Project Description: Personal Virtual File System

ETH Zurich

April 4, 2013

Hand-out:	April 4, 2013
Due:	May 13, 2013 (final deadline)

1 Overview

The goal of this project is to develop a single-file Virtual File System (**VFS**) in either Java or C#. A VFS is usually built on top of a host file system, and it enables access to files located in different host file systems in a uniformed way. A single-file VFS could also be used as a manageable container with the functionality of a concrete file system through the usage of software. Typical applications of a single-file VFS include virtual machines, emulators, encryption (providing a FS with encrypted data), or even programming IDEs (IntelliJ[1]). You can find more information on VFSs on Wikipedia[2].

2 Deadlines

All project work must be done in teams of **3 people**. The project itself is divided in 3 parts, each with its own deadline:

- The VFS core (due April 8, 2013)
- The VFS browser (due April 22, 2013)
- The VFS synchronization server (due May 13, 2013)

At each deadline you need to hand in a written report on the design and implementation of the corresponding part. The proposed project structure allows incremental development.

3 Code Repository

Use the SVN repository at `https://code.vis.ethz.ch/`. Create a project using as name the groupID that will be sent to you via email. Add your assistants username as project member (for usernames see 7).

4 Project Grading

You can earn 100 points in the project. The points are composed as follows:

- Requirements coverage, max 40 points
- Code quality, max 20 points

[1]`http://confluence.jetbrains.com/display/IDEADEV/IntelliJ+IDEA+Virtual+File+Systemg`
[2]`http://en.wikipedia.org/wiki/Virtual_file_system`

- Unit tests, max 15 points
- Final report and presentation, max 15 points
- Bonus features, max 10 points

The grade will be determined based on the project's final implementation and the final report as submitted by May 13. However, each team has to provide an implementation and a report summarizing the current design and project status at each intermediate deadline. The implementation at the intermediate deadlines has to be runnable.

4.1 Statement coverage of unit tests

Statement coverage[3] is a measure used in software testing to describe the percentage of statements that has been tested: in general, the higher the coverage, the more reliable the test results. In this project we will use the following tools to collect coverage information on your tests.[4]

- Java:
  Eclipse and the EclEmma plugin (`http://www.eclemma.org/`)
- C#: JetBrains dotCover (`http://www.jetbrains.com/dotcover/`)[5]

4.2 Code smells

Code smells[6] are indications of potential problems in the source code. We will use the following tools to detect smells in your code:

- Java: PMD (`http://pmd.sourceforge.net/`)
- C#: VS Code Analysis (`http://msdn.microsoft.com/en-us/library/3z0aeatx.aspx`)

For both tools the quick start guide is available on the course's website.

5 General notes

1. The VFS core, except for the bonus features, may *not* rely on any DBMS system or 3rd party library.
2. The system should be robust with respect to incorrect user input. For example, when a user tries to import files from a nonexistent directory, the VFS should not crash.
3. Design your application in a modular way to support separation of concerns. For example, VFS core should not depend on the GUI.

6 Project Details

6.1 Part 1: VFS core (due April 8)

The goal of the first part is to develop the **VFS core**. The VFS core operates on **virtual disks**, each being a single file in the host file system. The core provides an API to facilitate creating, modifying and deleting virtual disks. Particularly, the core provides means for its client to: 1) create a virtual disk of user-specified size, as well as delete an existing virtual disk; 2) import files and directories from the host file system to a virtual disk, as well as export files and directories from the VFS to the host

[3]`http://en.wikipedia.org/wiki/Code_coverage`
[4]All tools will be running under the default settings.
[5]You can use dotCover with the academic license we sent you before.
[6]`http://en.wikipedia.org/wiki/Code_smell`

file system; and 3) navigate through the directories of a virtual disk, as well as rename, copy, remove or move existing files and directories in the virtual disk. The size of a virtual disk is fixed during its lifetime, and errors should be reported when pending operations are infeasible due to this restriction.

6.1.1 Requirements

1. The virtual disk must be stored in a single file in the working directory in the host file system.
2. VFS must support the creation of a new disk with the specified maximum size at the specified location in the host file system.
3. VFS must support several virtual disks in the host file system.
4. VFS must support disposing of the virtual disk.
5. VFS must support creating/deleting/renaming directories and files.
6. VFS must support navigation: listing of files and folders, and going to a location expressed by a concrete path.
7. VFS must support moving/copying directories and files, including hierarchy.
8. VFS must support importing files and directories from the host file system.
9. VFS must support exporting files and directories to the host file system.
10. VFS must support querying of free/occupied space in the virtual disk.

6.1.2 Bonus Features

Basic

1. Compression, if implemented with 3d party library.
2. Encryption, if implemented with 3d party library.
3. Elastic disk: Virtual disk can dynamically grow or shrink, depending on its occupied space.

Advanced

1. Compression, if implemented by hand (you can take a look at the arithmetic compression[7]).
2. Encryption, if implemented by hand.
3. Large data: This means, that VFS core can store & operate amount of data, that can't fit to PC RAM (typically, more than 4Gb).

6.1.3 Hints

1. Develop a set of unit tests prior to the development, try to work using the Test-Driven-Development (TDD) approach.
2. The key challenge is to develop an efficient storage structure at physical layer. You can take an advantage of specialized data structures, like B-trees[8].
3. Keep in mind that in the next part you will be required to implement file search.
4. At this point, it is helpful to develop a bash-like console application, to test your VFS (with operations like ls, cd, copy, etc)

[7] `http://en.wikipedia.org/wiki/Arithmetic_coding`
[8] `http://en.wikipedia.org/wiki/B-tree#In_filesystems`

6.2 Part 2: VFS browser (due April 22)

The goal of this part is to provide a Graphical User Interface (GUI) for the VFS core. The target platform is up to you: it could be a desktop application (Windows or Linux), web-application or a mobile client. The GUI must support all implemented operations from Part 1. We require navigation with both mouse/touch and keyboard. For this part of the project you only have to support a single user of the VFS, multi-user support is not required. You are encouraged to design a nice and user-friendly interface. Bonus points are given for a responsive GUI that does not stall during long-running operations, like import or search. Programming an additional, different client is another option to earn bonus points.

6.2.1 Requirements

1. The browser should be implemented on one of the following platforms: desktop, web or mobile.
   (a) Web applications written in C# must use Silverlight.
   (b) Web applications written in Java must either use Google Web Toolkit or JavaFX.
   (c) Mobile applications must target Android or Windows Phone.
2. The browser should support all operations from Part 1 (VFS core). For example, users should be able to select a file/folder and copy it to another location without using console commands.
3. The browser should support both single and multiple selection of files/folders.
4. The browser should support keyboard navigation. The mandatory set of operations includes folder navigation, going to parent and child folders (this is optional for mobile applications due to limited keyboard functionality).
5. The browser should support mouse navigation (or touch in case of the mobile platform). The required operations are the same as in requirement 4.
6. The browser should support file-name search based on user-given keybwords. The search should provide options for: case sensitive/ case insensitive search; restrict search to folder; restrict search to folder and subfolders.

6.2.2 Bonus Points

Basic

1. Responsive UI, i.e. the browser does not stall during long-running operations (i.e. file search or import).
2. Advanced search; For example, search with wildcards/regexp, and approximate search based on some metric, e.g. *edit distance.*
3. Nice-to-have features like operation progress report (e.g. the number of files processed during export) or drag-and-drop for manipulative operations (move, copy, import).

Advanced

1. The browser is implemented for an additional platform.
2. Efficient full-text search (using some sort of indexing).

Other additional features can be approved by your assistant on a case-by-case basis.

6.2.3 Hints

1. The web interface of Dropbox could be used as an example of a good interface.

6.3 Part 3: Synchronization Server (due May 13)

The goal of this part is to develop a synchronization server which will propagate changes in the virtual disk from one machine to another over a network. The management of the distributed VFSs is made on an account basis, i.e. in order to use the server, one must have an account and link a virtual disk to this account. Note that unlinked disks should not be synchronized. You should extend the browser from the previous part to be a client of the synchronization server. Additional points are given for implementing a concurrent server (when changes on one account are done simultaneously on different machines). Another way to earn bonus points is to provide a mocked set of unit tests.

7 Contacts

Christian Estler (estlerh)	christian.estler@inf.ethz.ch
Alexey Kolesnichenko (alexeyk)	alexey.kolesnichenko@inf.ethz.ch
Max (Yu) Pei (ypei)	yu.pei@inf.ethz.ch
Julian Tschannen (juliant)	julian.tschannen@inf.ethz.ch