int size)
```

Sets the size for the current representation of the object (in Bytes).

Parameters:
size - Size of the representation of the object.

● **getSize**

```
public int getSize()
```

Returns the size of the current representation of the object (in Bytes).

Returns:
Size of the current representation of the object.

● **toString**

```
public String toString()
```

Returns a string representation of the class name and relevant attributes.

Overrides:
toString in class Object

Interface cache.CacheManager

---

public interface **CacheManager**

Cache manager for a cache, deciding which element should be removed if there is no more place in the cache.

Version:
  0.1, Nov 1998
Author:
  Beat Signer

Methods

getVictim

```
public abstract String getVictim(Cache cache)
```

Returns the element which should be removed.

Parameters:
  cache - The cache an element has to be removed from.
Returns:
  Key of the element to be removed.

getWinner

```
public abstract String getWinner(Cache cache)
```

Returns the element with the best performance which will be the last one to be removed.

Parameters:
  cache - The cache the winner has to be detected from.
Returns:
  Key of the element with the best performance.

Class cache.FileCache

```
java.lang.Object
   |
   +----cache.Cache
           |
           +----cache.FileCache
```

---

public class **FileCache**
extends Cache

Cache which writes cached elements on a disk allowing to use different caching strategies by choosing a cache manager.

Version:
0.1, Nov 1998
Author:
Beat Signer

Constructors

● **FileCache**

```
 public FileCache(int size,
                  CacheManager cacheManager,
                  String cacheDir)
```

Constructs a new cache.

Parameters:
size - Maximun size of the cache (in kB).
cacheManager - Cache manager which should be used for the cache.
cacheDir - Location where the cache should store it's files.

Methods

● **load**

```
 public static FileCache load(String cacheDir) throws IOException, ClassNotFoundException
```

Restores the cache.

Parameters:
cacheDir - Location of the cache index file.
Returns:
FileCache read from the file. NULL if file doesn't exist.
Throws: ClassNotFoundException
Class of a serialized object cannot be found.
Throws: IOException
Any of the usual Input/Output related exceptions.

● **store**

```
 public void store()
```

Stores the cache to a file.

● **add**

```
 public void add(Object data,
                 String key) throws NoPlaceException
```

Adds an element to the cache. The element is stored in a file and only the name of the file is added to the cache index. For unique filenames, the time in milliseconds since midnight GMT on January 1, 1970 is used. So there will be a naming problem in the year 292473178 due to an overflow of the LONG variable which can be simply resolved by resetting the cache :-)

Parameters:
data - Data which should be cached.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
There is no more place in the cache and no element can be removed.
Overrides:
add in class Cache

● **add**

```
public void add(InputStream data,
                String key,
                String extension) throws NoPlaceException
```

Adds an element to the cache. The element is stored in a file and only the name of the file is added to the cache index. For unique filenames, the time in milliseconds since midnight GMT on January 1, 1970 is used. So there will be a naming problem in the year 292473178 due to an overflow of the LONG variable which can be simply resolved by resetting the cache :-)

Parameters:
data - Data which should be cached.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
There is no more place in the cache and no element can be removed.

● **get**

```
public Object get(String key)
```

Returns the element with the corresponding key from the cache. If the element is in the cache, the element is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key if it is in the cache, NULL otherwise.
Overrides:
get in class Cache

● **getFileName**

```
public String getFileName(String key)
```

Returns the filename of the cache entry for a corresponding key. If the element is in the cache, the filename is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Filename of the cache entry for the corresponding key if it is in the cache, NULL otherwise.

● **remove**

```
public void remove(String key)
```

Removes the element with the corresponding key from the cache if present.

Parameters:
key - Key of the element which has to be removed.
Overrides:
remove in class Cache

● **containsKey**

```
public boolean containsKey(String key)
```

Checks if an element with a certain key is in the cache.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
TRUE, if element with the corresponding key is in the cache, FALSE otherwise.

Overrides:
containsKey in class Cache

● **isEmpty**

```
public boolean isEmpty()
```

Checks if the cache is empty.

Returns:
TRUE if the cache is empty, FALSE otherwise.
Overrides:
isEmpty in class Cache

● **keys**

```
public Enumeration keys()
```

Returns an enumeration of the keys of all the elements which are currently in the cache.

Returns:
Keys of all the elemets which are currently in the cache.
Overrides:
keys in class Cache

● **getCacheEntry**

```
public CacheEntry getCacheEntry(String key)
```

Special method to get an element for a correspondig key. This method doesnt't maintain the whole statistic of hits etc. and is only for internal usage! To get an element from the Cache use 'get()'.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key.
Overrides:
getCacheEntry in class Cache

● **toString**

```
public String toString()
```

Returns a string representation of the class name and relevant attributes.

Returns:
String representation of the class name and relevant attributes.
Overrides:
toString in class Cache

Class cache.LRUBCacheManager

```
java.lang.Object
   |
   +----cache.LRUBCacheManager
```

---

public class **LRUBCacheManager**
extends Object
implements CacheManager, Serializable

Cache manager which uses an LRU strategy but gives a bonus to elements with a high 'hitCount'. The weight of an entry is defined as follows: weight = alpha*(1/log(time since last access)) + (1-alpha)*log(number of hits)

Version:
    0.1, Jan 1999
Author:
    Beat Signer

Constructors

LRUBCacheManager

```
public LRUBCacheManager()
```

Methods

getVictim

```
public String getVictim(Cache cache)
```

Returns the element with the lowest value for the weighting function.

Parameters:
    cache - The cache an element has to be removed from.
Returns:
    Key of the element to be removed.

getWinner

```
public String getWinner(Cache cache)
```

Returns the element with the best condition which will be the last one removed from the cache.

Parameters:
    cache - The cache the winner has to be detected from.
Returns:
    Key of the element with the best condition.

Class cache.LRUCacheManager

```
java.lang.Object
   |
   +----cache.LRUCacheManager
```

---

public class **LRUCacheManager**
extends Object
implements CacheManager, Serializable

Cache manager which removes the element which was least recently used.

Version:
 0.1, Nov 1998
Author:
 Beat Signer

Constructors

● **LRUCacheManager**

```
public LRUCacheManager()
```

Methods

● **getVictim**

```
public String getVictim(Cache cache)
```

Returns the element which was least recently used.

Parameters:
 cache - The cache an element has to be removed from.
Returns:
 Key of the element to be removed.

● **getWinner**

```
public String getWinner(Cache cache)
```

Returns the element with the best condition which will be the last one removed from the cache.

Parameters:
 cache - The cache the winner has to be detected from.
Returns:
 Key of the element with the best condition.

Class cache.MemoryCache

```
java.lang.Object
   |
   +----cache.Cache
           |
           +----cache.MemoryCache
```

---

public class **MemoryCache**
extends Cache

Non persistent memory cache allowing to use different caching strategies by choosing a cache manager.

Version:
0.1, Dec 1998
Author:
Beat Signer

Variables

cache

```
 protected Hashtable cache
```

Constructors

MemoryCache

```
 public MemoryCache(int size,
                    CacheManager cacheManager)
```

Constructs a new cache.

Parameters:
size - Maximun size of the cache (in kB).
cacheManager - Cache manager which should be used for this cache.

Methods

add

```
 public void add(Object data,
                 String key,
                 int size) throws NoPlaceException
```

Adds an element to the cache.

Parameters:
data - Data which should be cached.
key - Unique key for the storage in the cache.
size - Because in Java it isn't possible to get the size of an object, the user has to explicitly specify the size of the object to be added.
Throws: NoPlaceException
Exception is throwed if there is no more place in the cache and no element can be removed.

get

```
 public Object get(String key)
```

Returns the element with the corresponding key from the cache. If the element is in the cache, the element is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key if it is in the cache, NULL otherwise.
Overrides:
get in class Cache

● **remove**

```
public void remove(String key)
```

Removes the element with the corresponding key from the cache if present.

Parameters:
key - Key of the element which has to be removed.
Overrides:
remove in class Cache

● **containsKey**

```
public boolean containsKey(String key)
```

Checks if an element with a certain key is in the cache.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
TRUE, if element with the corresponding key is in the cache, FALSE otherwise.
Overrides:
containsKey in class Cache

● **isEmpty**

```
public boolean isEmpty()
```

Checks if the cache is empty.

Returns:
TRUE if the cache is empty, FALSE otherwise.
Overrides:
isEmpty in class Cache

● **keys**

```
public Enumeration keys()
```

Returns an enumeration of the keys of all the elements which are currently in the cache.

Returns:
Keys of all the elemets which are currently in the cache.
Overrides:
keys in class Cache

● **getCacheEntry**

```
public CacheEntry getCacheEntry(String key)
```

Special method to get an element for a correspondig key. This method doesn't maintain the whole statistic of hits etc. and is only for internal usage! To get an element from the Cache use 'get()'.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key.
Overrides:
getCacheEntry in class Cache

● **toString**

```
public String toString()
```

Returns a string representation of the class name and relevant attributes.

Returns:
String representation of the class name and relevant attributes.
Overrides:
toString in class Cache

Class cache.NoPlaceException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----cache.NoPlaceException
```

---

public class **NoPlaceException**
extends Exception

Exception which is raised if there is no more place in the cache and an element can't be added because the cache manager doesn't return a victim.

Version:
0.1, Dec 1998
Author:
Beat Signer

Constructors

NoPlaceException

```
 public NoPlaceException()
```

Constructs a new Exception.

NoPlaceException

```
 public NoPlaceException(String text)
```

Constructs a new Exception with additional information.

Parameters:
text - Additional information.

Class cache.NORCacheManager

```
java.lang.Object
   |
   +----cache.NORCacheManager
```

---

public class **NORCacheManager**
extends Object
implements CacheManager, Serializable

Cache manager which removes no element from the cache.

Version:
 0.1, Dec 1998
Author:
 Beat Signer

Constructors

NORCacheManager

```
public NORCacheManager()
```

Methods

getVictim

```
public String getVictim(Cache cache)
```

Returns allways NULL and no element will be removed.

Parameters:
 cache - The cache an element has to be removed from.
Returns:
 Allways NULL.

getWinner

```
public String getWinner(Cache cache)
```

Returns allways NULL.

Parameters:
 cache - The cache the winner has to be detected from.
Returns:
 Allways NULL.

Class cache.ReferenceFileCache

```
java.lang.Object
   |
   +----cache.Cache
           |
           +----cache.ReferenceFileCache
```

---

public class **ReferenceFileCache**
extends Cache

Cache which writes cached elements on a disk and allows to use different caching strategies by choosing a cache manager. For each entry there exists a list containing the queries the element is used by (Reference counting). An element will only be deleted if there are no more references to the element.

Version:
  0.1, Jan 1999
Author:
  Beat Signer

Constructors

ReferenceFileCache

```
public ReferenceFileCache(int size,
                          CacheManager cacheManager,
                          String cacheDir)
```

Constructs a new cache.

Parameters:
  size - Maximun size of the cache (in kB).
  cacheManager - Cache manager which should be used for this cache.
  cachDir - Location where the cache should store it's files.

Methods

load

```
public static ReferenceFileCache load(String cacheDir) throws IOException, ClassNotFoundExcept
```

Restore the cache.

Parameters:
  cacheDir - Location of the cache index file.
Returns:
  'Cache' Object read from the file. NULL if file doesn't exist.
Throws: ClassNotFoundException
  Class of a serialized object cannot be found.
Throws: IOException
  Any of the usual Input/Output related exceptions.

store

```
public void store()
```

Stores the cache to a file.

add

```
public void add(Object data,
                String key,
                String reference) throws NoPlaceException
```

Adds an element to the cache. If the element is already in the cache, a new reference to the element is stored in the 'referenceTable'.

Parameters:
data - Data which should be added to the cache.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
There is no more place in the cache and no element can be removed.

● **add**

```
public void add(Object data,
                String key) throws NoPlaceException
```

Adds an element to the cache. The element is stored in a file and only the name of the file is added to the cache index. For unique filenames, the time in milliseconds since midnight GMT on January 1, 1970 is used. So there will be a naming problem in the year 292473178 due to an overflow of the LONG variable which can be simply resolved by resetting the cache :)

Parameters:
data - Data which should be cached.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
Exception is throwed if there is no more place in the cache and no element can be removed.
Overrides:
add in class Cache

● **add**

```
public void add(InputStream data,
                String key,
                String extension,
                String reference) throws NoPlaceException
```

Adds an element to the cache. If the element is already in the cache, a new reference to the element is stored in the 'referenceTable'.

Parameters:
data - Data which should be added to the cache.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
Exception is throwed if there is no more place in the cache and no element can be removed.

● **add**

```
public void add(InputStream data,
                String key,
                String extension) throws NoPlaceException
```

Adds an element to the cache. The element is stored in a file and only the name of the file is added to the cache index. For unique filenames, the time in milliseconds since midnight GMT on January 1, 1970 is used. So there will be a naming problem in the year 292473178 due to an overflow of the LONG variable which can be simply resolved by resetting the cache :-)

Parameters:
data - Data which should be cached.
key - Unique key for the storage in the cache.
Throws: NoPlaceException
Exception is throwed if there is no more place in the cache and no element can be removed.

● **get**

```
public Object get(String key)
```

Returns the element with the corresponding key from the cache. If the element is in the cache, the element is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Element for the corresponding key if it is in the cache, NULL otherwise.
Overrides:
get in class Cache

● **getFileName**

```
public String getFileName(String key)
```

Returns the filename of the cache entry for a corresponding key. If the element is in the cache, the filename is returned and 'hitCount' respective 'lastHit' are updated.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
Filename of the cache entry for the corresponding key if it is in the cache, NULL otherwise.

● **remove**

```
public void remove(String key,
                   String reference)
```

Removes the element with the corresponding key from the cache if present. The element is only removed, if there are no more references to the element!

Parameters:
key - Key of the element which has to be removed.

● **remove**

```
public void remove(String key)
```

Removes the element with the corresponding key from the cache if present.

Parameters:
key - Key of the element which has to be removed.
Overrides:
remove in class Cache

● **containsKey**

```
public boolean containsKey(String key)
```

Checks if an element with a certain key is in the cache.

Parameters:
key - Key for which the cache has to be checked for.
Returns:
TRUE if element with the corresponding key is in the cache, FALSE otherwise.
Overrides:
containsKey in class Cache

● **isEmpty**

```
public boolean isEmpty()
```

Checks if the cache is empty.

Returns:
TRUE if the cache is empty, FALSE otherwise.
Overrides:
isEmpty in class Cache

● **keys**

```
public Enumeration keys()
```

Returns an enumeration of the keys of all the elements which are currently in the cache.

Returns:
Keys of all the elemets which are currently in the cache.
Overrides:
keys in class Cache

● **getCacheEntry**

```
public CacheEntry getCacheEntry(String key)
```

Special method to get an element for a correspondig key. This method doesnt't maintain the whole statistic of hits etc. and is only for internal usage! To get an element from the Cache use 'get()'.

Parameters:
key - Key for which the cache has to be checked for.

Returns:
Element for the corresponding key.

Overrides:
getCacheEntry in class Cache

● **toString**

```
public String toString()
```

Returns a string representation of the class name and relevant attributes.

Returns:
String representation of the class name and relevant attributes.

Overrides:
toString in class Cache

Class cache.SizeNotChangeableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----cache.SizeNotChangeableException
```

---

public class **SizeNotChangeableException**
extends Exception

Exception which is raised if the cache size is changed to a size smaller than the size of all the entries but the cache manager doesn't allow to remove any element at all.

Version:
    0.1, Dec 1998
Author:
    Beat Signer

Constructors

SizeNotChangeableException

```
public SizeNotChangeableException()
```

Constructs a new exception.

SizeNotChangeableException

```
public SizeNotChangeableException(String text)
```

Constructs a new exception with detailed information.

Parameters:
    text - Detailed information.

Class cache.StatisticCache

```
java.lang.Object
   |
   +----cache.Cache
           |
           +----cache.MemoryCache
                   |
                   +----cache.StatisticCache
```

public class **StatisticCache**
extends MemoryCache
implements Cloneable

Special Memory cache allowing to get the n winner elements.

Version:
0.1, Jan 1999

Author:
Beat Signer

Constructors

● **StatisticCache**

```
 public StatisticCache(int size,
                       CacheManager cacheManager)
```

Constructs a new Statistic Cache

Parameters:
size - Maximun size of the cache (in kB).
cacheManager - Cache manager which should be used for the cache.

Methods

● **clone**

```
 public Object clone()
```

Supports the cloning of a statistic cache.

Overrides:
clone in class Object

● **getWinner**

```
 public Enumeration getWinner(int n)
```

Return the n cache entries with the best condition.

Parameters:
n - Number of winners to be returned.

Returns:
n cache entries with the best condition.

A.3 The tools Package

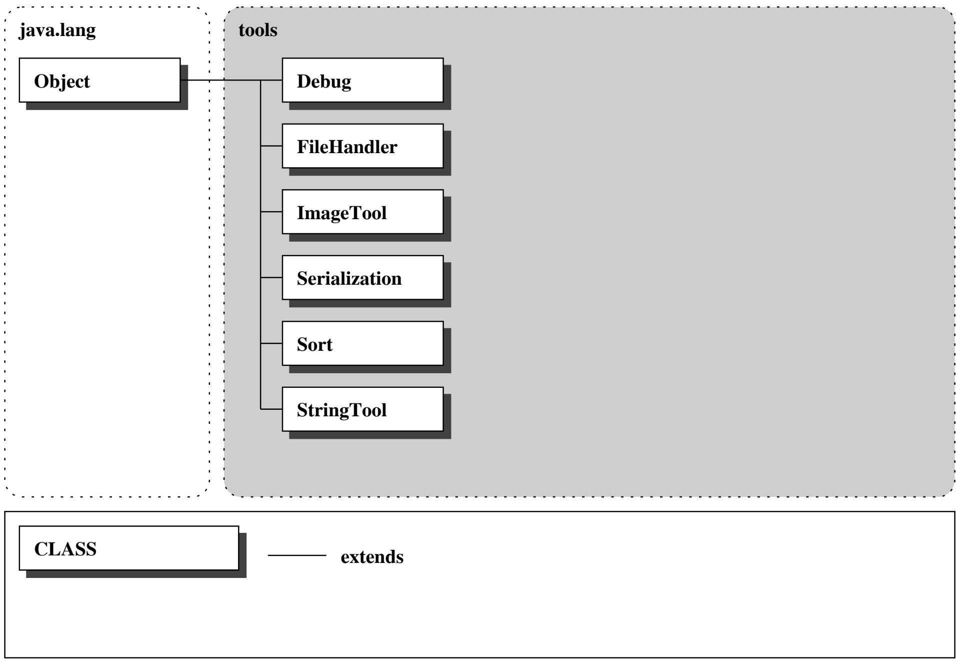


Figure A.3: The tools package

Class tools.Debug

```
java.lang.Object
   |
   +----tools.Debug
```

---

public class **Debug**
extends Object

Tool for performing additional outputs. A good compiler will eliminate the method calls if the corresponding constant is set false.

Version:
    0.1, Feb 1999
Author:
    Beat Signer

Constructors

Debug

```
 public Debug()
```

Methods

printMethod

```
 public static final void printMethod(String output)
```

Debugging output of method calls if METHOD_DEBUGGING is TRUE.

Parameters:
    output - String to be printed.

printException

```
 public static final void printException(String output)
```

Debugging output of exceptions if EXCEPTION_DEBUGGING is TRUE.

Parameters:
    output - String to be printed.

printInformation

```
 public static final void printInformation(String output)
```

Output of additional information if INFORMATION_DEBUGGING is TRUE.

Parameters:
    output - String to be printed.

printSecurity

```
 public static final void printSecurity(String output)
```

Output of security information if SECURITY_DEBUGGING is TRUE.

Parameters:
    output - String to be printed.

Class tools.FileHandler

```
java.lang.Object
   |
   +----tools.FileHandler
```

---

public class **FileHandler**
extends Object

Tools for file handling.

Version:
    0.1, Dec 1998
Author:
    Beat Signer

Variables

BUFFER_SIZE

```
 public static final int BUFFER_SIZE
```

Constructors

FileHandler

```
 public FileHandler()
```

Methods

readFile

```
 public static String readFile(String path,
                               String filename) throws OptionalDataException, IOException, Clas
```

Reads a String from the specified file from the specified location.

Parameters:
    path - Location of the file containing the string.
    filename - Name of the file containing the string.
Returns:
    String read from the file.
Throws: IOException
    Any of the usual Input/Output related exceptions.
Throws: OptionalDataException
    Primitive data was found in the stream instead of objects.
Throws: ClassNotFoundException
    Class of a serialized object cannot be found.

writeFile

```
 public static void writeFile(String path,
                              String filename,
                              String data) throws IOException
```

Writes a string to a file at the specified location.

Parameters:
    path - Location the file has to be written at.
    filename - Name of the file the string has to be written in.
    data - String to be written in a file.
Throws: IOException

Any of the usual Input/Output related exceptions.

● **deleteFile**

```
public static void deleteFile(String path,
                              String filename)
```

Deletes a file with the specified path and name.

Parameters:
path - Location of the file to be deleted.
filename - Name of the file to be deleted.

● **readObject**

```
public static Object readObject(String path,
                                String filename) throws OptionalDataException, IOException, Cl
```

Reads an object with the specified filename from specified location.

Parameters:
path - Location the file has to be read from.
filename - Name of the file to be read.
Returns:
Object read from the file.
Throws: IOException
Any of the usual Input/Output related exceptions.
Throws: OptionalDataException
Primitive data was found in the stream instead of objects.
Throws: ClassNotFoundException
Class of a serialized object cannot be found.

● **writeObject**

```
public static void writeObject(Object object,
                               String path,
                               String name) throws IOException
```

Writes an object to a file with the specified filename and path.

Parameters:
object - Object to be written to a file.
path - Path the file has to be written at.
name - Name of the file the Object has to be written in.
Throws: IOException
Any of the usual Input/Output related exceptions.

Class tools.ImageTool

```
java.lang.Object
   |
   +----tools.ImageTool
```

---

public class **ImageTool**
extends Object

Tools for image processing.

Version:
0.1, Jan 1999

Author:
Beat Signer

Variables

PIXEL_SIZE

```
 public static int PIXEL_SIZE
```

Constructors

ImageTool

```
 public ImageTool()
```

Methods

sizeOf

```
 public static int sizeOf(Image image)
```

Computes the size of an image.

Parameters:
image - Image the size has to be computed for.

Class tools.Serialization

```
java.lang.Object
   |
   +----tools.Serialization
```

---

public class **Serialization**
extends Object

Tools for serialisation.

Version:
0.1, Dec 1998

Author:
Beat Signer

Constructors

Serialization

```
 public Serialization()
```

Methods

toString

```
 public static final String toString(Serializable object)
```

Serializes an Object to a string.

Parameters:
object - Object to be serialized.

Returns:
String containing the serialized object.

toObject

```
 public static final Object toObject(String string) throws IOException, ClassNotFoundException
```

Deserializes an Object from a String.

Parameters:
string - String the object has to be build from.

Returns:
Object deserialized from the string.

Throws: ClassNotFoundException
Class of a serialized object cannot be found.

Throws: IOException
Any of the usual Input/Output related exceptions.

vectorToString

```
 public static final String vectorToString(Vector data,
                                           String delimiter)
```

Writes a vector to a string.

Parameters:
data - Vector to be written to a string.
delimiter - Used for the separation of two elements.

Returns:
String containing the elements of the vector.

Class tools.Sort

```
java.lang.Object
   |
   +----tools.Sort
```

---

public class **Sort**
extends Object

Tool to sort a vector of strings.

Version:
0.1, Dec 1998
Author:
Beat Signer

Constructors

Sort

```
public Sort()
```

Methods

processVector

```
public static final Vector processVector(Vector unsorted)
```

Sorts the elements of a vector. Uses a simple selection sort.

Parameters:
unsorted - Vector to be sorted.
Returns:
Vector containing the sorted elements.

Class tools.StringTool

```
java.lang.Object
   |
   +----tools.StringTool
```

---

public class **StringTool**
extends Object

Tool for string manipulation.

Version:
0.1, Feb 1999
Author:
Beat Signer

Constructors

StringTool

```
 public StringTool()
```

Methods

shorten

```
 public static final String shorten(String string,
                                    int length)
```

Shortens a string to a specified length. If the string is smaller than the desired length, the string is returned without any manipulation.

Parameters:
string - string to be shorten.
length - Desired size of the string.
String - of the size less or equal to 'length'.

A.4 The views Package

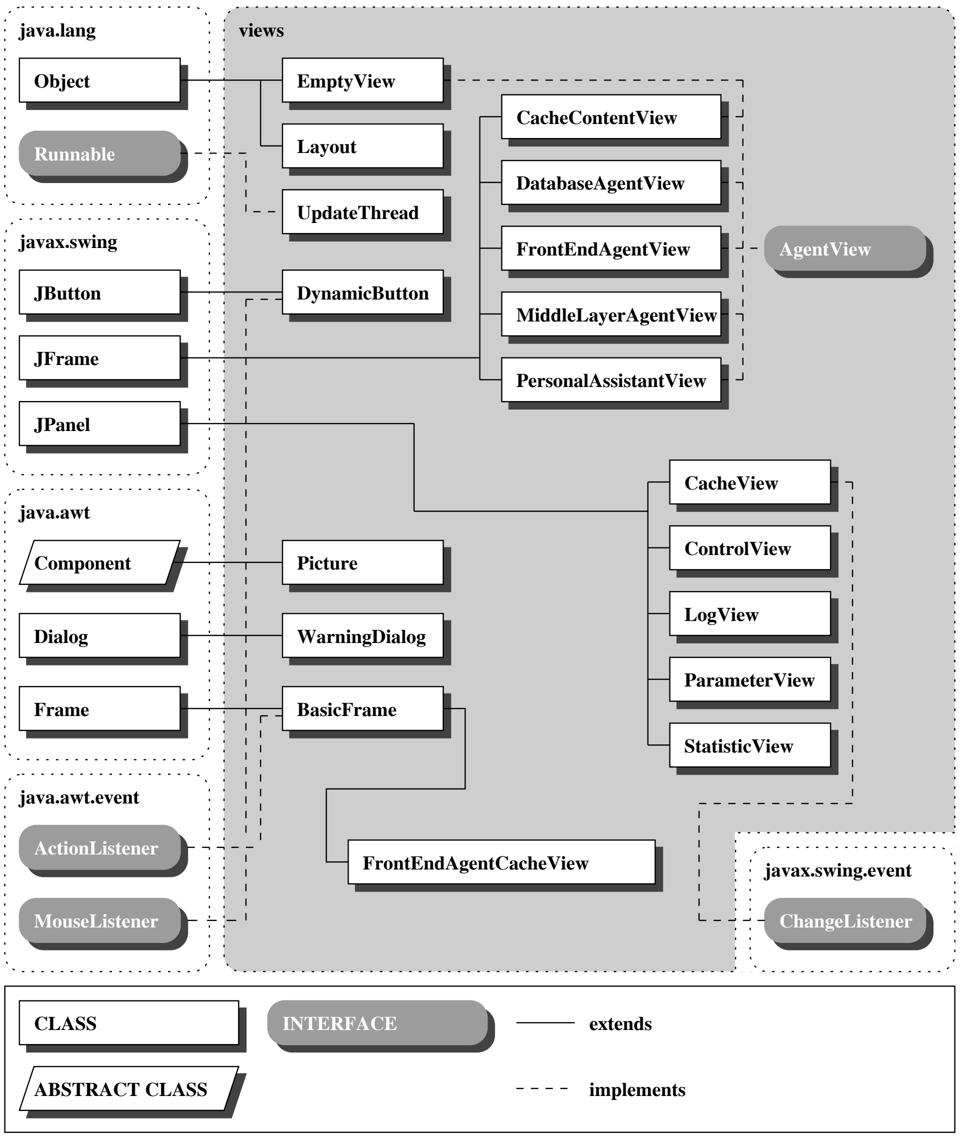


Figure A.4: The views package

Interface views.AgentView

public interface **AgentView**

Interface for all agent views.

Version:
  0.1, Feb 1999
Author:
  Beat Signer

Methods

getCaller

```
public abstract Agent getCaller()
```

Returns the agent which generated the view.

Returns:
  Agent which generated the view.

setLog

```
public abstract void setLog(String text)
```

Sets the text in the log view.

Parameters:
  text - Text to be set in the log view.

addToLog

```
public abstract void addToLog(String text)
```

Adds the message to the log view.

Parameters:
  text - Message to be added to the log view.

getLog

```
public abstract String getLog()
```

Parses the log from the log view.

Returns:
  Log parsed from the log view.

setListenPort

```
public abstract void setListenPort(String port)
```

Sets the listen port view.

Parameters:
  port - Value to be shown in the listen port view.

getListenPort

```
public abstract int getListenPort()
```

Parses the listen port from the listen port view.

Returns:
  Listen port parsed from the listen port view.

● **setServerHost**

```
public abstract void setServerHost(String host)
```

Sets the server host view.

Parameters:
host - Value to be shown in the server host view.

● **getServerHost**

```
public abstract String getServerHost()
```

Parses the server host from the server host view.

Returns:
Server host parsed from the server port view.

● **setServerPort**

```
public abstract void setServerPort(String port)
```

Sets the server port view.

Parameters:
port - Value to be shown in the server port view.

● **getServerPort**

```
public abstract int getServerPort()
```

Parses the server port from the server port view.

Returns:
Server port parsed from the server port view.

● **setImageBase**

```
public abstract void setImageBase(String base)
```

Sets the image base view.

Parameters:
base - Value to be shown in the image base view.

● **getImageBase**

```
public abstract String getImageBase()
```

Parses the image base from the image base view.

Returns:
Image base parsed from the image base view.

● **setHelpURL**

```
public abstract void setHelpURL(String url)
```

Sets the helpURL view.

Parameters:
url - Value to be shown in the helpURL view.

● **getHelpURL**

```
public abstract String getHelpURL()
```

Parses the helpURL from the helpURL view.

Returns:
HelpURL parsed from the helpURL view.

● **setClientCount**

```
public abstract void setClientCount(String count)
```

Sets the client count.

Parameters:
count - Value to be shown in the client count view.

● **setStatus**

```
public abstract void setStatus(String status)
```

Sets the status message.

Parameters:
status - Message to be shown in the status line.

● **setQueryCount**

```
public abstract void setQueryCount(String count)
```

Sets the query count.

Parameters:
count - Value to be shown in the query count view.

● **setCacheHit**

```
public abstract void setCacheHit(String hitRate)
```

Sets the cache hit rate.

Parameters:
hitRate - Value to be shown in the hit rate view.

● **setMeanQueryTime**

```
public abstract void setMeanQueryTime(String queryTime)
```

Sets the mean query time.

Parameters:
queryTime - Value to be shown in the mean query time view.

● **setQueryTime**

```
public abstract void setQueryTime(String queryTime)
```

Sets the query time.

Parameters:
queryTime - Value to be shown in the query time view.

● **setImageQueryTime**

```
public abstract void setImageQueryTime(String queryTime)
```

Sets the image query time which is the time needed to return the address of the image and NOT the time to transfer the image itself!.

Parameters:
queryTime - Value to be shown in the image query time view.

● **startClicked**

```
public abstract void startClicked()
```

Handles a pressed start button.

● **stopClicked**

```
public abstract void stopClicked()
```

Handles a pressed stop button.

● **setVisible**

```
public abstract void setVisible(boolean visible)
```

Sets the visible state of the agent view.

Parameters:
vivible - TRUE if the agent view should be shown, FALSE otherwise.

Class views.BasicFrame

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----views.BasicFrame
```

---

public class **BasicFrame**
extends Frame
implements ActionListener

Basic frame which allows to close the frame, add componenets to a container and generate new buttons with an optional action listener.

Version:
: 0.1, Dec 1998

Author:
: Beat Signer

Constructors

BasicFrame

```
public BasicFrame()
```

Constructs new BasicFrame with the name of the class as default title.

Methods

close

```
public void close()
```

Closes the frame.

getButton

```
public Button getButton(String name,
                        boolean action)
```

Generates an new button with a defined name and an optional action listener.

Parameters:
: name - Name to be displayed on the Button.
: action - TRUE if the user wishes an action listener to be added to the button.

Returns:
: New Button with an optional actionListener.

addComponent

```
public static void addComponent(Container container,
                                Component component,
                                int top,
                                int left,
                                int bottom,
                                int right)
```

Adds a component to a container with a GridBagLayout.

Parameters:
container - The container the component should be added.
component - The Component to be added.
top - Border at the top.
left - Border at the left.
bottom - Border at the bottom.
right - Border at the right

See Also:
GridBagConstraints

● **addComponent**

```
public static void addComponent(Container container,
                                Component component,
                                int gridx,
                                int gridy,
                                int gridwidth,
                                int gridheight,
                                int top,
                                int left,
                                int bottom,
                                int right,
                                int anchor,
                                int fill)
```

Adds a component to a container with a GridBagLayout.

Parameters:
container - The container the component should be added.
component - The Component to be added.
gridx - Speciefies the cell at the left of the component's display area, where the leftmost cell has 'gridx = 0'.
gridy - Specifies teh cell at the top of the component's display area, where rhe topmost cell has 'gridy =0'.
gridwidth - Specifies the number of cells in a row for the component's display area.
gridheight - Specifies the number of cells in a colum for the component's display area.
top - Border at the top.
left - Border at the left.
bottom - Border at the bottom.
right - Border at the right
anchor - Used when the component is smaller than its display area. It determines where to place the component.
fill - Used when the component's display area is larger than the componenet's requested size. It determines wheter to resize the component, and if so, how.

See Also:
GridBagConstraints

● **actionPerformed**

```
public void actionPerformed(ActionEvent event)
```

Invoked if an action occurs.

Parameters:
event - Generated action event.

Class views.CacheContentView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----javax.swing.JFrame
                                           |
                                           +----views.CacheContentView
```

public class **CacheContentView**
extends JFrame

View showing teh following cache contents: key, processing time, last hit time and hit count.

Version:
 0.1, Feb 1999
Author:
 Beat Signer

Constructors

CacheContentView

```
 public CacheContentView(Cache cache,
                         String title)
```

Constructs a new cache view.

Parameters:
 cache - The modell containig the data.

Methods

update

```
 public synchronized void update()
```

Updates the cache view.

Class views.CacheView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----views.CacheView
```

---

public class **CacheView**
extends JPanel
implements ChangeListener

View showing relevant cache parameters.

Version:
0.1, Jan 1999
Author:
Beat Signer

Constructors

CacheView

```
 public CacheView(Cache cache,
                  String title,
                  boolean imageButton)
```

Constructs a new cache view.

Parameters:
cache - The modell containing the data.
title - Title for the cache view.
imageButton - TRUE if images should be shown on the buttons, FALSE otherwise.

Methods

updateProgress

```
 public void updateProgress(int newSize)
```

Updates the progress bar.

Parameters:
newSize - Value to be shown in the progress bar.

updateSize

```
 public void updateSize(String newSize)
```

Updates the cache size view.

Parameters:
newSize - Value to be shown i the cache size view.

stateChanged

```
 public void stateChanged(ChangeEvent event)
```

Invoked if the data in the modell changed. A new thread is generated which will update the view.

Parameters:
event - Change event invokin the method.

● **update**

```
public void update()
```

Updates the whole cache view.

Class views.ControlView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----views.ControlView
```

---

public class **ControlView**
extends JPanel

Button line containing a status line and buttons to control the agent view.

Version:
  0.1, Jan 1999
Author:
  Beat Signer

Constructors

● **ControlView**

```
 public ControlView(AgentView parent,
                    boolean imageButton)
```

Constructs a new control view.

Parameters:
  imageButton - TRUE if images should be shown on the buttons, FALSE otherwise.

Methods

● **setStart**

```
 public void setStart()
```

Disables the start button and enables the stop button.

● **setStop**

```
 public void setStop()
```

Disables the stop button and enables the start button.

● **setStatus**

```
 public void setStatus(String status)
```

Shows a message in the status line.

Parameters:
  status - Message to be shown in the status line.

Class views.DatabaseAgentView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----javax.swing.JFrame
                                           |
                                           +----views.DatabaseAgentView
```

---

public class **DatabaseAgentView**
extends JFrame
implements AgentView

Database agent view showing the caches and other relevant parameters.

Version:
  0.1, Feb 1999
Author:
  Beat Signer

Constructors

DatabaseAgentView

```
public DatabaseAgentView(DatabaseAgent caller)
```

Constructs a new database agent view.

Parameters:
  caller - Database agent generating the view.

Methods

getCaller

```
public Agent getCaller()
```

Returns the agent which generated the view.

Returns:
  Agent which generated the view.

setLog

```
public void setLog(String text)
```

Sets the text in the log view.

Parameters:
  text - Text to be set in the log view.

addToLog

```
public void addToLog(String text)
```

Adds the message to the log view.

Parameters:
text - Message to be added to the log view.

● **getLog**

```
public String getLog()
```

Parses the log from the log view.

Returns:
Log parsed from the log view.

● **setListenPort**

```
public void setListenPort(String port)
```

Sets the listen port view.

Parameters:
port - Value to be shown in the listen port view.

● **getListenPort**

```
public int getListenPort()
```

Parses the listen port from the listen port view.

Returns:
Listen port parsed from the listen port view.

● **setServerHost**

```
public void setServerHost(String host)
```

Sets the server host view.

Parameters:
host - Value to be shown in the server host view.

● **getServerHost**

```
public String getServerHost()
```

Parses the server host from the server host view.

Returns:
Server host parsed from the server port view.

● **setServerPort**

```
public void setServerPort(String port)
```

Sets the server port view.

Parameters:
port - Value to be shown in the server port view.

● **getServerPort**

```
public int getServerPort()
```

Parses the server port from the server port view.

Returns:
Server port parsed from the server port view.

● **setImageBase**

```
public void setImageBase(String base)
```

Sets the image base view.

Parameters:

base - Value to be shown in the image base view.

● **getImageBase**

```
public String getImageBase()
```

Parses the image base from the image base view.

Returns:
Image base parsed from the image base view.

● **setHelpURL**

```
public void setHelpURL(String url)
```

Sets the helpURL view.

Parameters:
url - Value to be shown in the helpURL view.

● **getHelpURL**

```
public String getHelpURL()
```

Parses the helpURL from the helpURL view.

Returns:
HelpURL parsed from the helpURL view.

● **setClientCount**

```
public void setClientCount(String count)
```

Sets the client count.

Parameters:
count - Value to be shown in the client count view.

● **setStatus**

```
public void setStatus(String status)
```

Sets the status message.

Parameters:
status - Message to be shown in the status line.

● **setQueryCount**

```
public void setQueryCount(String count)
```

Sets the query count.

Parameters:
count - Value to be shown in the query count view.

● **setCacheHit**

```
public void setCacheHit(String hitRate)
```

Sets the cache hit rate.

Parameters:
hitRate - Value to be shown in the hit rate view.

● **setMeanQueryTime**

```
public void setMeanQueryTime(String queryTime)
```

Sets the mean query time.

Parameters:
queryTime - Value to be shown in the mean query time view.

● **setQueryTime**

```
public void setQueryTime(String queryTime)
```

Sets the query time.

Parameters:
queryTime - Value to be shown in the query time view.

● **setImageQueryTime**

```
public void setImageQueryTime(String queryTime)
```

Sets the image query time.

Parameters:
queryTime - Value to be shown in the image query time view.

● **startClicked**

```
public void startClicked()
```

Handles a pressed start button.

● **stopClicked**

```
public void stopClicked()
```

Handles a pressed stop button.

Class views.DynamicButton

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +----javax.swing.JButton
                                           |
                                           +----views.DynamicButton
```

---

public class **DynamicButton**
extends JButton
implements MouseListener

Image Button changing its border on rollovers.

Version:
    0.1, Feb 1999
Author:
    Beat Signer

Constructors

DynamicButton

```
 public DynamicButton(String label,
                      ImageIcon image)
```

Constructs a new dynamic button.

Parameters:
    label - Label of the button.
    image - Image to be shown on the button.

DynamicButton

```
 public DynamicButton(String label)
```

Constructs a new dynamic button without an image.

Parameters:
    label - Label of the button.

Methods

mouseEntered

```
 public void mouseEntered(MouseEvent event)
```

Handles an entering of the cursor in the button area.

Parameters:
    event - Associated mouse event.

mouseExited

```
 public void mouseExited(MouseEvent event)
```

Handles a leaving of the cursor from the button area.

Parameters:
event - Associated mouse event.

● **mouseClicked**

```
public void mouseClicked(MouseEvent event)
```

Handles a clicked button.

Parameters:
event - Associated mouse event.

● **mousePressed**

```
public void mousePressed(MouseEvent event)
```

Handles a pressed button.

Parameters:
event - Associated mouse event.

● **mouseReleased**

```
public void mouseReleased(MouseEvent event)
```

Handles a released button.

Parameters:
event - Associated mouse event.

Class views.EmptyView

```
java.lang.Object
   |
   +----views.EmptyView
```

---

public class **EmptyView**
extends Object
implements AgentView

Empty view for agents without a view.

Version:
    0.1, Feb 1999
Author:
    Beat Signer

Constructors

EmptyView

```
public EmptyView(Agent caller)
```

Constructs a new empty view.

Parameters:
    caller - Agent generating the view.

Methods

getCaller

```
public Agent getCaller()
```

Returns the agent which generated the view.

Returns:
    Agent which generated the view.

setLog

```
public void setLog(String text)
```

Sets the text in the log view.

Parameters:
    text - Text to be set in the log view.

addToLog

```
public void addToLog(String text)
```

Adds the message to the log view.

Parameters:
    text - Message to be added to the log view.

getLog

```
public String getLog()
```

Parses the log from the log view.

Returns:

Log parsed from the log view.

● **setListenPort**

```
public void setListenPort(String port)
```

Sets the listen port view.

Parameters:
port - Value to be shown in the listen port view.

● **getListenPort**

```
public int getListenPort()
```

Parses the listen port from the listen port view.

Returns:
Listen port parsed from the listen port view.

● **setServerHost**

```
public void setServerHost(String host)
```

Sets the server host view.

Parameters:
host - Value to be shown in the server host view.

● **getServerHost**

```
public String getServerHost()
```

Parses the server host from the server host view.

Returns:
Server host parsed from the server port view.

● **setServerPort**

```
public void setServerPort(String port)
```

Sets the server port view.

Parameters:
port - Value to be shown in the server port view.

● **getServerPort**

```
public int getServerPort()
```

Parses the server port from the server port view.

Returns:
Server port parsed from the server port view.

● **setImageBase**

```
public void setImageBase(String base)
```

Sets the image base view.

Parameters:
base - Value to be shown in the image base view.

● **getImageBase**

```
public String getImageBase()
```

Parses the image base from the image base view.

Returns:
Image base parsed from the image base view.

● **setHelpURL**

```
public void setHelpURL(String url)
```

Sets the helpURL view.

Parameters:
url - Value to be shown in the helpURL view.

● **getHelpURL**

```
public String getHelpURL()
```

Parses the helpURL from the helpURL view.

Returns:
HelpURL parsed from the helpURL view.

● **setClientCount**

```
public void setClientCount(String count)
```

Sets the client count.

Parameters:
count - Value to be shown in the client count view.

● **setStatus**

```
public void setStatus(String status)
```

Sets the status message.

Parameters:
status - Message to be shown in the status line.

● **setQueryCount**

```
public void setQueryCount(String count)
```

Sets the query count.

Parameters:
count - Value to be shown in the query count view.

● **setCacheHit**

```
public void setCacheHit(String hitRate)
```

Sets the cache hit rate.

Parameters:
hitRate - Value to be shown in the hit rate view.

● **setMeanQueryTime**

```
public void setMeanQueryTime(String queryTime)
```

Sets the mean query time.

Parameters:
queryTime - Value to be shown in the mean query time view.

● **setQueryTime**

```
public void setQueryTime(String queryTime)
```

Sets the query time.

Parameters:
queryTime - Value to be shown in the query time view.

● **setImageQueryTime**

```
public void setImageQueryTime(String queryTime)
```

Sets the image query time.

Parameters:
queryTime - Value to be shown in the image query time view.

● **startClicked**

```
public void startClicked()
```

Handles a pressed start button.

● **stopClicked**

```
public void stopClicked()
```

Handles a pressed stop button.

● **setVisible**

```
public void setVisible(boolean visible)
```

Sets the visible state of the agent view.

Parameters:
vivible - TRUE if the agent view should be shown, FALSE otherwise.

Class views.Layout

```
java.lang.Object
   |
   +----views.Layout
```

---

public class **Layout**
extends Object

Tool for layouting with a grid bag layout.

Version:
    0.1, Jan 1999
Author:
    Beat Signer

Constructors

Layout

```
 public Layout()
```

Methods

addComponent

```
 public static void addComponent(Container container,
                                 Component component,
                                 int top,
                                 int left,
                                 int bottom,
                                 int right)
```

Adds a component to a container with a GridBagLayout.

Parameters:
    container - The container the component should be added.
    component - The Component to be added.
    top - Border at the top.
    left - Border at the left.
    bottom - Border at the bottom.
    right - Border at the right
See Also:
    GridBagConstraints

addComponent

```
 public static void addComponent(Container container,
                                 Component component,
                                 int gridx,
                                 int gridy,
                                 int gridwidth,
                                 int gridheight,
                                 int top,
                                 int left,
                                 int bottom,
                                 int right,
                                 int anchor,
                                 int fill,
                                 double weightx,
                                 double weighty)
```

Adds a component to a container with a GridBagLayout.

Parameters:

container - The container the component should be added.
component - The Component to be added.
gridx - Speciefies the cell at the left of the component's display area, where the leftmost cell has 'gridx = 0'.
gridy - Specifies teh cell at the top of the component's display area, where rhe topmost cell has 'gridy =0'.
gridwidth - Specifies the number of cells in a row for the component's display area.
gridheight - Specifies the number of cells in a colum for the component's display area.
top - Border at the top.
left - Border at the left.
bottom - Border at the bottom.
right - Border at the right
anchor - Used when the component is smaller than its display area. It determines where to place the component.
fill - Used when the component's display area is larger than the componenet's requested size. It determines wheter to resize the component, and if so, how.

See Also:

GridBagConstraints

Class views.LogView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----views.LogView
```

---

public class **LogView**
extends JPanel

Log view showing the contents of the logfile.

Version:
0.1, Jan 1999
Author:
Beat Signer

Constructors

● **LogView**

```
public LogView(boolean imageButton)
```

Constructs a new log view.

Parameters:
imageButton - TRUE if images should be shown on the buttons, FALSE otherwise.

Methods

● **add**

```
public void add(String text)
```

Adds some text to the log view.

Parameters:
text - Text to be added to the log view.

● **set**

```
public void set(String text)
```

Sets some text to the log view.

Parameters:
text - Text to be shown in the log view.

● **get**

```
public String get()
```

Returns the text in the log view.

Returns:
Text parsed from the log view.

Class views.MiddleLayerAgentView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----javax.swing.JFrame
                                           |
                                           +----views.MiddleLayerAgentView
```

---

public class **MiddleLayerAgentView**
extends JFrame
implements AgentView

Middle layer agent view showing the caches and other relevant parameters.

Version:
0.1, Feb 1999
Author:
Beat Signer

Constructors

MiddleLayerAgentView

```
public MiddleLayerAgentView(MiddleLayerAgent caller)
```

Constructs a new middle layer agent view.

Parameters:
caller - Middle layer agent generatin the view.

Methods

getCaller

```
public Agent getCaller()
```

Returns the agent which generated the view.

Returns:
Agent which generated the view.

setLog

```
public void setLog(String text)
```

Sets the text in the log view.

Parameters:
text - Text to be set in the log view.

addToLog

```
public void addToLog(String text)
```

Adds the message to the log view.

Parameters:
text - Message to be added to the log view.

● **getLog**

```
public String getLog()
```

Parses the log from the log view.

Returns:
Log parsed from the log view.

● **setListenPort**

```
public void setListenPort(String port)
```

Sets the listen port view.

Parameters:
port - Value to be shown in the listen port view.

● **getListenPort**

```
public int getListenPort()
```

Parses the listen port from the listen port view.

Returns:
Listen port parsed from the listen port view.

● **setServerHost**

```
public void setServerHost(String host)
```

Sets the server host view.

Parameters:
host - Value to be shown in the server host view.

● **getServerHost**

```
public String getServerHost()
```

Parses the server host from the server host view.

Returns:
Server host parsed from the server port view.

● **setServerPort**

```
public void setServerPort(String port)
```

Sets the server port view.

Parameters:
port - Value to be shown in the server port view.

● **getServerPort**

```
public int getServerPort()
```

Parses the server port from the server port view.

Returns:
Server port parsed from the server port view.

● **setImageBase**

```
public void setImageBase(String base)
```

Sets the image base view.

Parameters:

base - Value to be shown in the image base view.

● **getImageBase**

```
public String getImageBase()
```

Parses the image base from the image base view.

Returns:
Image base parsed from the image base view.

● **setHelpURL**

```
public void setHelpURL(String url)
```

Sets the helpURL view.

Parameters:
url - Value to be shown in the helpURL view.

● **getHelpURL**

```
public String getHelpURL()
```

Parses the helpURL from the helpURL view.

Returns:
HelpURL parsed from the helpURL view.

● **setClientCount**

```
public void setClientCount(String count)
```

Sets the client count.

Parameters:
count - Value to be shown in the client count view.

● **setStatus**

```
public void setStatus(String status)
```

Sets the status message.

Parameters:
status - Message to be shown in the status line.

● **setQueryCount**

```
public void setQueryCount(String count)
```

Sets the query count.

Parameters:
count - Value to be shown in the query count view.

● **setCacheHit**

```
public void setCacheHit(String hitRate)
```

Sets the cache hit rate.

Parameters:
hitRate - Value to be shown in the hit rate view.

● **setMeanQueryTime**

```
public void setMeanQueryTime(String queryTime)
```

Sets the mean query time.

Parameters:
queryTime - Value to be shown in the mean query time view.

● **setQueryTime**

```
public void setQueryTime(String queryTime)
```

Sets the query time.

Parameters:
queryTime - Value to be shown in the query time view.

● **setImageQueryTime**

```
public void setImageQueryTime(String queryTime)
```

Sets the image query time.

Parameters:
queryTime - Value to be shown in the image query time view.

● **startClicked**

```
public void startClicked()
```

Handles a pressed start button.

● **stopClicked**

```
public void stopClicked()
```

Handles a pressed stop button.

Class views.ParameterView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----views.ParameterView
```

---

public class **ParameterView**
extends JPanel

View showing the relevant paramters of an agent.

Version:
  0.1, Jan 1999
Author:
  Beat Signer

Constructors

● **ParameterView**

```
public ParameterView()
```

Constructs a new parameters view.

Methods

● **setListenPort**

```
public void setListenPort(String port)
```

Sets the listen port view.

Parameters:
  port - Value to be shown in the listen port view.

● **getListenPort**

```
public int getListenPort()
```

Parses the listen port from the listen port view.

Returns:
  Listen port parsed from the listen port view.

● **setServerHost**

```
public void setServerHost(String host)
```

Sets the server host view.

Parameters:
  host - Value to be shown in the server host view.

● **getServerHost**

```
public String getServerHost()
```

Parses the server host from the server host view.

Returns:
Server host parsed from the server port view.

● **setServerPort**

```
public void setServerPort(String port)
```

Sets the server port view.

Parameters:
port - Value to be shown in the server port view.

● **getServerPort**

```
public int getServerPort()
```

Parses the server port from the server port view.

Returns:
Server port parsed from the server port view.

● **setImageBase**

```
public void setImageBase(String base)
```

Sets the image base view.

Parameters:
base - Value to be shown in the image base view.

● **getImageBase**

```
public String getImageBase()
```

Parses the image base from the image base view.

Returns:
Image base parsed from the image base view.

● **setHelpURL**

```
public void setHelpURL(String url)
```

Sets the helpURL view.

Parameters:
url - Value to be shown in the helpURL view.

● **getHelpURL**

```
public String getHelpURL()
```

Parses the helpURL from the helpURL view.

Returns:
HelpURL parsed from the helpURL view.

● **setEditable**

```
public void setEditable(boolean editable)
```

Enables and disables the editing of the parameter fields.

Parameters:
editable - True if the fields should be editable, FALSE otherwise.

Class views.Picture

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----views.Picture
```

---

public class **Picture**
extends Component

Container for images.

Version:
0.1, Dec 1998
Author:
Beat Signer

Constructors

Picture

```
public Picture(Image image)
```

Constructs a new container for an image. param image The image to be displayed in the image container.

Picture

```
public Picture(String imgSrc)
```

Constructs a new container for an image.

Parameters:
imgSrc - URL of the image to be displayed in the image container.

Methods

paint

```
public void paint(Graphics g)
```

Paints the Picture.

Parameters:
The - graphics context to use for painting.
Overrides:
paint in class Component

getPreferredSize

```
public Dimension getPreferredSize()
```

Gets the preferred size of this picture.

Returns:
Dimension object representing the preferred size of this picture.
Overrides:
getPreferredSize in class Component

getMinimumSize

```
public Dimension getMinimumSize()
```

Gets the minimum size of this picture.

Returns:

Dimension object representing the minimum size of this picture.

Overrides:

getMinimumSize in class Component

Class views.StatisticView

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----views.StatisticView
```

---

public class **StatisticView**
extends JPanel

View for statistics like cache hit rate, query time etc.

Version:
0.1, Jan 1999
Author:
Beat Signer

Constructors

StatisticView

```
public StatisticView()
```

Constructs a new statistic view.

Methods

setClientCount

```
public synchronized void setClientCount(String count)
```

Sets the client count.

Parameters:
count - Value to be shown in the client count view.

setQueryCount

```
public synchronized void setQueryCount(String count)
```

Sets the query count.

Parameters:
count - Value to be shown in the query count view.

setCacheHit

```
public synchronized void setCacheHit(String hitRate)
```

Sets the cache hit rate.

Parameters:
hitRate - Value to be shown in the hit rate view.

setMeanQueryTime

```
public synchronized void setMeanQueryTime(String time)
```

Sets the mean query time.

Parameters:
queryTime - Value to be shown in the mean query time view.

● **setQueryTime**

```
public synchronized void setQueryTime(String time)
```

Sets the query time.

Parameters:
queryTime - Value to be shown in the query time view.

● **setImageQueryTime**

```
public synchronized void setImageQueryTime(String time)
```

Sets the image query time which is the time nedded to return the address of the image and NOT the time to transfer the image itself!

Parameters:
queryTime - Value to be shown in the image query time view.

Class views.UpdateThread

```
java.lang.Object
   |
   +----views.UpdateThread
```

---

public class **UpdateThread**
extends Object
implements Runnable

Thread for updating the CacheView.

Version:
0.1, Feb 1999
Author:
Beat Signer

Constructors

● **UpdateThread**

```
public UpdateThread(CacheView parent)
```

Constructs a new update thread.

Parameters:
parent - CacheView to be updated.

Methods

● **run**

```
public void run()
```

Updates the CacheView.

Class views.WarningDialog

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----views.WarningDialog
```

---

public class **WarningDialog**
extends Dialog

Show a modal warning dialog to the user.

Version:
0.1, Dec 1998
Author:
Beat Signer

Constructors

WarningDialog

```
 public WarningDialog(Frame parent,
                      String warning)
```

Constructs a new warning dialog.

Parameters:
parent - Parent Frame.
warning - String of the message to be displayed. the '#' is the representative for the beginning of a new line.

Bibliography

[EN97] Antonia Erni and Moira C. Norrie. *Agent Based Internet Database Services.* 4th Doctoral Consortium in Conjunction with 9th Conf. on Advanced Information Systems Engineering (CAiSE'97), Barcelona, Spain, 1997.

[ENK98] Antonia Erni, Moira C. Norrie, and Adrian Kobler. Generic agent framework for internet information systems. In *Proceedings of IFIP WG 8.1 98 Conference, Bejing, China*, 1998.

[Fla97] David Flanagan. *Java in a Nutshell: A Desktop Quick Reference.* A Nutshell handbook. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, second edition, 1997.

[Fra96] Michael J. Franklin. *Client Data Caching: A Foundation for High Performance Object Database Systems.* Kluwer Academic Publishers, 101 Philip Drive, Norwell, MA 02061, 1996.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[GM97] David M. Geary and A. L. McClellan. *Graphic Java: Mastering the AWT.* Prentice-Hall, 2550 Garcia Avenue, Mountain View, CA 94043-1100, first edition, 1997.

[HC97] Cay S. Horstmann and Gary Cornell. *Core Java 1.1: Volume 1: Fundamentals.* Prentice-Hall, 2550 Garcia Avenue, Mountain View, CA 94043-1100, third edition, 1997.

[HC98] Cay S. Horstmann and Gary Cornell. *Core Java 1.1: Volume 2: Advanced Features.* Prentice-Hall, 901 San Antonio Road, Palo Alto, CA 94303, first edition, 1998.

[Rus97] Philip Ruser. Agent-Based Product Data Information System. Diploma thesis, Institute for Information Systems, ETHZ, 1997.

[SG94] Avi Silberschatz and Peter Galvin. *Operation System Concepts.* Addison-Wesley, 1994.

[Tan92] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Upper Saddle River, NJ 07458, 1992.

[Wue95] Alain Wuergler. Object Model System: An Object Database Management System for the OM Data Model. Master's thesis, Institute for Information Systems, ETHZ, 1995.