IN3405 Bachelor Thesis

SwiftTV

Bringing 4th generation P2P to SmartTV

Smart experience and Connectivity
Movies, shows, apps and more
The ultimate home-base of entertainment
This Smart TV does it all
Samsung SMART TV
YouTube

Authors:
S.E. Austin (4005996)
A. Drif (4030532)
W.K.H. Ko (4005686)
J.E.T. Tan (4032918)

Supervisor:
Dr.Ir. J.A. Pouwelse

July 13, 2012

Preface

In the third year of the Computer Science curriculum at Delft University of Technology, students have to do their Bachelor project where they have to work under supervision of a client being a company or a professor at the university. This project consists of creating a software solution for a given realistic problem with the skills and knowledge that are obtained during the Bachelor programme.

This report is part of our Bachelor Project, carried out in the academic year 2011-2012. The goal of this project is to develop a 4th generation peer-to-peer application to be used on a Samsung SmartTV.

We would like to thank our supervisor, J.A. Pouwelse, and the Tribler team, who helped us with this project. In particular, we would like to thank for their help:

- Dr. Arno Bakker;
- Ir. R. Petrocco;
- P.B. Schoon;
- T.M. Schaap;
- Ir. N.S.M. Zeilemaker;
- The SamyGO Community.

Contents

Summary

Since their invention, TV's have become one of the most popular media devices and can be found in almost every livingroom in the world. For a long time, the functionality of the TV stayed the same: the ability to view television programs at certain fixed times of the day. Recently there has been development in the television market adding computing power and internet connectability to televisions. These new features open a whole new world of possibilities.

The goal of this project was to create an application that runs on a Samsung SmartTV and uses the libswift peer-to-peer engine to download, upload and stream files. To create an application for a Samsung SmartTV a software development kit has been provided which allows programmers to create apps using JavaScript, HTML, CSS and Flash. This software development kit was used to create the front-end of our application. The front-end consists of an internal media player to handle streaming content and media playback found on an external USB device.

A Samsung SmartTV runs a linux kernel in which we can properly run our download-engine which is written in C++. To be able to do this we gained root access to the TV, so it became possible to operate in the linux environment. After this was done, the back-end for our application was implemented in C++. In order to make the front-end communicate with the back-end, we used the client-server architecture where the front-end acts as a client and the back-end as a server.

It was also needed to provide a communication mechanism between TV's, so that content could be found and shared between TV's. This was achieved by using the Dispersy and DHT modules developed by the tribler team. These modules were, however, implemented in Python. In order to make use of this, a part of our application is implemented in Python.

Several steps were taken in order to develop this application. An analysis of requirements was made, which serves as the foundation of the design of our application. A design following the client-server pattern was made, after which class and sequence diagrams were created.

During development we encountered a lot of problems, mostly because the TV runs a stripped version of the linux kernel. The consequence was that a lot of things were not available on the TV, so we had to cross-compile the necessities for the TV or use a binary compatible development platform. Other problems we encountered were for example that the TV sends TCP RST packages to

itself. Also, problems during implementation occured when we used threads for the implementation of the HTTP server.

Even though we encountered these problems, we were still able to maintain our schedule and develop everything in time. By doing several things together and by dividing the work correctly we were able to overcome the mentioned problems. The result is the first fourth generation peer-to-peer application in the world running natively on a Samsung SmartTV. Even though this is an achievement we are proud of, the application is still a proof of concept since current TV's are not strong enough to run the application smoothly. This might change in the future, when TV's become more powerful.

Chapter 1

Introduction

Since their invention, TV's have become one of the most popular media devices and can be found in almost every livingroom in the world. For a long time, the functionality of the TV stayed the same: the ability to view television programs at certain fixed times of the day. Recently there has been development in the television market adding computing power and internet connectability to televisions. These new features open a whole new world of possibilities. One of these possibilities, and the one our project focusses on, being the ability to stream video content on demand over the Internet.

The reason peer-to-peer streaming is such an attractive feature is because it is at the base of video on demand technology. It has always been possible to view video content on televisions but the programs are always at set times and people are not always available at certain times to watch the programs they would like to see. Streaming content on demand would solve this problem completely by allowing users to view the content they are interested in whenever they like. Current technology is capable of providing this service now but it has not yet become popular in televisions. Our goal is to create an application that fulfills this need and set a step in the direction of on demand peer-to-peer video content.

The client, but also supervisor of this project is Dr.Ir. Johan Pouwelse from the Parallel and Distributed Systems group on the faculty of EEMCS of the Delft University of Technology. He is the scientific director of several peer-to-peer research initiatives and also the founder and leader of the Tribler[9] [14] peer-to-peer research team. Tribler is an open source peer-to-peer client that is completely decentralised. The application to be implemented is required to have the same download-engine, libswift,[8] as Tribler excluding the part that handles torrents. The application comes along with an internal media player to handle streaming content and possibly media playback found on an external USB device and should run natively on the TV.

Libswift is a lightweight 4th generation peer-to-peer based transport protocol for content dissemination that can run on top of other protocols, such as UDP, TCP, HTTP, or as RTP profile. It can be used for both live streaming and conventional downloading purposes. 4th generation peer-to-peer software means that it is fully self-organised, removing the need for any server.[13]

Because libswift is a very generic protocol, it is easy to merge with the existing technology of the TV. Since the smart TV has limited memory and CPU resources, it is necessary that the application is built as lightweight as possible. The streaming capabilities also enables the user to maximise utility of the TV, while still being able to download data and store it. A major drawback is however that downloading and streaming content is limited to the memory of the TV. Because of this limitation, we were only able to create a proof of concept application, rather than an application which can be used immediately by end users.

Chapter 2

Problem Statement

The goal of this project is to create an application that runs on a Samsung SmartTV and uses the libswift peer-to-peer engine to download, upload and stream files.[1] To create an application for a Samsung SmartTV[2] a software development kit has been provided which allows programmers to create apps using JavaScript, HTML, CSS and Flash (See terms of reference).

A Samsung SmartTV however runs a linux kernel in which we can properly run our download-engine which is written in C++. To be able to do this we must gain root access to the TV, so it becomes possible to operate in the linux environment (See also orientation report). A problem that arises here is that the SDK front-end will be completely separated from the back-end where libswift will run. A solution has to be found to escape from this sandboxed environment. It is also needed to provide a communication mechanism between TV's, so that content can be found and shared between TV's.

Problems to be solved

- Root a Samsung television
- Create a C++ back-end with swift
- Break out of the sandbox
- Create a JavaScript front-end
- Inter-TV communication

This application will be a proof of concept to show that televisions are capable of running peer-to-peer software natively, without any attached devices, in this case with the help of the libswift download-engine. Therefore usability is not the primary goal of this application, although the application will be designed to be as usable as possible.

[1] `http://www.tribler.org/trac/wiki/SwiftTV`
[2] `http://www.samsung.com/nl/experience/tv/smarttv/`

Chapter 3

Requirements Analysis

In this chapter our requirement elicitation is provided. Here we discuss the elements the client wants to see back in the end-product. For this purpose the client, who also took the role of supervisor upon himself together with the Tribler team, was interviewed. The requirements that were gathered from this interview were supplemented with additional requirements from our side to complete the product. The requirements were separated into functional and non-functional requirements. Then, use cases for our application were designed, which correspond with the clients wishes. In this phase, a business class diagram was also made which is easier to understand for the client than the complete class diagram of the whole application. After researching the framework present on the Samsung SmartTV and the possibilities of libswift, a design for the software architecture could be made. Lastly, implementation models such as sequence diagrams and class diagrams were made, which serve as blueprints for the application. Also a MosCoW model was made where the priority off the features to be implemented are stated.

3.1 Requirements

The requirements can be split in two kinds, functional and non-functional requirements. These requirements correspond to the use cases as mentioned in section 3.2. The non-functional requirements can then again be split in quality and platform requirements. In this section, the constraints the application has to deal with are also discussed.

3.1.1 Functional requirements

Functional requirements specify the core functionality of the application. These requirements are covered by the use cases in the next section. The functional requirements are as follows:

1. By using the libswift protocol, given two peers A and B:

    1.1. A is able to download from B (UC 12);

    1.2. A is able to upload to B (Should happen automatically after download is finished);

    1.3. A is able to stream to/from B (UC 11);

2. The users should be able to search for files other peers own by using Dispersy[10] (UC 9, 10);
3. The users should be able to search/browse files they own themselves (UC 2, 8, 10);
4. The users should be able to playback media content. Media in this context means any kind of video or audio file supported by the TV (UC 6); Media in this context means any kind of video or audio file (UC 6);
5. Users should be able to:

    5.1. Move files on the file system (UC 5);

    5.2. Edit files on the file system, i.e. rename files (UC 3);

    5.3. Remove files on the file system (UC 4);

6. Users should be able to sort files by name, size, date and type (UC 13, 14, 15, 16, 17);
7. Users should be able to separate private files from public files (UC 7);
8. Users should be able to limit the upload/download speed (UC 1).

3.1.2 Non-functional requirements

Non-functional requirements are those requirements that have to do with the operation of the system. Unlike functional requirements, they do not describe what the services the system should provide, but more how the system should work. We identified two kinds of non-functional requirements for our application, quality and platform requirements.

Quality requirements

9. The response time of the system should be as low as possible since TVs are real-time systems. We aim to limit the response time to 300 ms;
10. The RAM memory used by the application should not exceed 256 MB. This includes the size of the application itself and the memory needed for calculations;
11. The application may use 100% CPU power since no other applications should run along the application to be built;
12. Downloads should be continued without loss of data after a failure;
13. Unfinished downloads should be resumed at start up;

14. The application should be able to cope with external devices, memory sticks and other external storage devices in particular;
15. The user should be able to make use of (part of) the application at any time, even when no Internet connection is available.

Platform requirements

1. The C++ back-end which makes use of the libswift library should be implemented so that it is reusable for other systems that work with other front-ends (such as Android);
2. The application is made to run on a rooted Samsung SmartTV. To see whether a Samsung TV is rootable or not, please check `http://www.samygo.tv/`;

3.1.3 Constraints

The constraints specify what our own limitations are we have to take into account when implementing the application. These can be limitations put by the software we use, or put by ourselves to reduce the application's complexity.

1. Due to the limited memory available on the TV, it is not possible to download or stream and store great amounts of data on the TV. Therefore, usage of external storage devices is mandatory to make use of the application;
2. The application must be integrated with the Samsung framework, meaning that it should be recognised by the TV as JavaScript app developed with the Samsung SDK.

3.2 Use Cases

The use case models describe interactions between one or more actors and the system. The models are used to depict what the users can do with the system and how they should operate it. In our case, we only have one actor, which is the user who watches TV. This is because our application should always maintain the same behaviour for everyone who uses it; everyone using the application has the same rights, privileges and use cases of the system. The use cases are shown in figure 3.1, in which the relations between them are also shown. The user should be able to carry out all the use cases, these relations are left out for clarity. The use cases in figure 3.1 are explained in further detail in their descriptions and with the use of UML sequence diagrams in sections 3.2.1 and appendix respectively.

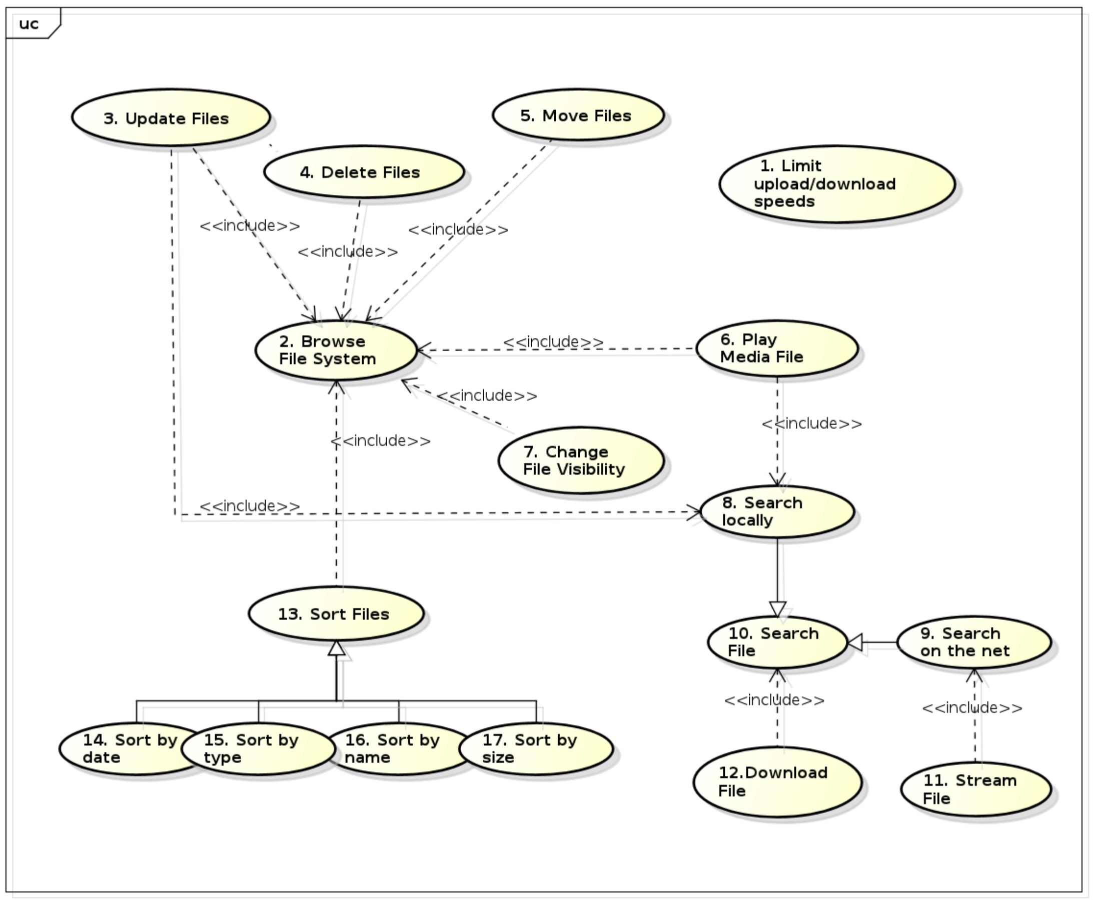


Figure 3.1: Use case diagram of the application to be built. Note that the associations from the user to the use cases are left out in order to keep the diagram clear.

3.2.1 Use Case Descriptions

Name	Limit upload/download speeds
Goal	Save bandwidth and memory.
Preconditions	–
Summary	The user adjusts the upload or download speed.
Steps	1. User clicks the settings button; 2. *System* shows the settings page; 3. User sets the download and upload speeds; 4. User clicks OK button; 5. *System* saves the settings and applies the new speeds;
Postconditions	Downloads and uploads will never exceed the new speeds.

Table 3.1: Use Case 1: Limit upload/download speeds. See appendix for the sequence diagram.

Name	Browse File System
Goal	Getting an overview of the files in the file system.
Preconditions	One or more external storage devices are connected.
Summary	The user browses through the directories on the file system.
Related Cases	Play Media File, Move File, Update File, Delete File, Sort Files, Change File Visibility.
Steps	1. User selects the device to browse; 2. *System* shows list of files and directories of the selected device.
Postconditions	-

Table 3.2: Use Case 2: Browse File System. See appendix for the sequence diagram.

Name	Update Files
Goal	Change the properties of a file.
Preconditions	A changeable file is available locally.
Summary	The user changes the properties of a local file.
Related Cases	Includes **Browse File System** and **Search locally**.
Steps	1. User selects the file to update; 2. *System* shows the properties of the file; 3. User changes the properties and clicks the OK button; 4. *System* changes the properties and saves them.
Postconditions	File properties are changed locally and if uploading, also online.

Table 3.3: Use Case 3: Update Files. See appendix for the sequence diagram.

Name	Delete Files
Goal	Delete one or more files.
Preconditions	A deletable file is available locally.
Summary	The user deletes a file.
Related Cases	Includes **Browse File System** and **Search locally**.
Steps	1. User selects the files to delete; 2. *System* asks for confirmation; 3. User clicks the OK button; 4. *System* deletes selected files.
Postconditions	Files are deleted locally and if uploading, also online.

Table 3.4: Use Case 4: Delete Files. See appendix for the sequence diagram.

Name	Move Files
Goal	Move one, multiple files or directories.
Preconditions	A moveable file or directory is available locally.
Summary	The user selects files or directories and moves them in the file system.
Related Cases	Includes **Browse File System** and **Search locally**.
Steps	1. User selects the files or directories to move; 2. User browses to the new location; 3. *System* moves the selected files and directories.
Postconditions	Files are moved to another directory.

Table 3.5: Use Case 5: Move Files. See appendix for the sequence diagram.

Name	Play Media File
Goal	Play a music or video file or display a picture.
Preconditions	A playable file is available.
Summary	The user opens a stream or a locally available file.
Related Cases	Includes **Browse File System** and **Search locally**.
Steps	1. User selects the file to play; 2. *System* either starts the photo viewer or media player to open the file.
Postconditions	File is opened and is viewable/listenable.

Table 3.6: Use Case 6: Play Media File. See appendix for the sequence diagram.

Name	Change File Visibility
Goal	Start or stops the uploading of a file.
Preconditions	An uploadable file is available. Also, the system must be connected to the Internet.
Summary	The user checks/unchecks the visibility box of a file to control file uploads.
Related Cases	Includes **Browse File System** and **Search locally**.
Steps	1. User checks or unchecks the visibility box of a file; 2. *System* either starts uploading the file and adds the upload to the Upload Manager or stops uploading the file and removes the upload from the Upload Manager.
Postconditions	File is uploaded or removed from the Upload Manager.

Table 3.7: Use Case 7: Change File Visibility. See appendix for the sequence diagram.

Name	Search locally
Goal	Find a file on an external storage device.
Preconditions	One or more external storage devices are connected.
Summary	The user searches with keywords for files in a file system.
Related Cases	Play Media File, Move File, Update File, Delete File, Change File Visibility. Inherits **Search File**
Steps	1. User presses Search button; 2. *System* shows Search menu; 3. User sets search range on "Local"; 4. User selects the file type to search; 5. User puts keywords in the textbox and presses Start button; 6. *System* returns with a list and number of found files.
Postconditions	A list of found files is returned together with the number of found files.

Table 3.8: Use Case 8: Search locally. See appendix for the sequence diagram.

Name	Search on the net
Goal	Find a file to download or stream on the Internet.
Preconditions	The system is connected to the Internet.
Summary	The user searches with keywords for media content on the Internet.
Related Cases	Stream File, Download File. Inherits **Search File**
Steps	1. User presses Search button; 2. *System* shows Search menu; 3. User sets search range on "Online"; 4. User selects the file type to search; 5. User puts keywords in the textbox and presses Start button; 6. *System* returns with a list and number of found files.
Postconditions	A list of found files is returned together with the number of found files.

Table 3.9: Use Case 9: Search on the net. See appendix for the sequence diagram.

Name	Search File
Goal	Find a file by using keywords.
Preconditions	A keyword with length > 0 is provided by the user.
Summary	The user searches for a file.
Related Cases	Generalizes **Search locally** and **Search on the net**.
Steps	1. User inputs a keyword; 2. User selects the file type to search; 3. *System* returns the list and number of found files.
Postconditions	A list and number of found files is returned.

Table 3.10: Use Case 10: Search File. See appendix for the sequence diagram.

Name	Stream File
Goal	Listen to a stream and play its content.
Preconditions	A network connection is established. Also, an external storage device must be connected.
Summary	The user plays the content of a stream. This can happen either online or within the local network.
Related Cases	Includes **Search on the net**.
Steps	1. Selects an uploaded file; 2. *System* shows options “Download” and “Open”; 3. User selects “Open”; 4. *System* starts the videoplayer and plays the stream.
Postconditions	A stream is opened and played by the videoplayer. Temporary files are removed after playing the stream.

Table 3.11: Use Case 11: Stream File. See appendix for the sequence diagram.

Name	Download File
Goal	Download a file and puts it in the file system.
Preconditions	A network connection is established. Also, an external storage device must be connected.
Summary	The user plays the content of a stream. This can happen either online or within the local network.
Related Cases	Includes **Search on the net**.
Steps	1. Selects an uploaded file; 2. *System* shows options “Download” and “Open”; 3. User selects “Download”; 4. *System* adds the Download to the Download Manager and starts download.
Postconditions	A file is retrieved from the Internet and stored on the file system.

Table 3.12: Use Case 12: Download File. See appendix for the sequence diagram.

Name	Sort Files
Goal	Arrange the files in a coherent way.
Preconditions	An external storage device is connected.
Summary	The user sorts the files.
Related Cases	Generalizes **Sort by type**, **Sort by name**, **Sort by date**, **Sort by size** .
Steps	1. User clicks on attribute “type”, “name”, “date” or “size”; 2. *System* sorts the files and returns the sorted list.
Postconditions	A sorted list is returned.

Table 3.13: Use Case 13: Sort Files. See appendix for the sequence diagram.

Name	Sort by date
Goal	Arrange the files by date.
Preconditions	An external storage device is connected.
Summary	The user sorts the files by date.
Related Cases	Inherits **Sort files**.
Steps	1. User clicks on attribute “date”; 2. *System* sorts the files by their dates of creation and returns the sorted list (ascending); 3. User clicks on attribute “date” again; 4. *System* sorts the files by their dates of creation and returns the sorted list (descending).
Postconditions	A list sorted by date is returned.

Table 3.14: Use Case 14: Sort by date. See appendix for the sequence diagram.

Name	Sort by type
Goal	Arrange the files by type.
Preconditions	An external storage device is connected.
Summary	The user sorts the files by type.
Related Cases	Inherits **Sort files**.
Steps	1. User clicks on attribute “type”; 2. *System* sorts the files by their types and returns the sorted list (ascending); 3. User clicks on attribute “type” again; 4. *System* sorts the files by their types and returns the sorted list (descending).
Postconditions	A list sorted by type is returned.

Table 3.15: Use Case 15: Sort by type. See appendix for the sequence diagram.

Name	Sort by name
Goal	Arrange the files by name.
Preconditions	An external storage device is connected.
Summary	The user sorts the files by name.
Related Cases	Inherits **Sort files**.
Steps	1. User clicks on attribute “name”; 2. *System* sorts the files by their names and returns the sorted list (ascending); 3. User clicks on attribute “name” again; 4. *System* sorts the files by their names and returns the sorted list (descending).
Postconditions	A list sorted by name is returned.

Table 3.16: Use Case 16: Sort by name. See appendix for the sequence diagram.

Name	Sort by size
Goal	Arrange the files by size.
Preconditions	An external storage device is connected.
Summary	The user sorts the files by size.
Related Cases	Inherits **Sort files**.
Steps	1. User clicks on attribute “size”; 2. *System* sorts the files by their sizes and returns the sorted list (ascending); 3. User clicks on attribute “name” again; 4. *System* sorts the files by their sizes and returns the sorted list (descending).
Postconditions	A list sorted by size is returned.

Table 3.17: Use Case 17: Sort by size. See appendix for the sequence diagram.

3.3 Business Class diagram

The business class diagram is meant to explain the structure of the system to be built. It depicts the components of the system and their relationships, the diagram can be found in the appendix. This section explains the classes of the business class diagram in more detail.

3.3.1 FileManager

As the name implies, the FileManager class manages files. With this, users are able to browse and search for files. Also, users will be able to add and manipulate them. The FileManager holds a list containing all files and their data, which can be sorted either by date, file name, file type or size.

3.3.2 File

Files are being managed by the FileManager, as mentioned earlier. They are data structures for files on the disk, containing all necessary information about them. Files can be wrapped into a Sendable class, after which it can be published online. Also, files can be media files which can be opened by the MediaPlayer class.

3.3.3 Media

A special case of a File, which can be opened and played by the MediaPlayer. Media can be any kind of video or music file.

3.3.4 MediaPlayer

Is able to open Media files and Streams, which can then be showed on the TV. This is one of the more important classes since we are working on a TV, which is meant for displaying videos and such.

3.3.5 Sendable

Wrapper class for Files, so that the user is able to transfer the Files over the Internet. Contains basic data about the peers and the transfer speed. Sendables can be started, paused, stopped or resumed. A Sendable can then be either an Upload, Download or a Stream.

3.3.6 Download

Data structure for all incoming transfers, extends from the Sendable class. This class is made to hold special data its siblings (Upload and Stream) do not own, such as the download percentage. Also, this class is made so that it is easier to distinguish the different kinds of Sendable classes.

3.3.7 Upload

Data structure for all outgoing transfers, extends from the Sendable class. This class is made to hold special data its siblings (Download and Stream) do not own, such as the upload amount. Also, this class is made so that it is easier to distinguish the different kinds of Sendable classes.

3.3.8 Stream

Data structure for all files being streamed, extends from the Sendable class. This class is made to hold special data its siblings (Download and Upload) do not own, such as the length of the streamed file. Also, this class is made so that it is easier to distinguish the different kinds of Sendable classes.

3.3.9 DownloadManager

A manager for all Sendable classes, except for Streams. This class holds all downloaded and uploaded files and is able to exercise the methods of a specific Download or Upload. This class is also one of the more important classes, since the project revolves around the libswift engine, which is meant for sharing files over the Internet.

3.4 MosCoW

Must have

Stream functionality (req. 1.3);
Download/upload functionality (req. 1.1 and 1.2);
Media playback (req. 4);

Should have
Browse in local file system (req. 3); Add, remove and rename files (req. 5); Seperate private and public files (req. 7); Limit upload/download speeds (req. 8);
Could have
Sort files by name/date/size/type (req. 6);
Would have
External access to application; Possibility to minimise app; Download continuation without loss of data (req 12); Download continuation at start up (req 13);

Table 3.18: List of priorities.

Stream functionality and media playback are the reasons to build this application on a smart TV rather than on a pc. TV's are machines specialised in displaying media content, so it would only be appropriate to make use of this “power” of the TV. Moreover, it would be wasted not to use this feature of TV's, because then we could just develop a normal application for the pc (which saves a lot of trouble, such as cross compiling).

Since we are building a peer-to-peer application for the Samsung SmartTV, download and upload functionality are also must haves for our application.

The “Should haves” are things we have to implement, but which are less important than the core features mentioned in the must haves. These are supporting features to make the application more user friendly and more advanced. For example, if the user downloaded a file (which is a must have feature), then it would also be nice if the user could browse to that file.

The only “Could have” we have is the ability to sort the files. This feature is less important because the application will not be rendered useless if this feature was not to be implemented. It will of course add more user friendliness to the application, but it is not that the users will not be able to use the application without this feature. Thus, sorting files is one of the less important features which we could consider implementing if we have time left.

The list shown in table 3.18 consists mainly of requirements as specified in the requirements. However, some features were also added to “Would have” because of their potential usefulness, but which would not be feasible within the current time or with use of the current technology. We recommend to add these features in the future if possible, since it would improve the user-friendliness of the application.

External access to application
Our system runs a HTTP webserver on the TV, so external devices can access our application by as a web application. This means that devices such as tablets, laptops and smart phones can interact with our application via HTTP. A use case scenario would for example be that the user searches for a video on a tablet and then streams it to the TV. This provides a lot more features which can improve the usability of the system.

Possibility to minimise app
It could be useful to be able to minimise to the application while downloading content, so other things can be done in the meantime. There are two main reasons we did not implement this. The first is that the TV is not powerful enough to support this. It takes too much processing power to run the application we built (mainly because Dispersy and DHT[6] are fairly heavy modules), so let alone running something else alongside. The second reason is that the Samsung framework does not provide this functionality; it is not possible to minimise a web application in the Samsung framework. If this becomes supported in the future -and if the TV's become more powerful-, this would be a nice feature to add.

Chapter 4

Design

This chapter describes the design of the application to be built, this design is based on the requirements gathered previously. The design has to incorporate solutions to the problems mentioned in section Problems. This means the design has to include classes for a back-end using swift, a frontend using Samsung's SDK, a sandbox workaround and a means for inter-TV communication. First we will discuss the current software archictecture on the TV. Then we will discuss an extended version of this architecture which will allow us to implement solutions for the aforementioned problems. Finally the class diagram and other UML models will be discussed.

4.1 Standard Samsung Software Architecture

As mentioned before in section Problems the Samsung SDK for the SmartTV only allows programmers to create applications using HTML, CSS, Javascript and Flash. All operations to be performed in the root file system will go via the SDK as a middle layer between our app and the root file system. This is because the Samsung SDK provides a set of methods in which this can be realised.

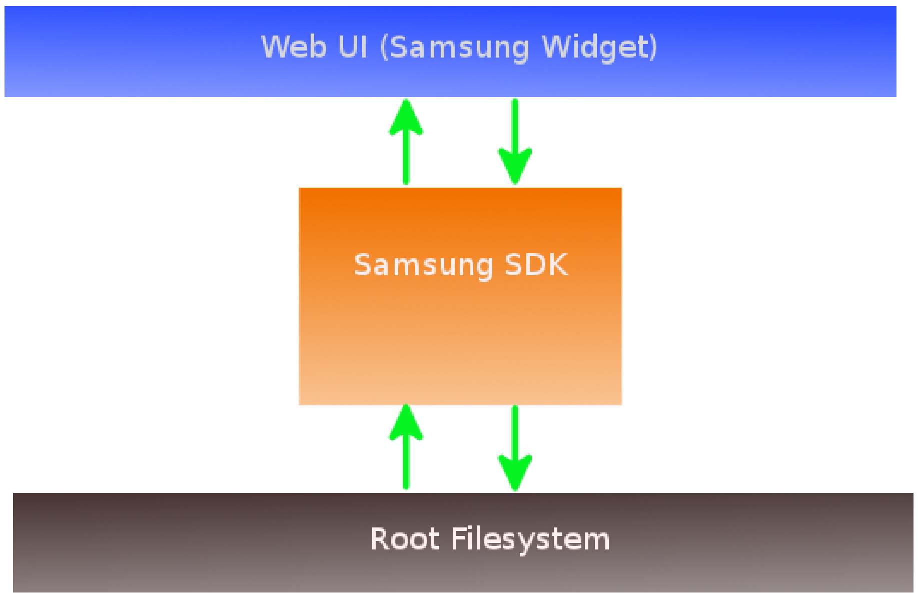


Figure 4.1: The original architecture created by Samsung.

4.2 Extended Software Architecture

The solution for being able to run custom made code has already been mentioned, namely getting root access to the TV. This was already done in the orientation phase of this project. This leaves only one problem, the sandbox. The frontend and back-end of our application need a way to communicate. It is not possible to perform direct calls to the back-end, because the Samsung SDK is completely sandboxed. Therefore we decided to use a networking approach, so the frontend and back-end will act as seperate applications, communicating via a network (client-server architecture).

In the Samsung SDK it is possible to send HTTP GET messages to any server and since we already have root access, it is possible to implement our own HTTP web server. This way one can create a bridge between the client software using Samsung's framework and our own made HTTP Server. Since the TV does not support HTML5 we were not able to use websockets which provide an open connection in which a reply can be received asynchronously at any time. Thus, we are restricted to the client-server architecture where you have to keep polling to obtain an updated value for the progress of a download for example or an updated list of search results.

The server is seperated in three subsystems. These are the HTTP server, which handles HTTP requests and acts as a controller, the Download Engine and the Search Engine, which makes use of dispersy and DHT implemented by the tribler team (which are, like libswift, fully-organised). This can also be seen in figure 4.2.

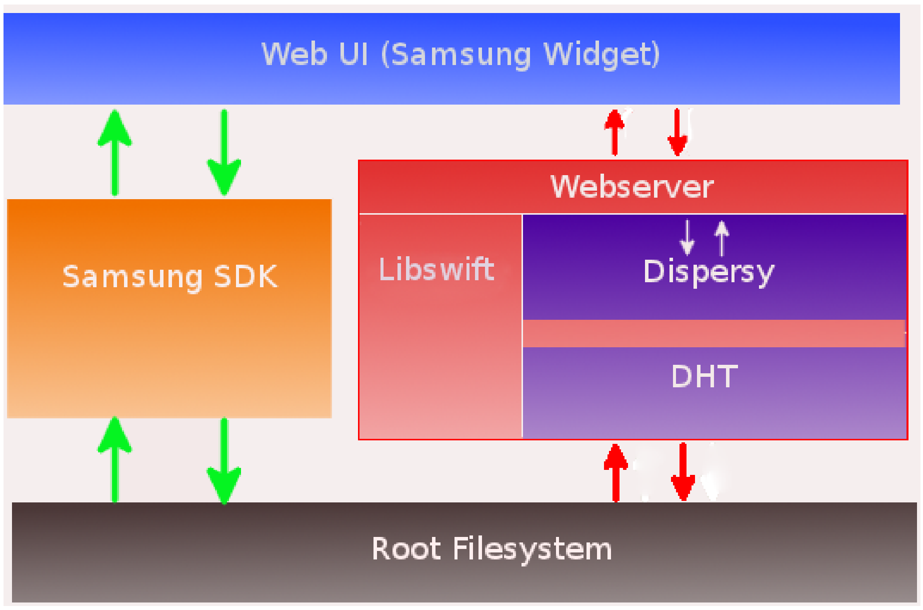


Figure 4.2: The adjusted architecture with our application. Our application is injected as a webserver daemon, here seen in red. The purple part is developed by the Tribler team, with which we have to communicate.

4.3 Class Diagram

Figure A.9 shows the class diagram for the webserver. These classes are then explained in more detail.

4.3.1 Download

A data structure to store all information regarding downloads from the swift engine and as an interface to the swift methods. This information can be accessed by the DownloadManager, which publishes the information of all Downloads to the web interface via the HttpServer class. This information can be retrieved by using swift methods. Core functionality of this class is based on the swift engine. In that sense, it is a wrapper class which makes use of swift methods.

4.3.2 Stream

This class serves as a data structure to store all information regarding streams from the swift engine and as interface to the swift methods. It is in essence the live-on-demand counterpart of the Download class, which is used to retrieve files from the Internet to store them on disk. Since there will always be only one stream opened at a time (assuming that people only watch one film at the same time), this class is designed according to the Singleton pattern.

4.3.3 DownloadManager

The DownloadManager, as the name implies, holds a list of Downloads and manages them. It can be accessed by the HttpServer, which controls this class whenever needed. It retrieves information from all Downloads and calls the methods of the correct Download. The DownloadManager also manages the Stream class. In order to save bandwidth, all Downloads are paused when a stream is opened. This is why the Stream class is also managed by the DownloadManager, because that way the DownloadManager is able to access both the Stream class and the Download class. For similar reasons as the HttpServer class, the DownloadManager is designed to be a static class.

4.3.4 SearchEngine

This class serves as the interface to the search functionality developed by the Tribler team. It starts dispersy and DHT, with which files can be sought on the net by calling the methods in the dispersy module. For similar reasons as the HttpServer class and DownloadManager class, this class is designed to be a static class.

Chapter 5

Implementation

The implementation of the application required the use of three different programming languages each with their own specific task. In order to create a GUI we were forced to use JavaScript as this is the only option Samsung has made available for creating apps. However, the peer-to-peer engine, libswift, is written in C++ which meant that no matter what, we would also have to run C++ code in harmony with JavaScript. As we would be running C++ anyway and due to the complexity of the core of the application we decided to build the core, which manages downloads, uploads and streams, in C++ as well. This made managing the libswift library easier than it would have been from JavaScript. Documentation for the C++ code can be found in the appendix

A language wrapper between C++ and JavaScript was not found, and could not have been used anyway because of the sandboxed environment. A solution we found to connect the JavaScript front-end with the C++ back-end was to use HTTP GET requests. By running an HTTP web server written in C++, a client-server communication mechanism could be built between the two languages. The JavaScript client can send HTTP requests to the server, and the server can perform the necessary function calls to the libswift engine and the other code necessary for the project. A small drawback is that the current firmware version of the tv does not have HTML5, which has support for websockets. Websockets remove the need of continuous polling, and allow the server to send responses(results) back whenever he's ready, without blocking the client.

In order to implement search functionality we made use of dispersy(message handling system used in Tribler) and a DHT (distributed hash table), both of which are written in python. In order to initialise both services and execute searches we needed a small amount of Python to manage dispersy and DHT. For this purpose a language wrapper was sufficient.The link between C++ and Python was provided by the Python/C API[1].

[1]`http://docs.python.org/c-api/`

5.1 JavaScript

The JavaScript front-end consists of 5 scenes (classes), namely Main, Browse, Downloads, Player and Settings. These scenes are the five graphical elements the front-end is able to switch to and from. In the Browse scene, functionality is built to: search remotely, browse usb devices, download or stream files, upload files and add files to a playlist. In the Downloads scene, the progress of downloads and uploads is visualized. In the Player scene, videos in the playlist can be played, and the streams can be watched. In the Settings scene some settings can be set, like the download path and limitations to upload and download speed. Most of this functionality is actually done by the C++ code, but in the scenes HTTP GET requests are sent to the webserver to execute the functions.

5.2 C++

The implementation of the C++ part ended up being quite different from the business class diagram, which can be found in the requirements chapter. The main reason for this is that the functionality of the filemanager could be done in JavaScript and as a result the necessity for that functionality in C++ dissapeared. There was also no need for a seperate download and upload class because in libswift a download and an upload are essentially the same thing. When you are downloading you are also automatically uploading and an upload is just a download at 100% completion. These changes meant that the end product looks a lot like the the class diagram in the design chapter.

5.3 Python

Implementing the Python part was as easy as it should be, since everything was already available from Tribler. The only thing needed to do was to call the methods we need and disable all other parts we do not use. Some patches were required in Tribler code, but were very easy to apply. The only file we made in Python was the DispersyInterface, which actually starts a Tribler session with only DHT and dispersy. To save resources, we were recommended to disable all the other parts of Tribler, since they are not used in our software.

5.4 Changes during development

During development it is almost inevitable that there will be changes from the original design. In this part we will discuss the largest changes we decided to make or were forced to make.

5.4.1 Software Improvement Group

Our code was evaluated by the Software Improvement Group[2], which checks the overall quality of software projects such as ours. During this project, we sent them our code twice. In general, they were satisfied with the quality of our code and documentation, meaning that our code is maintained well. Our only flaw was that certain methods were implemented in too many lines, making them complex. The solution to this was to break these methods up in submethods, which are then called in the original method. Also, we were not aware that the testsuite had to be sent as well, so they were not able to check our testsuite. At the time we sent our code for the first time to SIG, however, we already had a test suite testing the crucial parts of our system.

The second feedback confirmed our efforts in improving the code according to the guidelines the group gave us the first time. Even though our code scored better at some parts, it also scored less on other parts. Compared to the code we gave the first time, the code for the second feedback contained a lot of code which depend on each other. In other words, the coupling was a bit tighter than before. One of the reasons for this is that we extracted duplicate code from different files and implemented them in Settings.cpp, making Settings.cpp a generic file containing helper-functions. The result was that most files depend on Settings.cpp, because it contains a lot of generic functions. Even though our code grew a lot, we still managed to maintain the code well in general.

Their feedback can be found in the appendix.

5.4.2 Upload visibility

There were a number of options when it came to choosing how to decide which files would be uploaded. In the design we chose for a setting per file which could be set to true or false by the user which determined whether the file would be shared with others. Due to limited time and the fact that FileManager was no longer going to be implemented in C++ we decided to change the way users determine what should be uploaded.

All files in the default dowload folder automatically start uploading on startup. The user can browse to files on the tv and choose to upload them. Any files in the list of uploads and downloads can also be stopped and removed by the user to stop them from uploading. This implementation saved us some time by using functionality that was already implemented rather than needing to add another setting and having to remember whether a file has been set to visible or not.

5.4.3 Number of downloads

In the original design the plan was to allow the user to be able to download multiple files at the same time. This would be possible in principle and could

[2] `http://www.sig.eu/nl/`

easily be implemented in our application. However, in order to save processor power on the TV we decided to limit the user to one download at a time. The user is able to switch between downloads and the application automatically starts downloading the next download in the list after it has finished.

5.4.4 Excluded features

Due to limited time and hardware some features were not implemented at all so that we could focus more on the features we believed to be most important. One of these features was the ability to rename files which was a should have according to our design. However, we decided to drop this functionality as it is a very minor feature which would take a relatively large amount of time to implement. At the time there were more major features that needed to be implemented.

Another feature that was not implemented was sorting files by name, date, size and type. Just like renaming files this feature was not worth spending a lot of time on as it would not be a significant improvement to the application. Both of these features would be more important for a finished product but because we were focussing on creating a proof of concept other features took priority.

Chapter 6

Process

6.1 Planning

In the orientation report we proposed a planning where the workload was divided into different phases. We were able to keep up with this planning.

The first step was to create an environment in which to develop and test the application. This meant gaining root access to a Samsung television and cross compiling the necessary libraries for it. This was achieved before the project had officially started which allowed us to begin development immediately.

Our application heavily depends on libswift so the first goal was to be able to call libswift and use the library to stream and download files. This was achieved fairly quickly which meant we could start developing the fully functioning application. At the same time we also started development of the GUI in JavaScript. The main issue was creating the link between JavaScript and C++. Once we had settled for HTTP requests the development of the GUI and C++ side could be done in parallel although in the beginning the focus was on creating a command line controlled C++ application.

As soon as there was a basic C++ application we also began writing tests using gtest. This made further development easier and bug tracking faster. Once the C++ part of the application was as good as finished, the last phase was to implement search functionality which required Python to be run on the television in order to run dispersy and DHT. Getting Python, including all the required modules, working on the television did not take long. Soon after that we were able to use dispersy and DHT to return search results.

Once that all elements had been completed it was a matter of connecting them to create a fully functioning application. In the table below there is a more detailed week by week description of when what was done.

Week	Date	Tasks completed
0	Third quarter	• Gain root access to TV • Compile libswift for the TV • Make a demo video
1	30-04-2012	• Create orientation report • Design system architecture • Decide on communication method between JavaScript and C++ • Decide on message structure for HTTP requests • Create JavaScript structure
2	07-05-2012	• Make class diagrams and sequence diagrams • Make a video player in JavaScript • Get familiar with libswift • Create HTTP web server • Fix TCP RST bug • Implement download handler in HTTP server • expand HTTP server to support streaming • Run HTTP server as daemon
3	14-05-2012	• Fix forking and thread issues • Call the httpgw closecallback • Filebrowsing in JavaScript • Clean HTTP server code • Link HTTP server against streamer • Update SamyGO init scripts • IPtables only block our own RST packets
4	21-05-2012	• Create testsuite • Fix aspect ratio of the video player • Link swift against HTTP server • Implemented Download.cpp
5	28-05-2012	• Start implementing application as designed • Create DownloadManager • Make Streaming work as singleton • Create tests for Download.cpp

Table 6.1: Order of events

Week	Date	Tasks completed
6	04-06-2012	• Add exceptions • Handle mutliple downloads • make pasuing downloads possible • Add ability to build XML files • Add pseudo search functionality • Add mutex in DownloadManager • prepare code for SIG • Continue adding testcases
7	11-06-2012	• Add speedlimiting • Make switching between downloading and streaming possible • Get IP address dynamically in C++ • Start uploading files automatically on startup • Fixed bug when adding the same download twice • Created main menu in JavaScript
8	18-06-2012	• Create Utils • use FileTransfer in libswift to get statistics about downloads • make uploading on demand possible • Settings are automatically loaded at startup • Added Python binding in SearchEngine • Process SIG feedback • Parse XML files in JavaScript • Get Python working on TV • Get Dispersy working on QEMU • Get Dispersy working on TV and make searching possible • Get DHT working on the TV • Continue adding testcases

Table 6.2: Order of events

Week	Date	Tasks completed
9	25-06-2012	• Create progress bar in JavaScript • Clean JavaScript code • Create DownloadManager in JavaScript • Create mock for SearchEngine • Finish testsuite
10	02-07-2012	• Get IP address dynamically in JavaScript • Prepare code for SIG • Finish final report

Table 6.3: Order of events

See `http://youtu.be/cNaLfay73TE?hd=1` to view the development process of the application. Up until week 4, we committed everything under one name since we did everything together. Also, Jethro and Anass commited using the same account.

6.2 Division of work

In week 4, we had a simple, working application with the basic features. This was a good basis to further develop our application on, so we began to build the real application. In order to speed up the development, we started to divide the tasks so that we could work in parallel. Each time we encountered a problem or something we still had to implement, we wrote it down on sticky notes and put it on the whiteboard. 6.1 Whenever someone was done with his task, he could then take another task from the board.

Most of the time, we worked in groups of two people, but there were also times that we worked individually. As so, Anass led the development of the client side of the application. The server side was mostly implemented by Samuel and Wilson, while Jethro worked on different things (wherever help was needed). For example, even though we fixed the platform dependent problems together, Jethro was often the one in charge of fixing these problems.

6.3 Decision taking

The way we decided to implement certain features was rather simple. We first determined whether it was possible to implement the feature for the TV. If multiple solutions were proposed by the group, the complexity of the solution and the time it would take to implement it were taken into consideration. If all conditions were met, we just started to implement as much as possible.

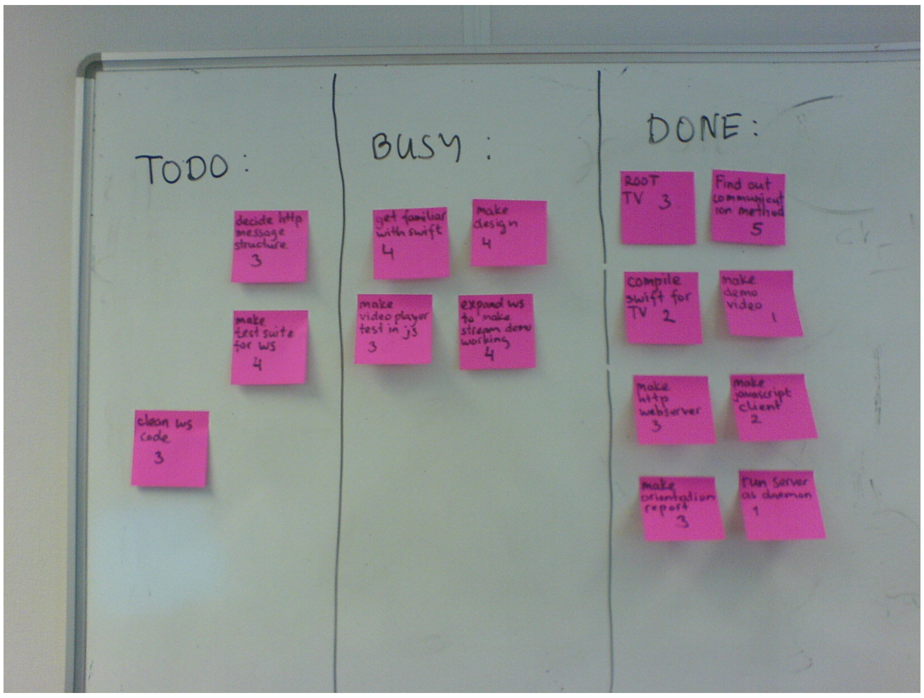


Figure 6.1: The whiteboard used to keep track of tasks.

6.4 Testing

We only wrote tests for C++ as this was the core of our application. We made use of the google test library which allowed for easy implementation of unit tests. The tests were developed in parallel with the development of the application which allowed for constant checking for bugs with every change. This way most bugs were immediately discovered and removed as soon as they appeared. Writing the tests also helped making the system robust and able to handle unexpected input and calls without crashing.

Every class has its own test class and almost every method has its own tests, usually a trivial test and additionally a couple of tests with unexpected input or unexpected situations. Certain classes, especially DownloadManager, required test cases to call multiple different methods in order to test certain situations. Although this resulted in some relatively long test cases it did guarantee a solid application capable of handling a large range of exceptional situations.

We chose not to fully test SearchEngine, the reason being that SearchEngine relies on dispersy and DHT. Both of these services are unreliable when it comes to returning search results, you do not always get the same results and sometimes you get no results. The testing environment requires a fixed return value every time. In order to achieve this we created a mock of the SearchEngine class which has the same functionality as the real SearchEngine except when it comes to searching. The mock always returns the exact same results which are hard coded in the mock. This allowed us to test HttpServer without having to worry about dispersy and DHT returning different values every time.

6.5 Problems encountered

This section focusses on all the important problems we encountered while developing the application for the TV. A brief explanation will be given per problem together with the solution we came up with. Some of these solutions were found with the help of the tribler team.

6.5.1 Rooting the television

The first hurdle before we could even start developing was gaining root access to the television. We needed this to be able to install libraries and execute code. Rooting the television should not have been an issue as there are tutorials and an app that roots the television for you from SamyGO[3]. The problem we ran into is that the televisions we got had a new firmware for which there was no way to root yet.

We had a quick look into finding a vulnerability in the new firmware that would allow us to execute arbitrary code and gain root access. We did find a possible vulnerability in ffmpeg, in the part used to decode Matroska video files, but developing an exploit for this would have taken too long as we did not have any experience in this field.

Another option was to downgrade the firmware to a previous version that would allow us to use the SamyGO app to gain root access. Samsung has not made the option of downgrading available, it is only possible to upgrade, so we had to trick the TV into thinking it was upgrading while actually it was installing old firmware. It is possible to install new firmware via a USB device which is what we initially tried. We took some old firmware, unpacked it, changed the version numbers inside, repacked it and recalculated the MD5 hash and changed that too so that they would match. Sadly, this is when we found out that the firmware also required a signature from Samsung which is something we did not have. This made downgrading via a USB device impossible.

We contacted SamyGO to ask whether there was any way to downgrade to an older firmware. Initially the answer was no but after some prodding they allowed us to make use of a server they had set up which pretends to be the Samsung firmware update server. Apparently the online firmware update did not require a Samsung signature or they had access to it. Either way, we now had a television with an old firmware which allowed us to run the SamyGO app and root the TV.

6.5.2 Installing missing dependencies

Even though the SmartTV runs a linux kernel, not everything was available from the start, so we had to install a great number of packages on the TV. For this purpose, we used cross-compiling toolchains to build executables compatible with the ARM [5] processors which resides in the TV. However, we failed to fully compile some of these packages, of which the Python interpreter is one of the most important. The Python interpreter was needed because the dispersy and DHT modules developed by the Tribler team were written in Python. Since we had to use these modules in our application, we also needed to install Python on the TV.

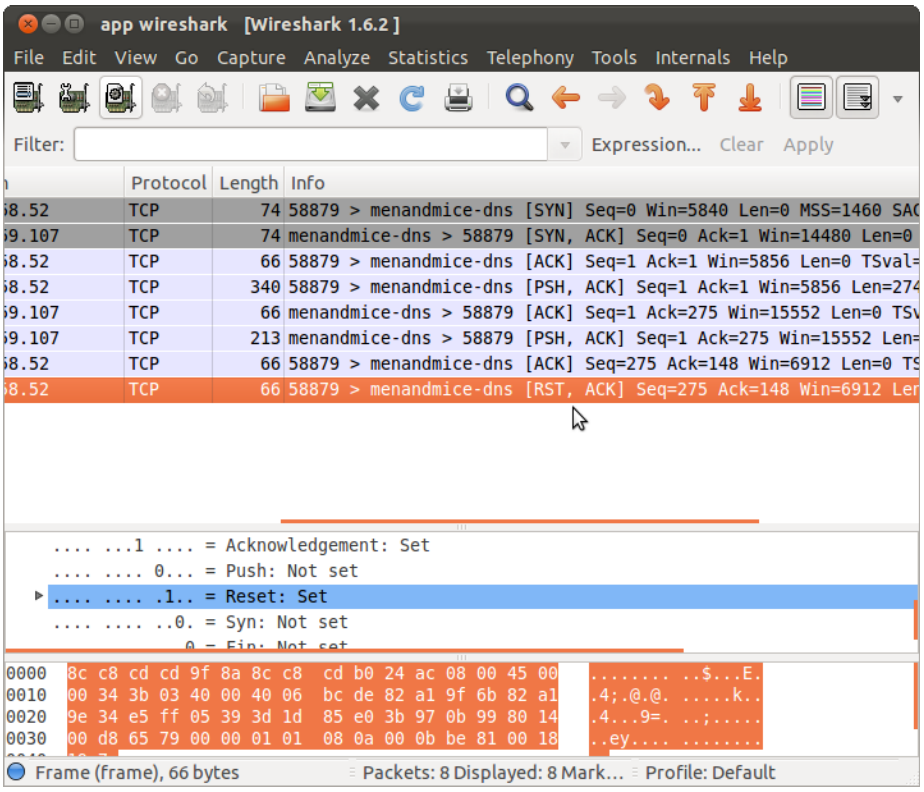
We succeeded in cross-compiling and running the interpreter on the TV, but not in cross-compiling the standard Python-modules we need to run simple Python executables. Our solution was to emulate the ARMv7 processor with QEMU [2]. Within this emulator, we installed a copy of Ubuntu, [4] which functioned as a binary compatible development platform and with which we were able to install the Python package with the apt tool. To achieve this, it was also needed to enable networking within QEMU. [11] Then we copied the binary files to the TV, after which we could succesfully run the Python interpreter together with all its dependencies.

Aside from Python, we also installed other applications via QEMU such as bash, sshd etc. to make it easier to work on the TV to accelerate the development.

6.5.3 TCP RST packages

It appeared that the Samsung TV sent TCP RST packages to itself after each HTTP request. TCP RST packages can be distinguished by looking at the fourth flag in the TCP header. This flag is the reset flag, which was activated in our case (see figure 6.2). Because of this, TCP connections to swift were interrupted each time a RST package was received, because the TCP connection was reset. This was a problem for file streaming, because the connection used by libswift to download the stream was continually interrupted, so the stream could not be played correctly.

We solved this problem by cross-compiling iptables[12] for the TV, so we could set up some rules to block these TCP RST packages. This required us to insert several kernel modules. Because this needs to be done each time the TV reboots, we added some rules to the init script of the TV so that these kernel modules are loaded automatically, together with the rules written for iptables (see appendix).


app wireshark [Wireshark 1.6.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
                Protocol Length Info
58.52           TCP          74 58879 > menandmice-dns [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SAC
59.107          TCP          74 menandmice-dns > 58879 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0
58.52           TCP          66 58879 > menandmice-dns [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=
58.52           TCP         340 58879 > menandmice-dns [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=274
59.107          TCP          66 menandmice-dns > 58879 [ACK] Seq=1 Ack=275 Win=15552 Len=0 TSv
59.107          TCP         213 menandmice-dns > 58879 [PSH, ACK] Seq=1 Ack=275 Win=15552 Len=
58.52           TCP          66 58879 > menandmice-dns [ACK] Seq=275 Ack=148 Win=6912 Len=0 TS
58.52           TCP          66 58879 > menandmice-dns [RST, ACK] Seq=275 Ack=148 Win=6912 Len
```

```
  .... ...1 .... = Acknowledgement: Set
  .... .... 0... = Push: Not set
▸ .... .... .1.. = Reset: Set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
```

```
0000  8c c8 cd cd 9f 8a 8c c8  cd b0 24 ac 08 00 45 00   ........ ..$...E.
0010  00 34 3b 03 40 00 40 06  bc de 82 a1 9f 6b 82 a1   .4;.@.@. .....k..
0020  9e 34 e5 ff 05 39 3d 1d  85 e0 3b 97 0b 99 80 14   .4...9=. ..;.....
0030  00 d8 65 79 00 00 01 01  08 0a 00 0b be 81 00 18   ..ey.... ........
```

Frame (frame), 66 bytes   Packets: 8 Displayed: 8 Mark...   Profile: Default

Figure 6.2: Screenshot of wireshark. The RST package is highlighted in orange. Also, in the lower part of the screen it can be seen that the RST flag is set.

6.5.4 Fork() and thread problem

For testing purposes, we used a webserver written in C++ we found on the Internet. We were not aware, however, that this webserver was implemented so that it would call the fork() method each time a HTTP request was received. We were using p_threads[7] to start several libevent[1] loops needed for downloading files and p_threads cannot access data directly within other processes. So we got the problem that the threads were not able to access the correct data.

The problem was that fork() creates a new process, which is an exact copy of the parent process (See figure 6.3). So all global variables and threads running within the process were also copied. Now, each time we tried to access a global variable, we noticed that the values of the variables were inconsistent because we were setting and reading the wrong variables. Unconsciously, we would set the value of a global variable, but read out the value of its copy instead, which has not been set yet. This caused unexpected behaviour of our application, on which we spent a lot of time. Strangely enough, this problem did not occur when we compiled and ran the program for our own pc's, but only when we tried it on the TV.

A solution is to implement pipes, which provide inter-process communication. With this, threads are able to access data outside their own process scope so the correct variables can be read.

In the end, we solved this problem by implementing a new HTTP server by ourselves, which makes use of the libevent library. This version of the webserver does not call the fork() method, so we did not have the problem of threads not being able to access the same data.

6.5.5 Communication between JavaScript and C++

One of the first issues we encountered was how to link JavaScript to C++ so that we could call C++ functions from within the GUI and get return values back. Initially we wanted to use a language binder like SquirrelFish or a JavaScript engine for C++ like google V8. However due to the restrictions set by Samsung you are forced to develop the JavaScript app using their SDK and it is impossible to add any additional tools. This left us with the standard JavaScript functionality and the Samsung API. As Samsung does not allow any development outside of JavaScript there was no support for calling C++ applications in their API either. The only remaining option was to use HTTP requests which is part of the standard functionality in JavaScript.

This did require us to be running an HTTP server in C++ which needs to be running before the Javascript app had even been started. Newer version of the firmware also support HTML5 and websockets. This would allow for better two way communication betweem JavaScript and C++ rather than JavaScript constantly polling the HTTP server when it needs updates. The downside of

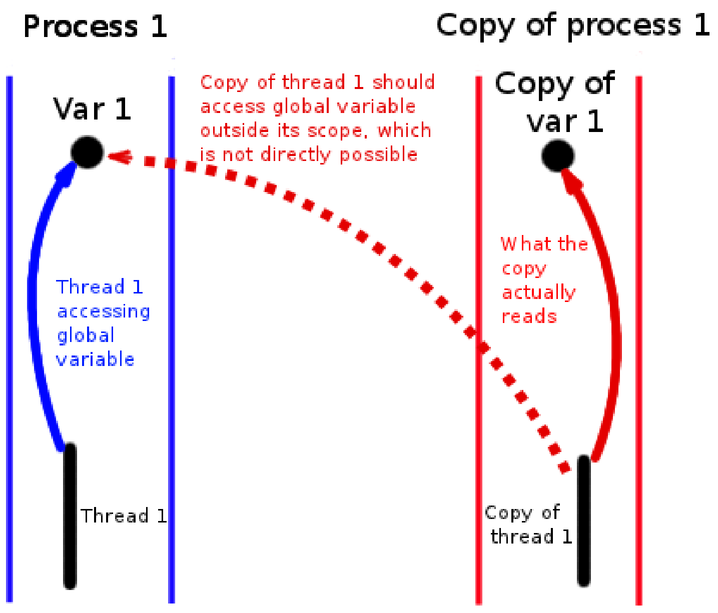

Process 1
Copy of process 1
Var 1
Copy of thread 1 should access global variable outside its scope, which is not directly possible
Copy of var 1
What the copy actually reads
Thread 1 accessing global variable
Thread 1
Copy of thread 1

Figure 6.3: The fork()-thread problem. Threads are accessing the wrong data.

this newer firmware is that there is no known way to gain root access yet which is why we could not use it.

6.5.6 Starting the HTTP server

Sending HTTP requests was the only way to communicate between JavaScript and C++ but there was no way to start the HTTP server in C++ from JavaScript. We needed to start the HTTP server as soon as the television was switched on. In principle it would have been possible to run the HTTP server as a daemon on start-up but it was not possible to edit the Samsung init scripts. SamyGO, however, also uses init scripts when they root the television which we could edit. Now whenever you root the television using SamyGO you also automatically start running the HTTP server for our application. This does mean that running SamyGO is required to be able to use our application.

6.5.7 USB write speed

Initially we were planning to use a USB storage device to store downloads and streams. As it turned out, the write speed of the TV to the USB device was so slow that streaming was not possible when using it to store the data. Downloading to an external USB device was also so slow that it was not practical to do so. This left us with only the internal memory of the TV for downloads and streams which is a very limited amount. This caused large files to be impossible to stream as they would fill the memory and cause the TV to hang. The maximum file size that can be downloaded is also very low as a result of this.

Chapter 7

Results

This chapter discusses the results of our project. A description of the built application will be given, alongside with screenshots. Also, graphs will be used to show the performance of our application.

7.1 SwiftTV

Since we built a download and stream application for the Samsung SmartTV based on libswift, we dubbed our application SwiftTV. The name is kept simple because other existing applications based on libswift have similar names. This section explains how SwiftTV works. Bigger screenshots can be found in the appendix.

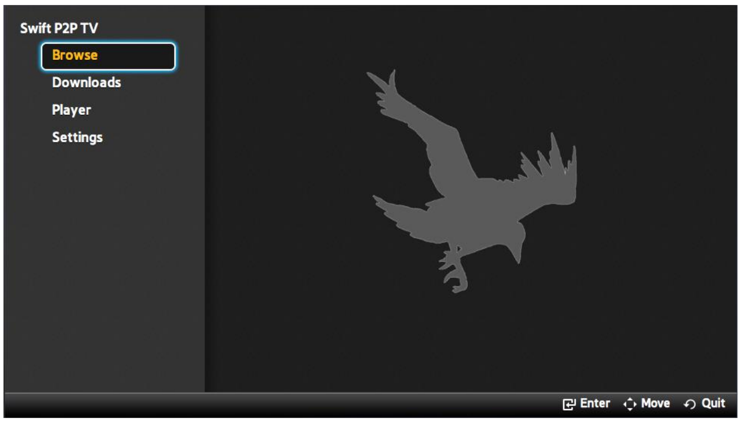


Figure 7.1: Screenshot of the Main scene.

The main menu points to the four different pages in the application, browse, downloads, player and settings. When selecting one of these pages the main menu remains visible.

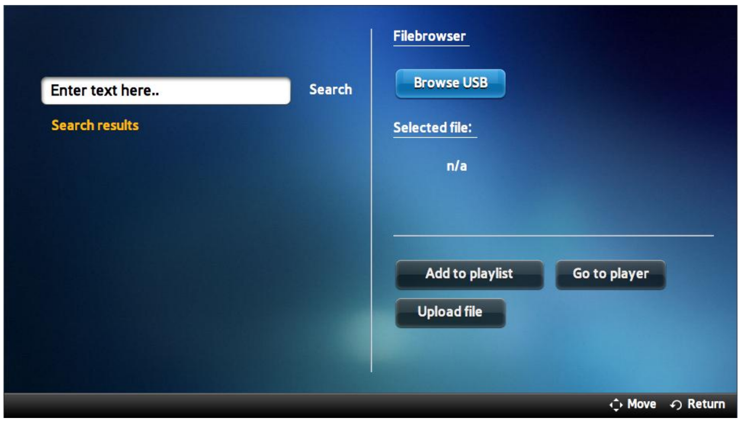
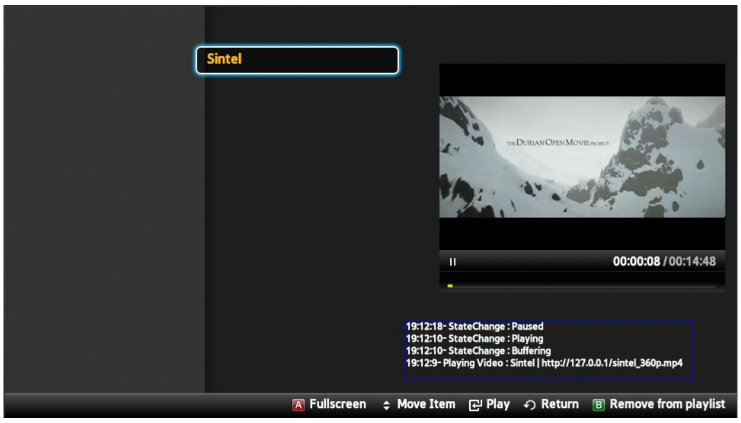


Figure 7.2: Screenshot of the Browse/Search scene.

Browse is where you can either browse your local filesystem to play a media file or search the internet for files to download or stream. Browsing the local filesystem is done with features included in the Samsung API and is therefor all done in JavaScript. When choosing a file from the browser it can be added to the playlist in the player page to be played. The search functionality sends a "/search:searchTerm" request to the HTTP server which in turn starts a search using Dispersy. After some time the search results are requested by JavaScript using a “/results” request. The results are returned in the form of an XML file and shown on screen.

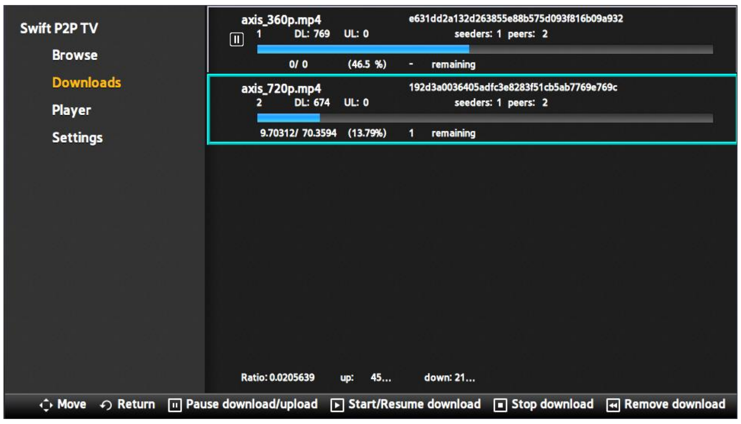
The user can select one of these results and choose to either download ot stream it. JavaScript either sends a “/add:hash” or “/stream:hash request” depending on the choice. If download is chosen the the download is added to the list in DownloadManager and start downloading. If stream is chosen a stream is opened and is added to the playlist on the player page.



```
19:12:18- StateChange : Paused
19:12:10- StateChange : Playing
19:12:10- StateChange : Buffering
19:12:9- Playing Video : Sintel | http://127.0.0.1/sintel_360p.mp4
```

Figure 7.3: Screenshot of the Media Player scene.

Player is where videos and streams can be viewed. To view videos or streams they must first be added to the playlist while in the browser. Videos will start playing in the small screen but full screen can be enabled afterwards.



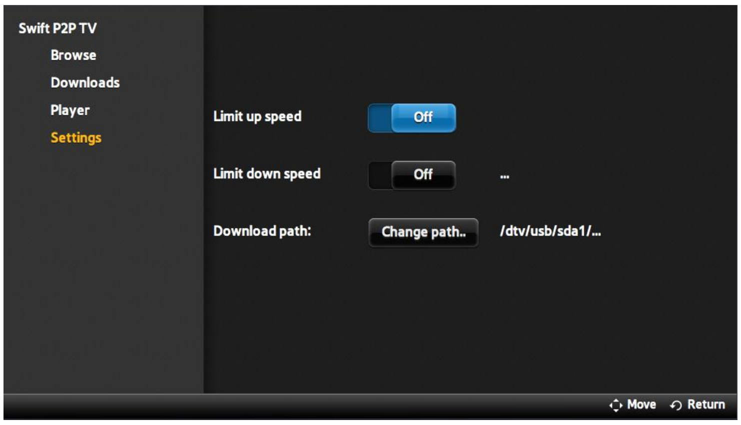
axis_360p.mp4 e631dd2a132d263855e88b575d093f816b09a932
1 DL: 769 UL: 0 seeders: 1 peers: 2
0/ 0 (46.5 %) - remaining

axis_720p.mp4 192d3a0036405adfc3e8283f51cb5ab7769e769c
2 DL: 674 UL: 0 seeders: 1 peers: 2
9.70312/ 70.3594 (13.79%) 1 remaining

Ratio: 0.0205639 up: 45... down: 21...

Figure 7.4: Screenshot of the Downloads scene.

Downloads is where the current list of downloads can be viewed. Downloads can be paused, stopped and resumed. JavasScript will send "/pause:hash", "/remove:hash" and "/resume:hash" requests to do this. When stopping a download it will be removed from the list.



Limit up speed Off

Limit down speed Off ...

Download path: /dtv/usb/sda1/...

Figure 7.5: Screenshot of the Settings scene.

In the settings page users can set their maximum download and upload speed as well as the download location. These settings are saved in memory and loaded on startup. When the settings are changed JavaScript sends a "/settings:upspeed:downspeed:downloadpath" request to the HTTP server.

7.2 Measurements

For this section, we measured the CPU and memory usage of the application against the time. This way, we can see how much resources our application needs. The measurements were done on a Samsung D7000 TV, [1] while playing a stream. The Sintel videoD was used for the tests. Several measurements were made, where we varied the download speed limits for the streams. Then the same measurements were done for the total CPU and memory usage, so we can see how much resources the TV needs in total while running our application. Also, measurements were taken while the TV was being idle, to show the exact differences between normal circumstances and situations that require more CPU and memory resources.

Figure 7.6: Screenshot of the video used. Video and screenshot are made by the developers of the Sintel project.

[1]`http://www.samsung.com/uk/consumer/tv-audio-video/television/led-tv/UE40D7000LUXXU`

7.2.1 Memory/CPU usage of SwiftTV

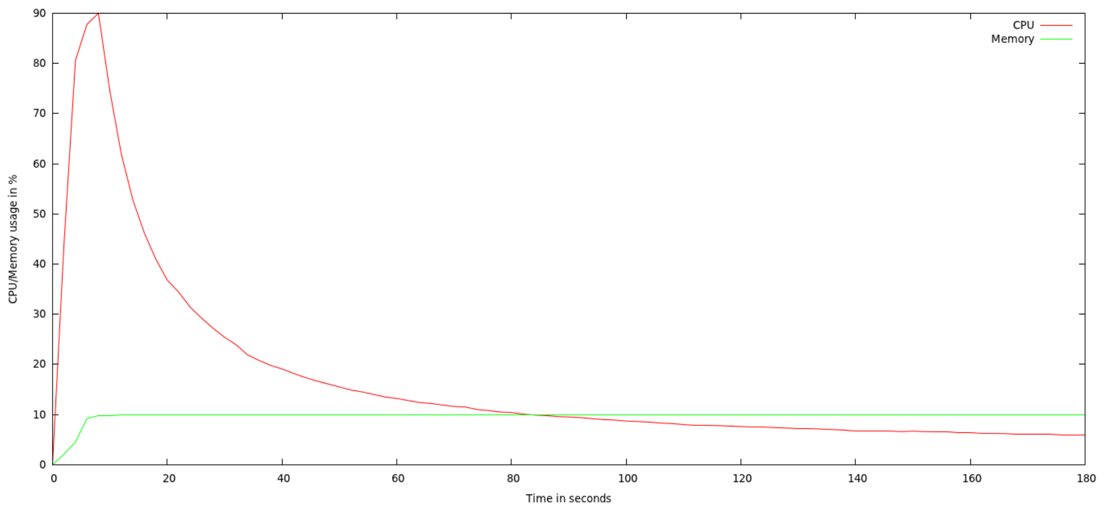


Figure 7.7: CPU and memory usage of the application, while being idle.

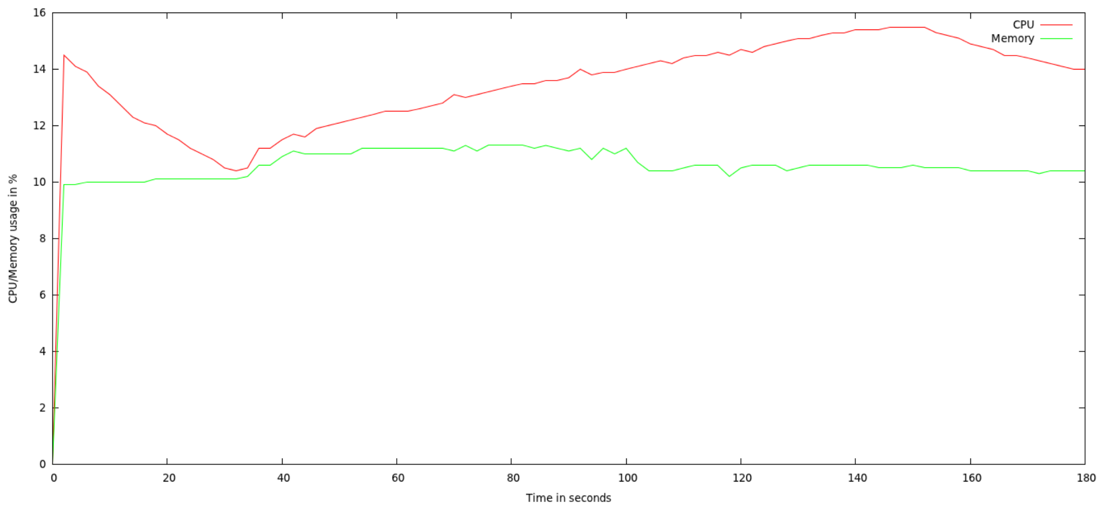


Figure 7.8: CPU and memory usage while streaming at 500 kB/s. The stream was stopped around 140 s.

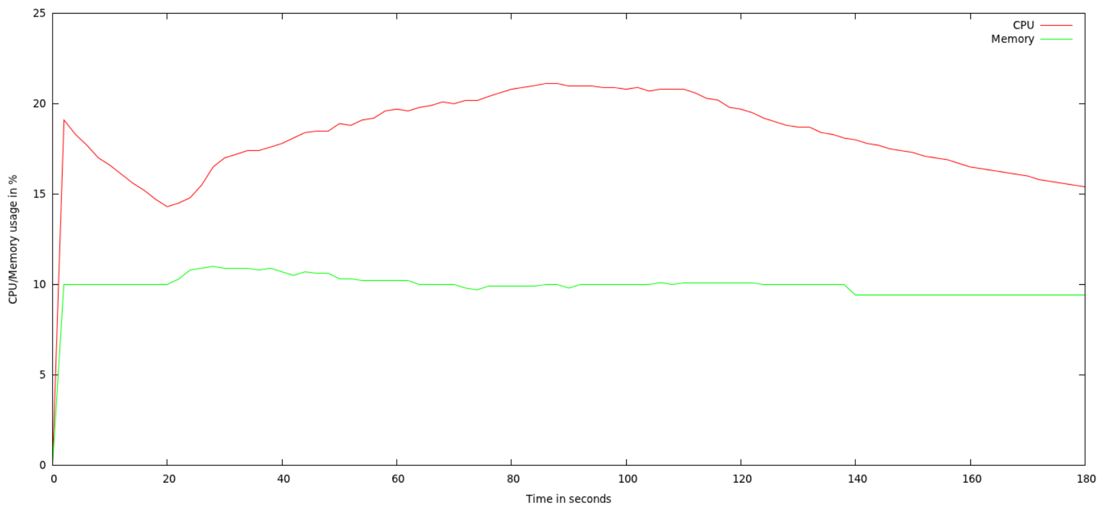


Figure 7.9: CPU and memory usage while streaming at 700 kB/s. This is approximately the bitrate of the video used. 7.6 The stream was stopped around 110 s.

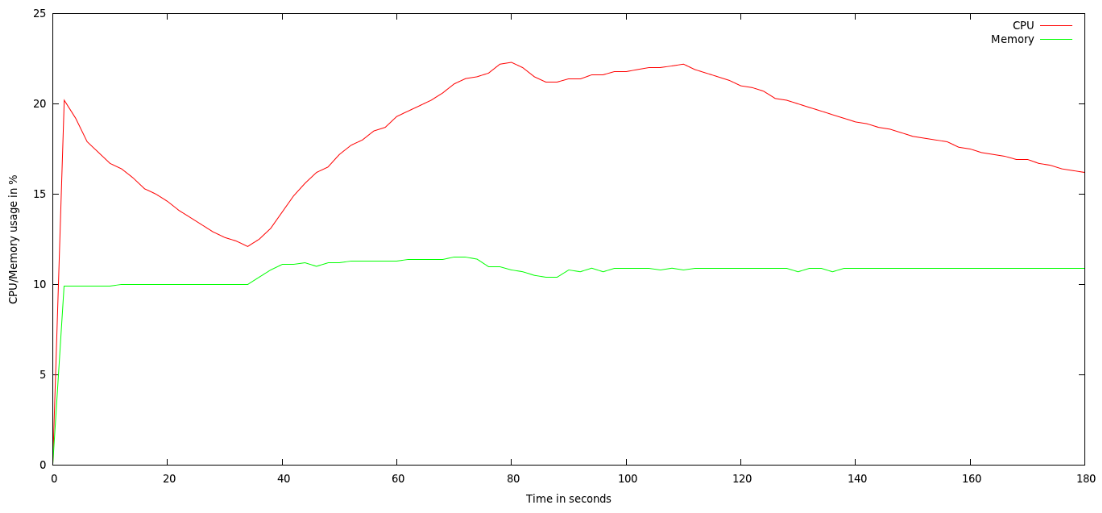


Figure 7.10: CPU and memory usage while streaming at 1 MB/s. The stream was stopped around 110 s.

Figure 7.7 shows the CPU and memory usage of only the application, while being idle. The peak in the beginning depicts the initialisation of the application, where modules such as Dispersy are started up. Figure 7.8 shows the CPU and memory usage while streaming the video at 500 kB/s. It can be seen that the CPU usage is always under 16%, while this upper limit grows with the bitrate we set, as shown in 7.9 and 7.10.

As expected, the lower the bitrate, the lower the CPU usage. However, the memory usage always stays the same, even when the application is idle. This goes against our expectations of memory usage of getting progressively higher the faster you stream. This means that our goal of limiting the memory usage to 256 MB has been achieved, since the TV has 350 MB of RAM, while we are only using 10% of it. Our goal of having a response time of 300 ms has not been achieved, however. From our own experience, we see that the application responds too slow (between 1 and 2 seconds). Even though we allowed the program to use 100% of CPU resources, we see that this was not even necessary since it only uses up to 23% when streaming at 1 MB/s.

Chapter 8

Conclusion

The constant struggle against the constraints of the TV was one of the things that made the project challenging and enjoyable. This is one of the reasons we started the project in the third quarter, working part-time to set up a TV ready to use in the fourth quarter. Developing on a device that nobody has really developed for before and having to solve new and unexpected problems along the way made this a very educational project. In the end, we succeeded in implementing a peer-to-peer application that runs natively on the TV, unlike other solutions using for example a set-top box. [1] With this, we are the first in the world to have implemented a fourth generation peer-to-peer application that runs natively on a TV.

Based on the final product it is safe to say that peer-to-peer applications involving streaming, downloading and uploading of files are possible on smart TV's. Due to hardware limitations the application was never able to reach its full potential but the results are very promising for future televisions with better hardware. Despite the hardware issues, the technology does work which makes our application the first fourth generation peer-to-peer file sharing application on a smart tv in the world.

Another important aspect, next to better hardware, as to whether more advanced applications will be developed for televisions is the amount of freedom manufacturers are willing to give to developers. As it stands, television manufacturers seem very wary of giving any access to their TV's, which is very understandable from a security point of view. However, if the full potential of the smart TV is to be reached, developers must be given full access to develop advanced programs or else the smart TV will remain a gimmick with limited functionality.

[1] `http://durao.net/2010/06/01/oplay-hdp-r1-%E2%80%93-bittorrent-client-with-firmware-1-27/`

Chapter 9

Recommendations

A major limitation during this project was the hardware in the television. The application we built depends on reasonably heavy python programs and requires quite a lot of memory when streaming. Although the CPU seemed to be able to keep up, the lack of memory was very noticeable when streaming. Thankfully, as technology improves and becomes cheaper, televisions will get better hardware and it will not be long until they will be able to easily run application like this one. For future versions of peer-to-peer applications on televisions, better hardware is a must.

Once televisions are powerful enough to handle high quality video streaming, a better version of the application can be created. Although we were very happy with the back-end of our application, the GUI part was not as refined because our focus was on the back-end. In order to make the application more user friendly the GUI needs to be improved. Thanks to the way our application is made, any GUI that can send HTTP requests and parse XML is able to interface with our application which makes changing to a completely different GUI very easy if necessary.

At the moment Dispersy and DHT are built into Tribler. When we needed both these services we were forced to run Tribler with a bunch of modules switched off, just leaving Dispersy, DHT and some other necessary elements. It would be better if Dispersy and DHT could be run as independent modules. This would save a lot of resources that unecessary parts of Tribler may be using.

Appendix A

UML Models

A.1 Business Class Diagram

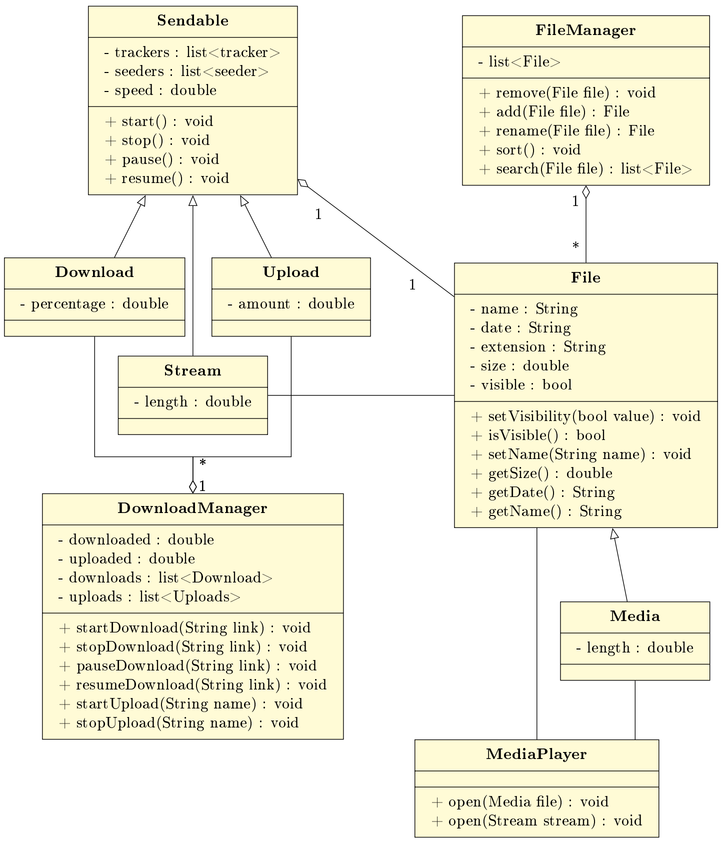


Figure A.1: Part of the class diagram. This part only shows the business classes comprehensible for end-users.

A.2 Use Case Dynamic Models

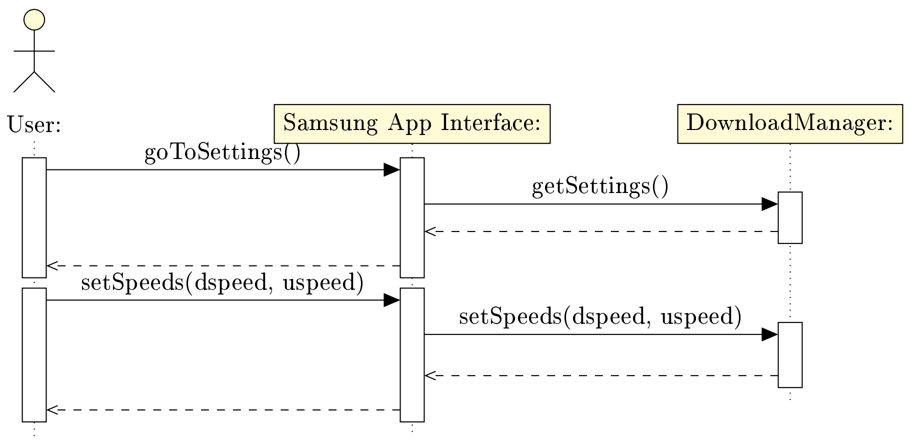


Figure A.2: Use Case 1: Limit upload/download speeds.

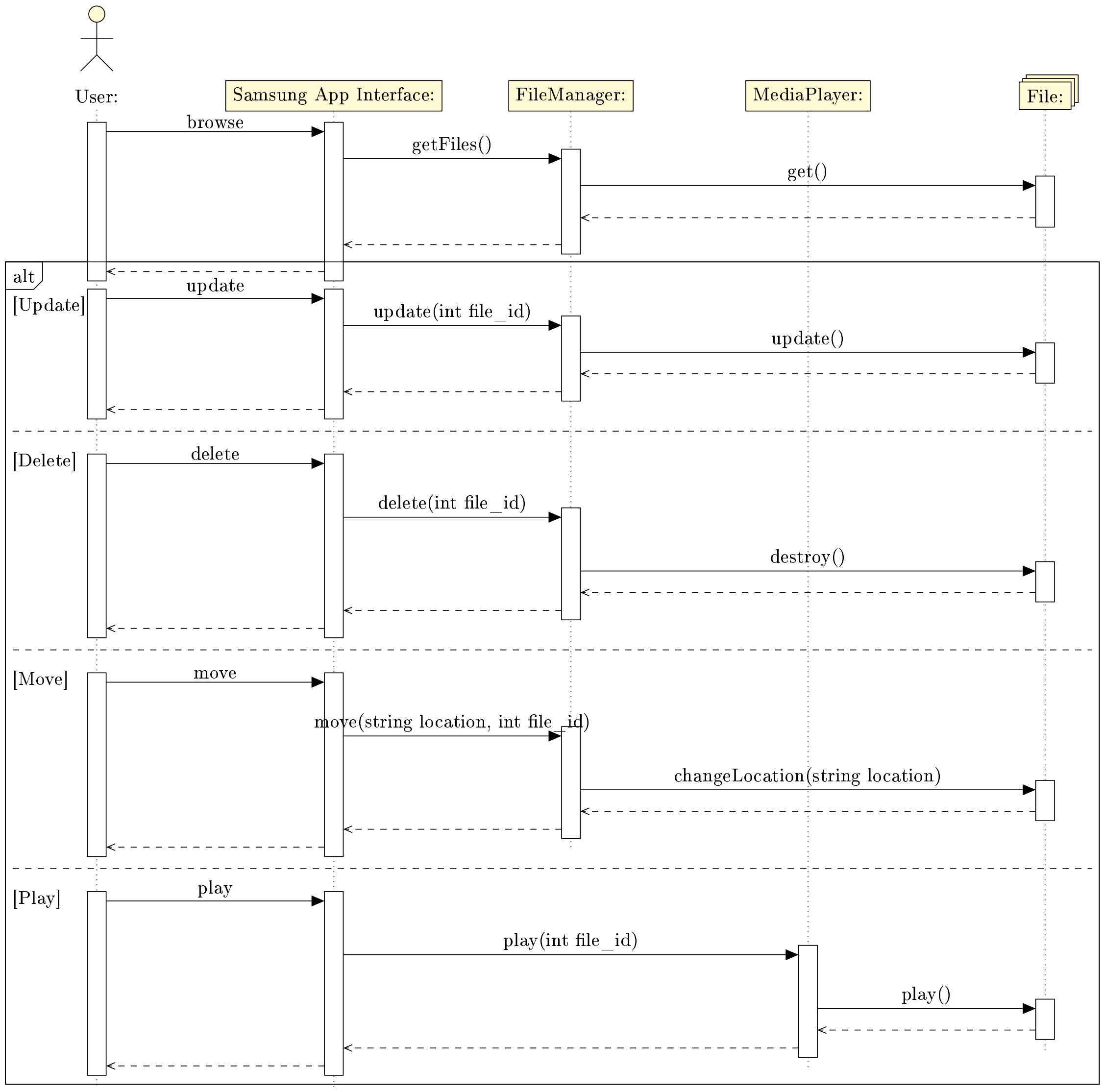


Figure A.3: Use Cases 2 to 6: Basic browsing and file manipulation

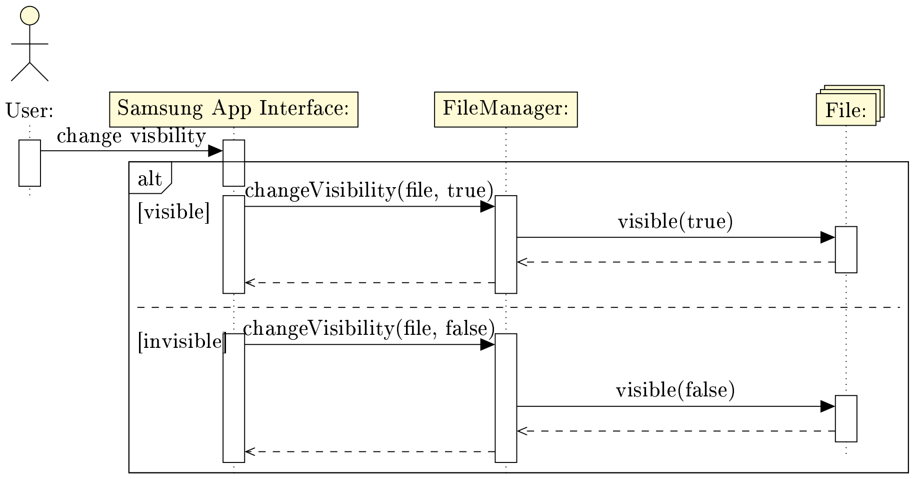


Figure A.4: Use Case 7: Change File Visibility

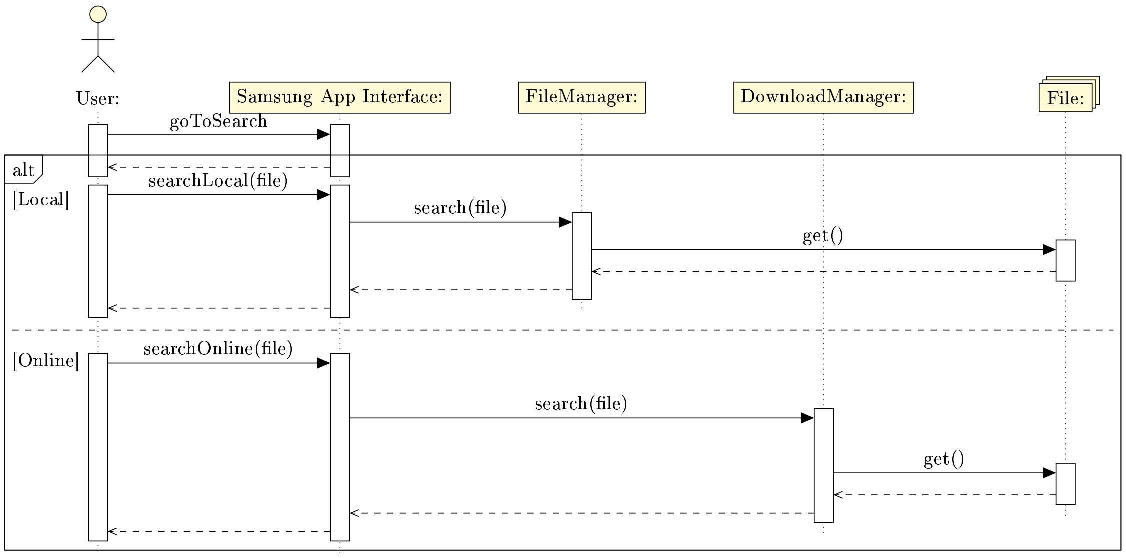


Figure A.5: Use Cases 8 to 10: Search.

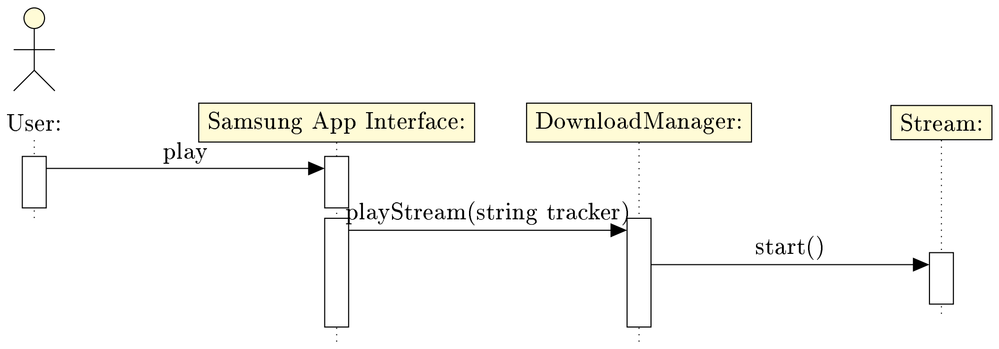


Figure A.6: Use Case 11: Stream File.

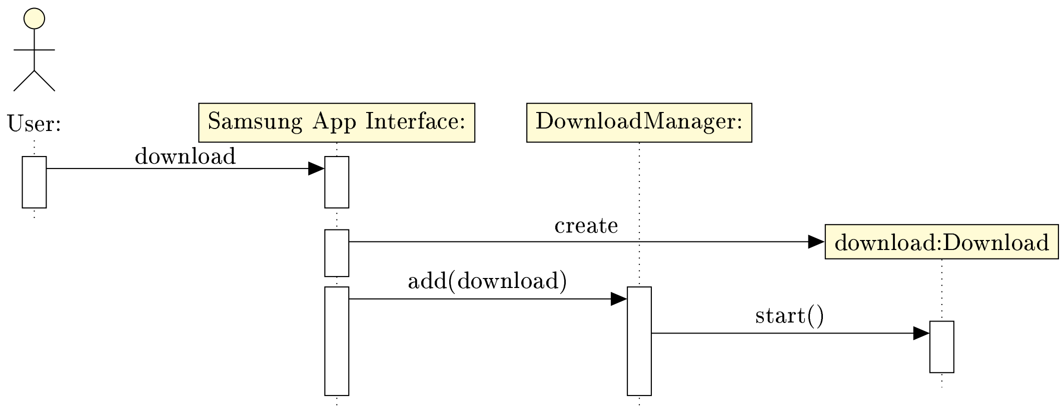


Figure A.7: Use Case 12: Download File.

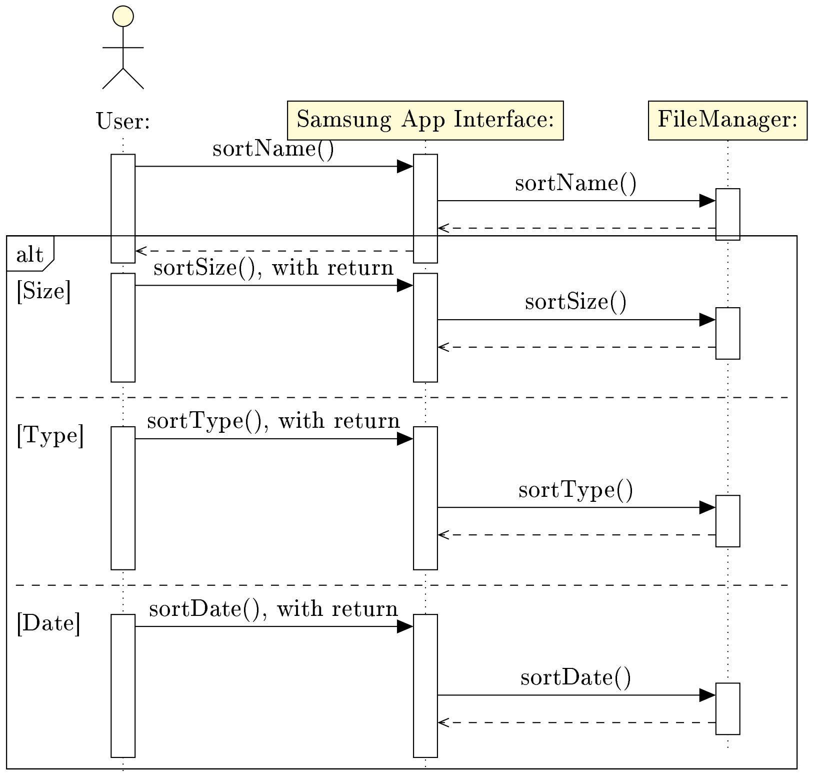


Figure A.8: Use Cases 13 to 17: Sort Files.

A.3 Class diagram

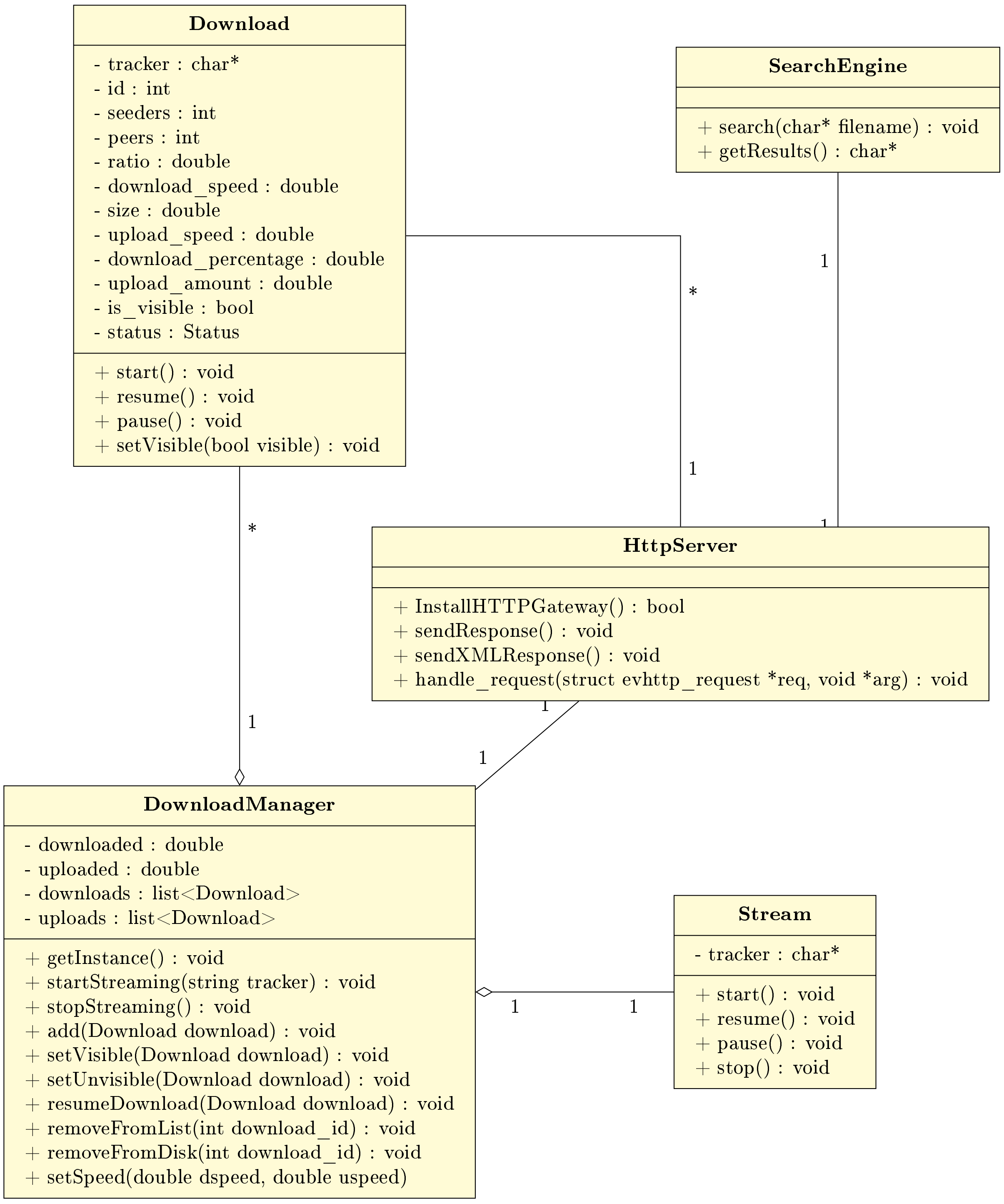

Download
- tracker : char*
- id : int
- seeders : int
- peers : int
- ratio : double
- download_speed : double
- size : double
- upload_speed : double
- download_percentage : double
- upload_amount : double
- is_visible : bool
- status : Status
+ start() : void
+ resume() : void
+ pause() : void
+ setVisible(bool visible) : void
SearchEngine
+ search(char* filename) : void
+ getResults() : char*
HttpServer
+ InstallHTTPGateway() : bool
+ sendResponse() : void
+ sendXMLResponse() : void
+ handle_request(struct evhttp_request *req, void *arg) : void
DownloadManager
- downloaded : double
- uploaded : double
- downloads : list<Download>
- uploads : list<Download>
+ getInstance() : void
+ startStreaming(string tracker) : void
+ stopStreaming() : void
+ add(Download download) : void
+ setVisible(Download download) : void
+ setUnvisible(Download download) : void
+ resumeDownload(Download download) : void
+ removeFromList(int download_id) : void
+ removeFromDisk(int download_id) : void
+ setSpeed(double dspeed, double uspeed)
Stream
- tracker : char*
+ start() : void
+ resume() : void
+ pause() : void
+ stop() : void
*
1
*
1
1
1
1
1
1
1

Figure A.9: Class diagram of the webserver subsystem.

A.4 Dynamic models

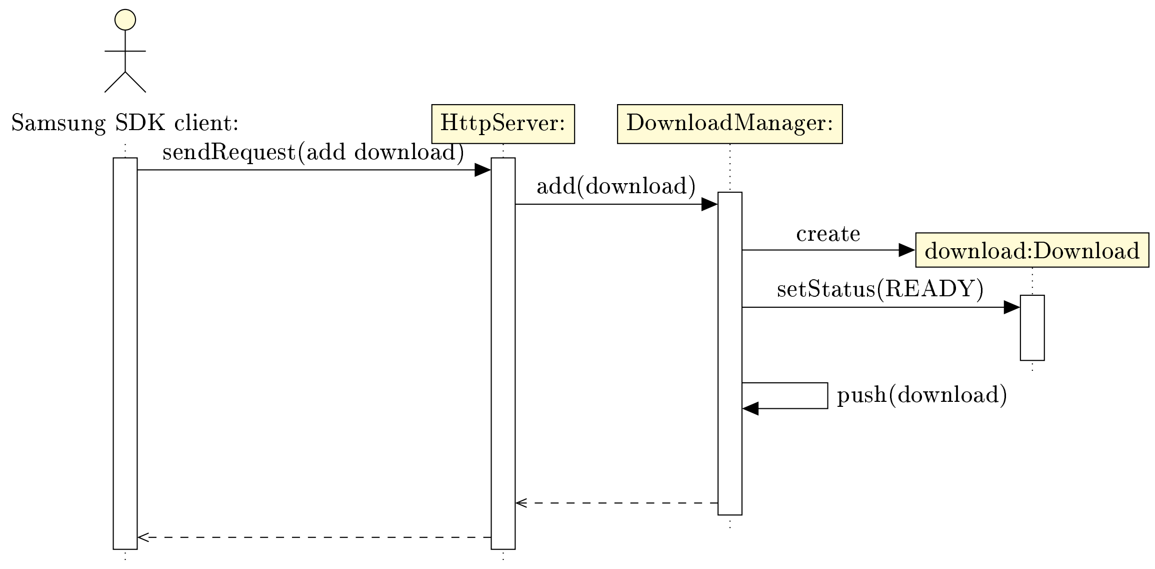


Figure A.10: Add download

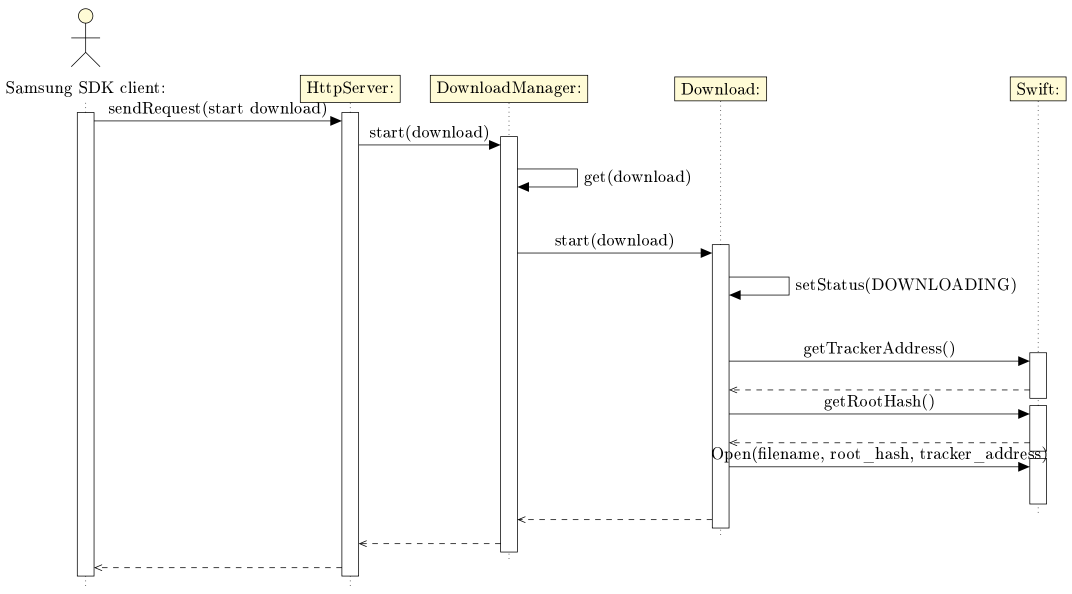


Figure A.11: Start download

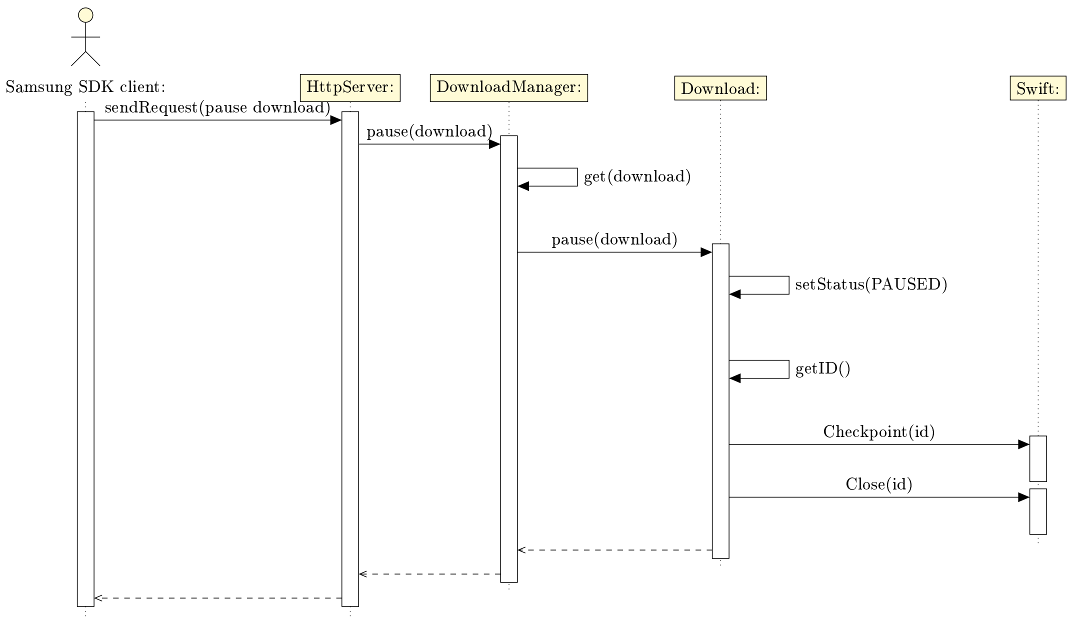


Figure A.12: Pause download

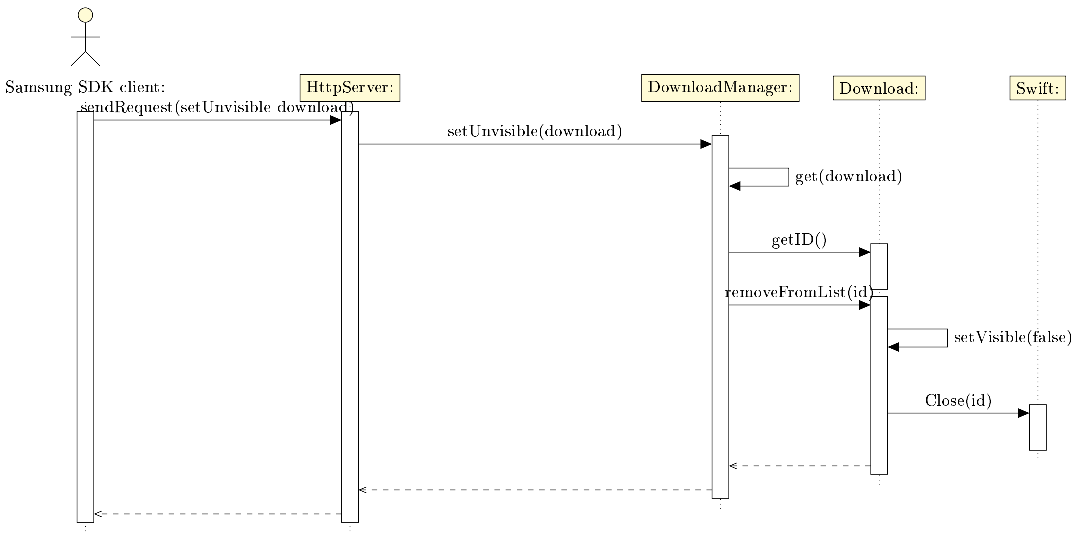


Figure A.13: Set unvisible

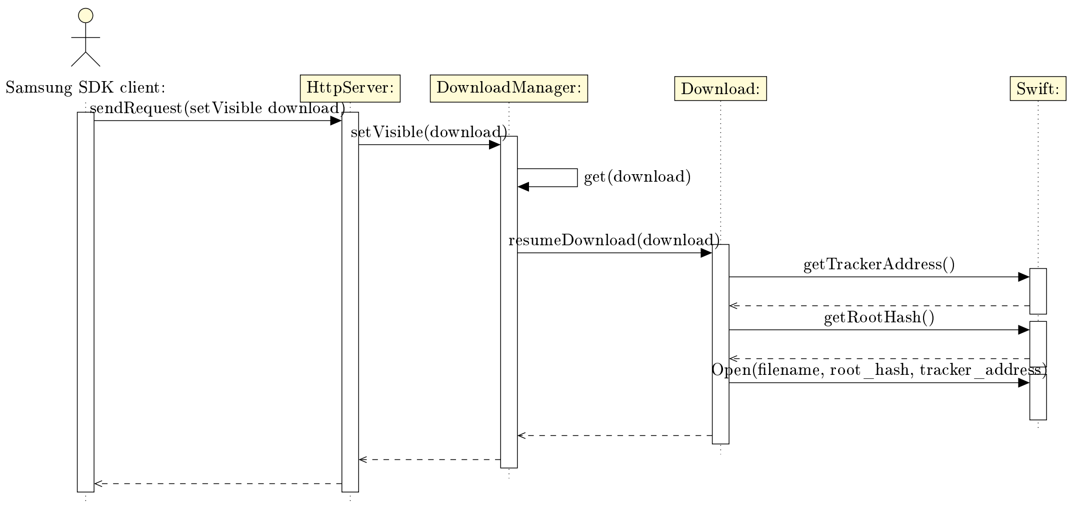


Figure A.14: Set visible

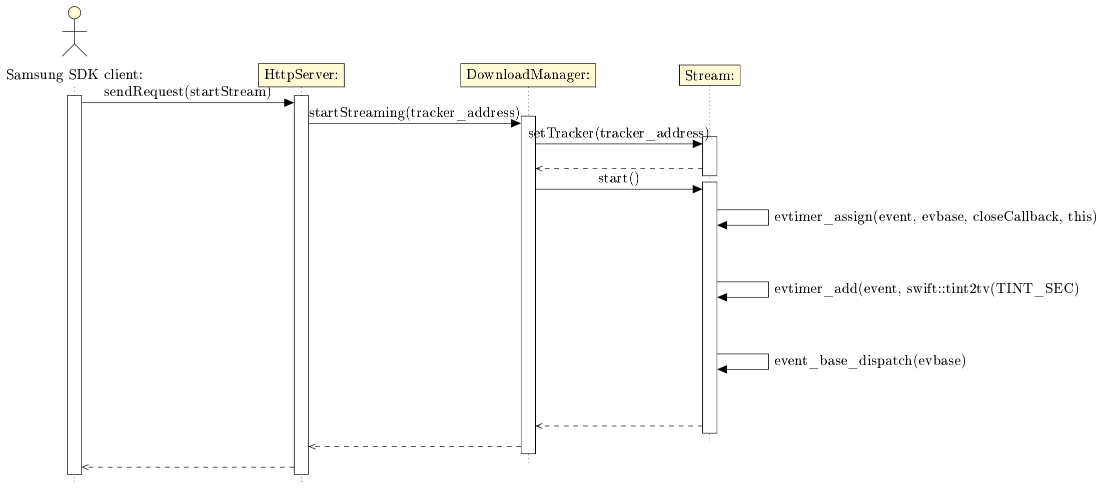

Samsung SDK client:
HttpServer:
DownloadManager:
Stream:
sendRequest(startStream)
startStreaming(tracker_address)
setTracker(tracker_address)
start()
evtimer_assign(event, evbase, closeCallback, this)
evtimer_add(event, swift::tint2tv(TINT_SEC)
event_base_dispatch(evbase)

Figure A.15: Start streaming

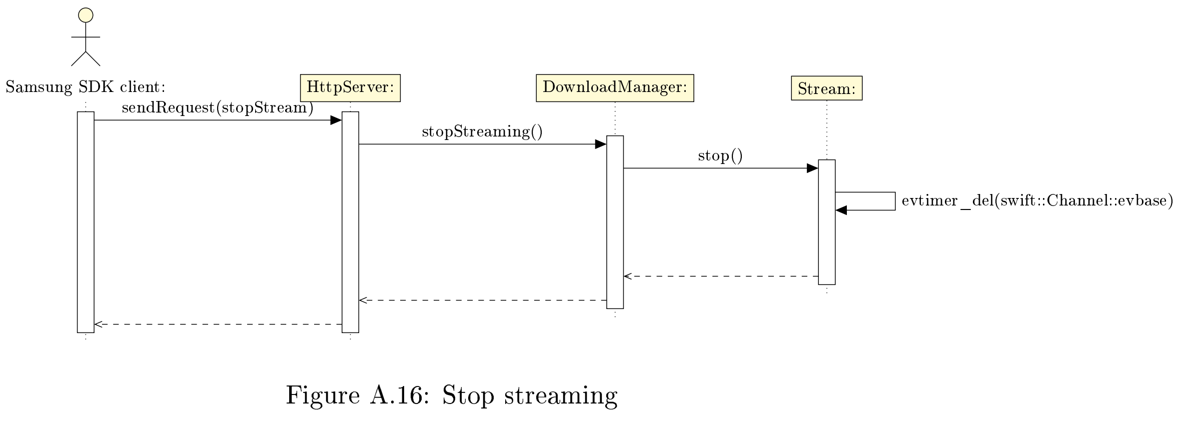


Figure A.16: Stop streaming

Appendix B

SIG feedback

B.1 First feedback

[Aanbevelingen] De code van het systeem scoort vier sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en de Unit Complexity.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de extreem lange methodes in dit systeem, zoals bijvoorbeeld de 'HttpServer::handleRequest'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// Message will look like: "/pause:roothash"' en ' // Some garbage collecting.' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Ook aan de kant van de 'swift_p2ptv' zijn langere methoden te vinden, bijvoorbeeld 'ScenePlayer.prototype.handleKeyDown'. In dit type methoden is het aan te raden om de afhandeling van een bepaalde 'keyCode' in een aparte methode te zetten, dit maak het makkelijker om de afzonderlijke stukken functionaliteit te begrijpen en te testen. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

B.2 Second feedback

[Hermeting] In de tweede upload zien we dat de omvang van het systeem fors is gestegen, maar dat daarbij de score voor onderhoudbaarheid vrijwel gelijk is gebleven. Er is een lichte stijging te zien in de scores voor Unit Size en Unit Complexity, met name aan de kant van de 'swift_p2ptv' lijken de units iets korter gemaakt te zijn. Door een daling van de score voor Module Coupling, welke het percentage van code wat relatief vaak wordt aangeroepen meet, blijft de score voor onderhoudbaarheid gelijk. Als laatste zien we in deze aanlevering ook test-code voor de server-kant.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Mochten er nog aanpassingen worden gedaan dan is het aan te raden kritisch naar 'Settings.cpp' te kijken en de functionaliteit die specifiek is voor de settings los te trekken van algemenere helper-functies.

Appendix C

Scripts

These scripts are already available in the TV when it is rooted. These files are provided by the SamyGO community and are loaded at startup. We edited these files and added the code snippets below to start certain applications and set up certain rules.

C.0.1 01_01_catch_crap.init

```
# Bash shell for the TV.
$SYSROOT/bin/busybox nc -l -l -p 1024 -e /mtd_down/widgets/user/SamyGO/SamyGO/bin/bash -i &

cp -fa /dtv/usb/sd*/smb_userdata /mtd_rwarea/smb_userdata

# Start sshd to enable sftp mounting of the whole system.
cp /mtd_down/widgets/user/SamyGO/SamyGO/modules/ssh_host_* /dtv/
chmod 600 /dtv/ssh_host_dsa_key
chmod 600 /dtv/ssh_host_rsa_key
sleep 1

/mtd_down/widgets/user/SamyGO/SamyGO/bin/sshd -f /mtd_down/widgets/user/SamyGO/SamyGO/sshd_config
```

C.0.2 02_04_vusb.init

```
sleep 3

# Run things for the libswift based TV widget.
# First inject some modules so iptables work.
insmod /mtd_down/widgets/user/SamyGO/SamyGO/modules/iptable_filter.ko
sleep 3
insmod /mtd_down/widgets/user/SamyGO/SamyGO/modules/xt_state.ko
sleep 3
insmod /mtd_down/widgets/user/SamyGO/SamyGO/modules/xt_conntrack.ko
sleep 3
# Drop all RST packets on port 1337.
iptables -A INPUT -p tcp --dport 1337 --tcp-flags RST RST -m state --state RELATED,ESTABLISHED -j DROP
sleep 3
iptables -A FORWARD -p tcp --dport 1337 --tcp-flags RST RST -m state --state RELATED,ESTABLISHED -j DROP
sleep 3

# Set the Python environment variables
export PYTHONHOME=/mtd_down/widgets/user/SamyGO/SamyGO/opt/privateer/
export PYTHONPATH=$PYTHONPATH:/mtd_down/widgets/user/SamyGO/SamyGO/tribler
export PYTHONOPTIMIZE=yes

# Start our HTTP webserver. Also the main app.
/mtd_rwcommon/widgets/user/SamyGO/SamyGO/tribler/ws &
```

Appendix D

Sintel video mediainfo

```
General
Complete name                              : sintel_360p.mp4
Format                                     : MPEG-4
Format profile                             : Base Media / Version 2
Codec ID                                   : mp42
File size                                  : 62.1 MiB
Duration                                   : 14mn 48s
Overall bit rate mode                      : Variable
Overall bit rate                           : 587 Kbps
Encoded date                               : UTC 2010-09-29 13:12:54
Tagged date                                : UTC 2010-09-29 13:12:54
gsst                                       : 0
gstd                                       : 888510
gssd                                       : B4A7D0FC4HH1340707962860946
gshh                                       : o-o.preferred.ams03s11.v14.lscache2.c.youtube.com

Video
ID                                         : 2
Format                                     : AVC
Format/Info                                : Advanced Video Codec
Format profile                             : Baseline@L2.0
Format settings, CABAC                     : No
Format settings, ReFrames                  : 1 frame
Format settings, GOP                       : M=1, N=30
Codec ID                                   : avc1
Codec ID/Info                              : Advanced Video Coding
Duration                                   : 14mn 48s
Bit rate                                   : 480 Kbps
Maximum bit rate                           : 1 915 Kbps
Width                                      : 480 pixels
Height                                     : 204 pixels
Display aspect ratio                       : 2.35:1
Frame rate mode                            : Constant
Frame rate                                 : 24.000 fps
Color space                                : YUV
Chroma subsampling                         : 4:2:0
Bit depth                                  : 8 bits
Scan type                                  : Progressive
Bits/(Pixel*Frame)                         : 0.204
Stream size                                : 50.9 MiB (82%)
Title                                      : (C) 2007 Google Inc. v08.13.2007.
Encoded date                               : UTC 2010-09-29 13:12:54
Tagged date                                : UTC 2010-09-29 13:12:55

Audio
ID                                         : 1
Format                                     : AAC
Format/Info                                : Advanced Audio Codec
Format profile                             : LC
Codec ID                                   : 40
Duration                                   : 14mn 47s
Bit rate mode                              : Variable
Bit rate                                   : 104 Kbps
Maximum bit rate                           : 187 Kbps
Channel(s)                                 : 2 channels
Channel positions                          : Front: L R
Sampling rate                              : 44.1 KHz
Compression mode                           : Lossy
Stream size                                : 11.0 MiB (18%)
Title                                      : (C) 2007 Google Inc. v08.13.2007.
Encoded date                               : UTC 2010-09-29 13:12:54
Tagged date                                : UTC 2010-09-29 13:12:55
```

Appendix E

Screenshots

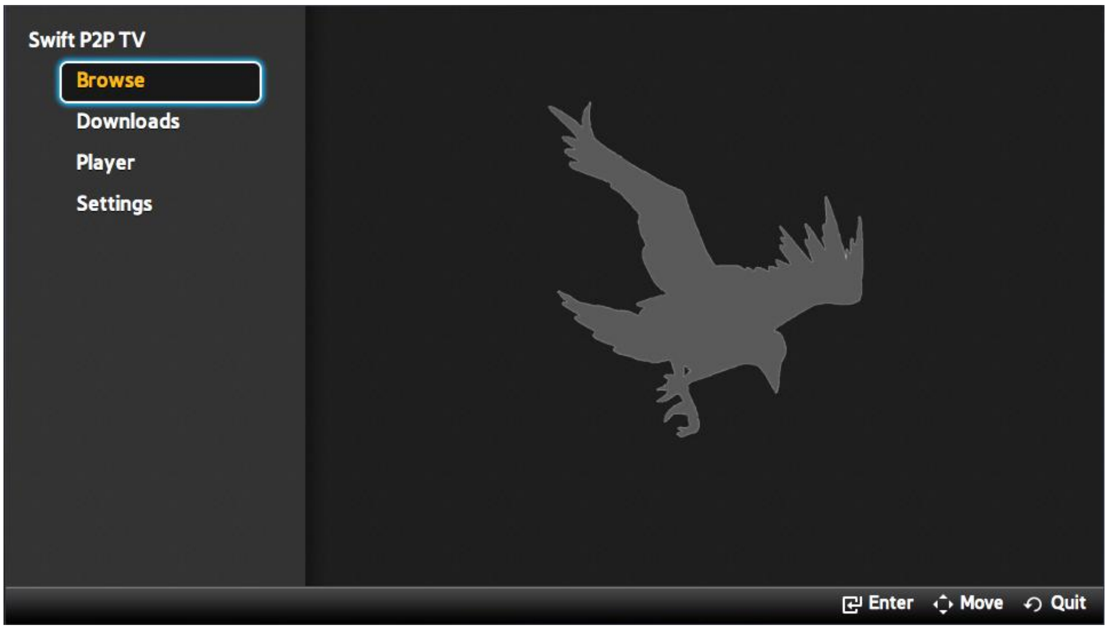
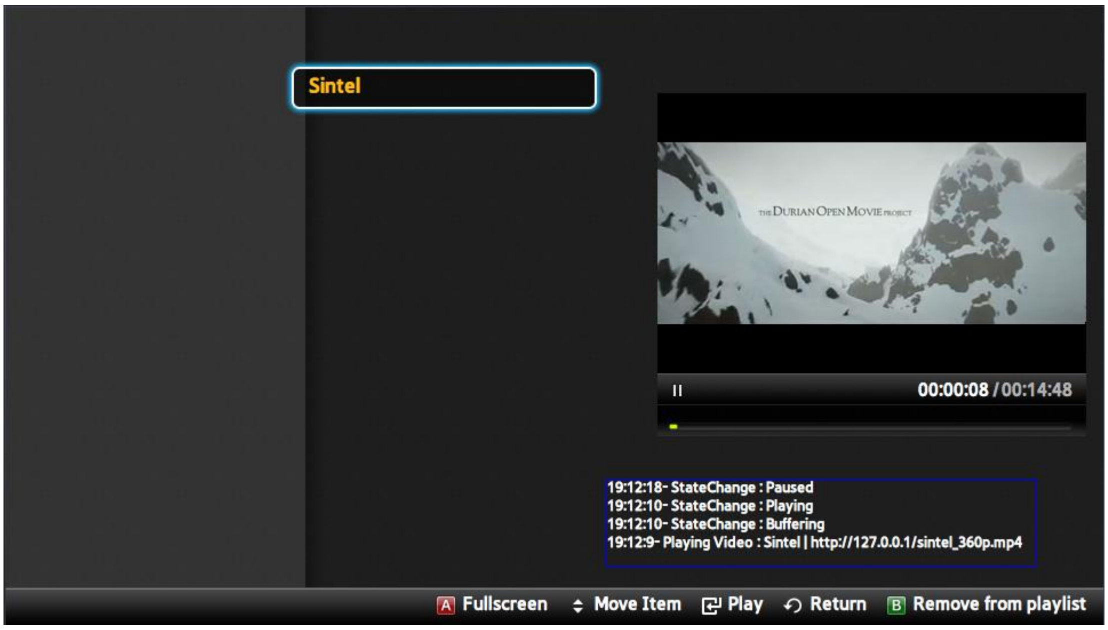
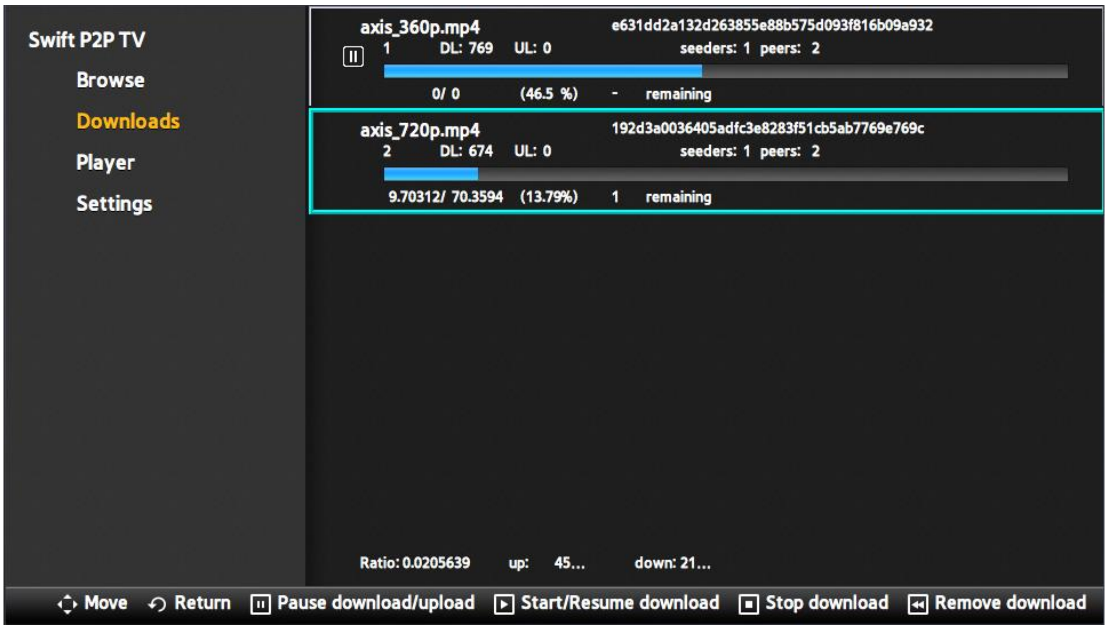


Figure E.1: Screenshot of the Main scene.



```
19:12:18- StateChange : Paused
19:12:10- StateChange : Playing
19:12:10- StateChange : Buffering
19:12:9- Playing Video : Sintel | http://127.0.0.1/sintel_360p.mp4
```

Figure E.2: Screenshot of the Media Player scene.

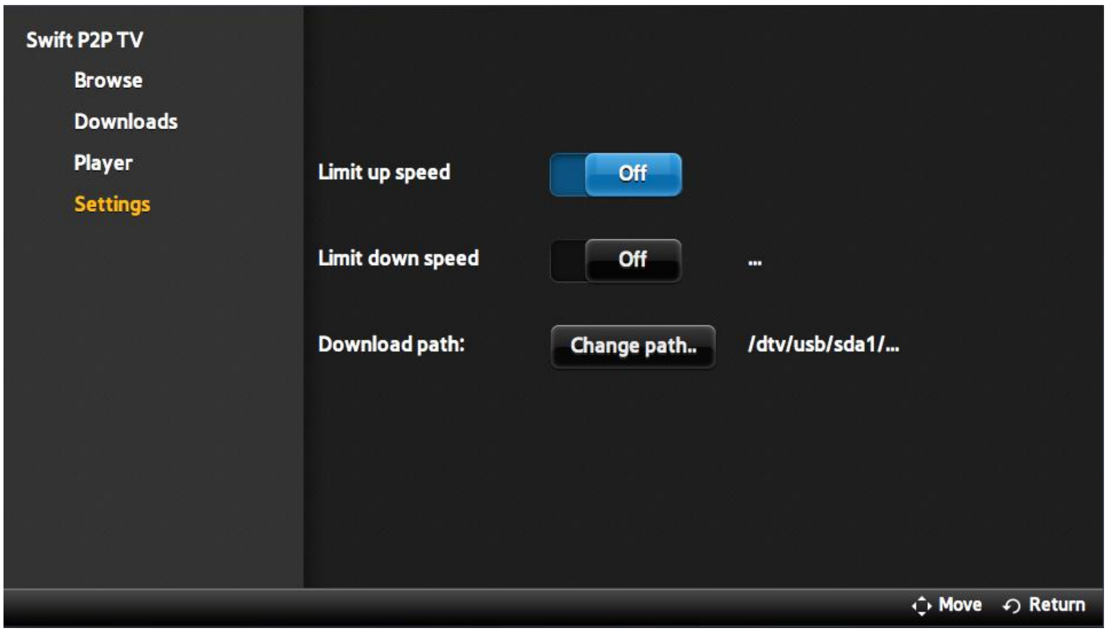


```
axis_360p.mp4                      e631dd2a132d263855e88b575d093f816b09a932
  1        DL: 769   UL: 0                   seeders: 1  peers:  2
        0/ 0       (46.5 %)     -    remaining

axis_720p.mp4                      192d3a0036405adfc3e8283f51cb5ab7769e769c
  2        DL: 674   UL: 0                   seeders: 1  peers:  2
  9.70312/ 70.3594 (13.79%)     1    remaining

Ratio: 0.0205639      up:    45...       down: 21...
```

Figure E.3: Screenshot of the Downloads scene.



Limit up speed Off

Limit down speed Off ...

Download path: /dtv/usb/sda1/...

Figure E.4: Screenshot of the Settings scene.

Appendix F

Terms of reference

P2P on Samsung TV

Student names

Anass Drif	4030532
Wilson Ko	4005686
Jethro Tan	4032918
Samuel Austin	4005996

Projectmanager and client

Dr. Johan Pouwelse and TU Delft P2P team.

General Goal

Our goal is to develop a P2P application for a Samsung TV, which runs Linux. In order to do so, it is needed to port a P2P engine (**libswift** engine) to the TV. First, it is necessary to root the Samsung TV to have access rights, using SamyGo. We will use BusyBox for this purpose, which enables us to add simple UNIX commands to the system.

Context

This paragraph describes and explains the technologies and projects related to this project.

Description P2P

Peer-to-peer computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. (en.wikipedia.org/wiki/Peer-to-peer).

Libswift project outline

Swift is a multiparty transport protocol; making use of swarming, it is basically an expansion on the TCP/IP protocol. Its mission is to disseminate content among a swarm of peers, it can be seen as bittorrent at the transport layer. Libswift is a cross-platform, C++ library which provides swift. With this, developing swift based applications in C++ becomes easier since the protocol is already available in libswift. The protocol is developed at Delft University of Technology, by dr. Johan Pouwelse, dr. Victor Grishchenko and dr. Arno Bakker and is deliverd as part of the P2P-next (http://p2p-next.org/). (http://libswift.org/)

Linux busybox + ARM cross-compile

BusyBox is a small executable containing many common UNIX utilities. It provides replacement for most of the utilities you usually find in GNU fileutils, shellutils etc. BusyBox is perfect for embedded systems such as TVs, because it provides a fairly complete "UNIX" environment. BusyBox has been written with size-optimization and limited resources in mind. It is also extremely modular so you can easily include or exclude commands (or features) at compile time. (www.busybox.net). For our project, we need to make a Samsung TV compatible with BusyBox, so we need to find a cross-compiler for the ARM processor to compile BusyBox.

http://wiki.samygo.tv/index.php5/Setting_up_a_cross-compilation_toolchain

SamyGo project
SamyGO is a project for legal reverse engineering and research on Samsung Television firmwares which is Open Sourced, partially. (www.samygo.tv).) With SamyGo, we can obtain root access and privileges needed to customize the Samsung TV to our needs. This is needed to install BusyBox for example.

Libswift dependencies
It is possible that libswift needs one or more applications in order to work correctly. It goes without saying that it will be our task to find out what the dependencies are and to install them to make libswift work.

Target roadmap

0) Download a cross-compile build environment for ARMv7
1) Get multiple (rootable) Samsung TVs.
2) Root these TVs with SamyGo.
3) Access TV via network or input device.
4) Port libswift to TV and make it work correctly.
5) Build application around libswift engine. (in quarter 4)

Steps 0 to 3 are not really part of the Bachelor project, but will function as preparation for step 5, which is the main goal of the Bachelor project. Therefore, steps 0 to 3 (additionally 4) will be executed in quarter 3 part time, so that the official assignment can start in quarter 4.

Student background knowledge and draft generic work division

The student team consists of four computer science students, two Software Technology students and two Media Knowledge Technology students. All students have experience with c++ and linux. The ST students will mainly concentrate on the low-level parts of the project, such as the firmware and the TV itself (hardware). On the other hand, the MKT students will focus on the interaction and features of the app to build. This does not mean that the students will only concentrate on their part, our goal is for everyone to work at all components of the project.

Appendix G

Orientation Report

IN3405 Bachelor Project

Orientation Report

YouTube
Samsung SMART TV
Smart experience and Connectivity
Movies, shows, apps and more
The ultimate home-base of entertainment
This Smart TV does it all

Authors:
S.E. Austin (4005996)
A. Drif (4030532)
W.K.H. Ko (4005686)
J.E.T. Tan (4032918)

Supervisor:
Dr.Ir. J.A. Pouwelse

Contents

Chapter 1

Introduction

In the third year of the Computer Science course in Delft, students have to do their Bachelor project, where they have to work under supervision of a professor.

This report is an orientation report for our Bachelor Project 2011-2012. The goal of this project is to develop a 4th generation peer-to-peer application to be used on a Samsung Smart TV. In this report we will set out the possible solutions that were explored for this goal, the final solution chosen and progress made to begin development of the application itself. Additionally, some technical background information and a planning for the rest of the project are included.

The client, but also supervisor of this project is Dr.Ir. Johan Pouwelse from the Parallel and Distributed Systems group on the faculty of EEMCS of the Delft University of Technology. He is the scientific director of several peer-to-peer research initiatives and also the founder and leader of the Tribler peer-to-peer research team. Tribler is an open source peer-to-peer client that is completely decentralised. The application to be implemented is required to have the same download-engine, libswift, as Tribler along with an internal media player to handle streaming content and possibly media playback found on an external USB device.

Libswift is a lightweight 4th generation peer-to-peer based transport protocol for content dissemination that can run on top of other protocols, such as UDP, TCP, HTTP, or as RTP profile. It can be used for both live streaming and conventional downloading purposes. 4th generation peer-to-peer software means that it is fully self-organised, removing the need for any server[1].

Because libswift is a very generic protocol, it should be easy to merge with the existing technology of the TV. Since the Smart TV has limited memory and CPU resources, it is necessary that the application to be build is as lightweight as possible. The streaming capabilities also enables the user to maximise utility of the TV, while still able to download data and store it. A major drawback is however that downloading and streaming content is limited to the memory of the TV.

[1]`http://www.tribler.org/trac/wiki/4thGenerationP2P`

Chapter 2

Usage of Libswift

Libswift comes as a command line application, the version we use can be found at `https://github.com/jettan/swiftarm/tree/master/swift`. With this application, it is possible to upload and download files. Streaming audio and video also belongs to the possibilities. These streams can be bound to a port which can be used on a media player to play back the content. Control of libswift can also be done via TCP. Unlike conventional peer-to-peer systems, libswift does not require a centralised tracker to operate. This allows libswift to work completely decentralised, which makes it very robust. A short video where the command line usage of libswift is demonstrated can be found on YouTube[1]. For more information on how libswift works, we refer the reader to `https://datatracker.ietf.org/doc/draft-ietf-ppsp-peer-protocol/`.

For our own application, we will have to use the same functionality as the command line application of libswift, but then stripped down to fit our needs. In short, we will use the swift.cpp file as an example to develop the basics for our application.

[1] `http://www.youtube.com/watch?v=MUCUMA0LVgc`

Chapter 3

Solutions

To improve the feasibility of this project, multiple solutions to develop the application were explored. In this section, we will list 3 solutions for the Samsung Smart TV.

3.1 Develop a native application

The first solution is to implement the peer-to-peer application so that it can be run natively on the TV itself. The TVs to be used are the models UE40D7000[1] and UE46D8000[2]. In order to run the application natively on the TV, root access is required, as the TV does not allow external libraries, like libswift, to be used in the Samsung SDK. In fact, one can only develop applications with HTML, DOM, CSS or JavaScript with Flash support and the built-in media player[3].

To gain root access, one must follow the steps on `http://wiki.samygo.tv/index.php5/Rooting_D_series_arm_cpu_models`.

These are the steps to gain root access:

Create the developer account.

1. Enter recovery menu on the TV (press info, menu, mute, power while TV is turned off)
2. Set the RS232 on debug and the watchdog off.
3. Save settings and reboot the TV normally.
4. Enter SmartHub.
5. Go to Settings.

[1]`http://www.samsung.com/nl/consumer/tv-audio-video/televisions/led-tv/UE40D7000LSXXN-spec`
[2]`http://www.samsung.com/nl/consumer/tv-audio-video/televisions/led-tv/UE46D8000YSXXH-spec`
[3]`http://www.samsungdforum.com/Devtools/Spec`

6. Create a developer account (name = develop, password any)
7. Exit SmartHub and reboot the TV.

Now install the hack:

1. Enter SmartHub.
2. Login on your developer account.
3. Go to Settings, Development, Setting Server IP
4. Enter 46.4.199.222 as IP.
5. Press User Application Synchronization (This could take a long time).
6. Exit Developer menu and SmartHub.
7. Reboot TV.
8. Enter SmartHub.
9. Execute the SamyGo widget.

With this the TV will be rooted. To cross-compile code for the Samsung TV, a cross-compilation toolchain is needed, which can be found at `https://opensource.samsung.com`. With this, we can cross-compile libswift for the TV. Further development will then involve the use of the Samsung SDK with another daemon that handles the libswift calls.

3.2 Developing an application for the Raspberry Pi

The second solution is to implement an application for an embedded system sporting an arm processor such as the Raspberry Pi[4] and using the Raspberry Pi as a set-top box on the Samsung Smart TV. Since the Raspberry Pi already runs a Linux distribution such as Fedora, development can be done on the device itself. Lots of libraries are already present and we can even make use of strong media players such as mplayer. In short, the application to be developed would then be a Tribler clone, but only for arm Linux.

3.3 Develop an application for the Galaxy Nexus

The main idea of this solution is to develop an application that runs on the Samsung Galaxy Nexus with Android Ice Cream Sandwich. Display will be forwarded to the TV screen by using an MHL cable[5]. Since the Tribler team already ported libswift for the Android successfully, we can proceed with the development of the application itself when choosing this solution.

[4]`http://www.raspberrypi.org`

[5]`http://www.handtec.co.uk/product.php/6075/samsung-galaxy-nexus-official-mhl-microusb-to-hdmi-adapter`

Chapter 4

Study of Feasibility

In this section, we will evaluate all solutions to study the feasibility of the whole project. Afterwards, we will select one solution to carry out and use the other solutions as backup plans.

The preparations for the first solution should not be hard to carry out, especially since all the steps needed to be taken are well documented on the Internet. Also, we can get help from the SamyGo developers when needed[1].

The second solution is in essence the same as the first solution, but on another platform than a TV (such as the Raspberry Pi).

As for the third solution, our client already possesses knowledge regarding libswift applications for Android. Furthermore, developing applications for Android is not hard because its extensive API and the many possibilities it offers[2]. So the third solution is also feasible.

The diversity of possibilities makes the project very feasible, because if the first solution really does not work out we can use the second solution as backup plan. Because the software written for the TV will be the same as the software written for a board like Raspberry PI, it is easy to switch between the two solutions. The only thing that needs to be done is to cross-compile the software for the correct platform. The third solution also makes use of the same Samsung TV model as the previous solutions, so switching to the third solution is easy.

Since the first solution is already quite feasible itself, it is best to try that and to use the other solutions as backup plans.

[1] `http://forum.samygo.tv`

[2] `http://developer.android.com/reference/packages.html`

Chapter 5

Target Roadmap

In order to keep the development of the application structured we have divided the functionality we would like to achieve into separate phases. Each phase implements a relatively small and overseeable chunk of functionality which builds upon and requires the previous phase to be completed. Each phase also guarantees a complete and fully functional piece of software. For example, this means that if by the end of the project we are unable to implement phase 4 we will still have a fully functional application, although with less functionality, as we have finished phase 3.

The phases as described now are aimed primarily at solution 1 although they can be easily modified to apply to the other solutions.

The functionality described in the phases could be categorised as must have, should have, would have and could have functionality. What exactly each phase will entail is explained below.

5.1 Phase 1

This phase concentrates on getting the very basics up and running. Our application will be built around the libswift library and therefore libswift must be compiled and executable on the TV. As libswift is for file transfer over the Internet we will also need to make the TV internet connectible and be able to download files. To access the TV easier we want to be able to access the TV remotely by Telnet, SSH or something similar.

To summarise:

- The TV is Internet connectible;
- The TV is remotely accessible;
- Libswift runs on the TV;
- You can download files from the Internet via the command-line.

At the moment, this phase is already finished as we already succeeded in gaining root access and have libswift up and running on the TV. See here6 for more about the current progress. Although we already have root access on the TV, we would like to find a method to gain root access on the newest firmware currently available so that our application will be compatible with the latest software.

5.2 Phase 2

The next step is to build our own application using functionality from the libswift library. This step will also be run from the command-line. We want to be able to download files using the libswift library and be able to pass them on to a media decoder and play them on the screen. Another feature that libswift offers is streaming of video rather than downloading a file completely and playing it once the download is complete. We would like to implement that in this phase as well.

- Downloaded files are passed on to a media decoder;
- Video plays on the screen;
- Download and play functionality;
- Full streaming functionality.

5.3 Phase 3

This phase is aimed at making a more user friendly application rather than the command line interface that was being used up until this point. The advantage is that all the download functionality has been implemented in the previous phases so all that needs to be done is build a GUI around the the previous phases. We would like the application to be controlled using the TV remote control and have some hard coded test swarms from which you can download files for demonstration purposes.

- A GUI controlled by the TV remote control;
- A file browser;
- A list of test swarms (hard coded) from which you can download and play.

5.4 Phase 4

In phase 4 we have a number of options. Due to the fact that it is probably unrealistic to implement them all within the given time we have decided to split them up into three groups. Which one we will choose will be decided at a later time during the project based on the amount of time we have left and our personal preference.

5.4.1 Phase 4A

The first option is to get Python, SQLite and M2Crypto working on the TV. This would enable us to run ‘Dispersie’, made by the Tribler team and let us implement searching, browsing and sharing functionality.

- Get Python, SQLite and M2Crypto to run;
- Use this to implement searching, browsing, streaming and sharing functionality.

5.4.2 Phase 4B

Another option is to expand the application by adding a more advanced GUI with search functionality. Also TV to TV download and upload would be interesting rather than downloading from a central server. We could also add more demo applications and functions.

- Expanding the GUI;
- Add Search functionality;
- TV to TV uploading and downloading;
- More demo apps;

5.4.3 Phase 4C

The last option is to connect a web cam to the TV and support streaming to multiple other TV's from the TV with the web cam.

- Live streaming from a web cam to multiple TV's using the libswift library.

5.4.4 Planning

Week	Goals
23th – 29th April	Finish Orientation report. Attempt to gain root access to newest firmware version of TV. Create design for the back-end of our application. Create design for the front-end of our application and find out how to integrate it with the back-end.
30th April – 6th May	Finish design of application. Create basic application with calls to libswift.
7th – 13th May	Start implementation of our application and test suite.
14th – 20th May	Continue implementation.
21st – 27th May	Continue implementation.
28th May – 3rd June	Start working towards finished product.
4th –10th June	Start writing report. Prepare code for SIG.
11th – 17th June	Write report. Improve code based on SIG feedback.
18th – 24th June	Prepare code for SIG.

Table 5.1: General planning for the whole project

Chapter 6

Current Progress

In this section, we will describe what we have been working on during the third quarter as preparation for the development of the TV application. Our first choice was to implement the application to run on the TV natively. Not only because it's challenging, but also because we believe it is possible for the TV to run it natively. The idea of having a TV without a set-top box or another device running the to be implemented application attracted us to do so. Moreover, since the Tribler team already ported libswift for Android and made a demo application, it wouldn't be as interesting as implementing the application to run native on the TV. Should the TV be unrootable, we would have chosen this solution. Implementing the application to run on the Raspberry Pi was our last backup solution. Not only due to the unknown availibility of the device, but also because developing an application for it wouldn't be much different than developing an application for your own computer.

6.1 Gaining root access to the TV

Initially we believed getting root access of the TV would be a simple step, but it turned out to be harder than we thought. To root the TV we planned on using the method provided by SamyGo (See how to root3.1), but because the firmware version of our TV was the latest, we could not use the SamyGo tool[1].

Since there was no official way to downgrade to a lower firmware version, we decided to try to trick the TV into downgrading. The idea was to create our own firmware which would be an old version of the firmware repackaged with a higher version number. Since Samsung provided access to the source of all the firmware versions on the Internet, all that needed to be done was to change the version number and repackage it. Sadly Samsung incorporates a salted signature into the packaging of the firmware. While the SamyGo team possesses a tool to decrypt the packages succesfully, there was no way to repackage them back, so this plan ended up failing.

[1]The SamyGo tool requires the TV to be at firmware $\leq$ 1019.0, whereas our TV was at 1021.0.

During our own attempts to downgrade the firmware we had contacted the developers at SamyGo to ask whether they had a solution for our firmware version or an approximate date for a new root kit. After some conversation they agreed to give us access to a server they used to spoof the Samsung online firmware updating server. Using the same technique we attempted offline, the idea was to trick the TV into downgrading by making it think it is upgrading. The online version however, succeeded to downgrade the TV to firmware 1015.0. Apparently a signature was not needed or was circumvented in some other way. In return of this favour, we promised we would put some time and effort into a method for gaining root access to the newest firmware version. The end result, however, was a TV with root access with which we could continue to the next step, getting libswift to run. After rooting the TV, it was possible to transfer files from and to the TV using FTP and to log into the TV using a netcat shell.

6.2 Compiling libswift for the TV

Even though we have compiled libswift for ARM machines before, this still proved to be a non-trivial task. Because the root file system on the TV is read-only, adding libraries like libevent, on which libswift depends was out of the question. This caused problems when trying to dynamically link libraries. By setting the 'static' flag on, libraries can be linked statically and solving this problem.

The problem that costed most of our time, however, had to do with the transfer of the compiled libswift to the TV. The libswift library would be compiled on a laptop and then transferred to the TV using FileZilla, an FTP client. After a long time of wondering why the compiled library would not run or even recognized as executable file by GDB[2] on the TV, we discovered that FileZilla automatically decides whether the file to be transferred is an ASCII file or a binary file. For some reason, FileZilla treated our compiled library as an ASCII file. Once we switched to using FTP in the command line and setting the transfer mode to binary this problem was resolved and got libswift running.

6.3 Running libswift on the TV

Once libswift was compiled and running on the TV we were finally able to actually test whether the TV could handle it. For this we tried a number of video and music files of different quality streamed from a laptop to the TV and from the TV to the laptop. Videos up to 720p with 24 frames per second streamed and played seamlessly. The TV's media player limitations, however, still apply.[3] When feeded 1080p video files, the TV began to have some troubles. We made a demonstration video of these results which can be seen on YouTube[4], this is the same video mentioned earlier.

[2]GNU debugger

[3]`http://www.samsung.com/uk/consumer/learningresources/media2.0/usb_faq.html`

[4]`http://www.youtube.com/watch?v=MUCUMA0LVgc`

Appendix H

Code API

Swiftarm

sig1

Generated by Doxygen 1.7.6.1

Thu Jul 5 2012 19:53:36

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 DownloadManager Namespace Reference

Classes

- struct Amount

Functions

- void calculateRatio ()
- void calculateDownloadAmount ()
- void calculateUploadAmount ()
- void setActiveDownload (Download ∗download)
- Download getActiveDownload ()
- std::vector< Download > getDownloads ()
- void init ()
- void startStream (std::string tracker, std::string hash)
- void stopStream ()
- void pauseAllDownloads ()
- void resumeAllDownloads ()
- void ∗ startStreamThread (void ∗arg)
- void updateDownloadStatistics ()
- std::string buildXML ()
- void ∗ dispatch (void ∗arg)
- int resumeDownload (std::string download_hash)
- int startDownload (const std::string download_hash)
- int getIndexFromHash (const std::string download_hash)
- struct Amount getDownloadAmount ()
- struct Amount getUploadAmount ()
- double getRatio ()
- double getMaxUpSpeed ()

- double getMaxDownSpeed ()
- void setMaxUpSpeed (double speed)
- void setMaxDownSpeed (double speed)
- void limitUpSpeeds (double speed)
- void limitDownSpeeds (double speed)
- void upload (std::string filename)
- void startUploads ()
- void downloadFirstInList ()
- void add (Download ∗download)
- void pauseDownload (const std::string download_hash)
- void switchDownload (std::string hash)
- void removeFromList (const std::string download_hash)
- void removeFromDisk (const std::string download_hash)
- void clearList ()

Variables

- static std::vector< Download > downloads

  Vector containing all downloads.
- static struct event evcompl

  Event needed to start download loop.
- static Download ∗ active_download

  The currently active download.
- static pthread_t streaming_thread

  Thread needed to play streams.
- static pthread_t thread

  Thread needed to download and upload.
- static pthread_mutex_t mutex

  Mutex to make downloads vector thread safe.
- static pthread_mutex_t active_download_mutex

  Mutex to make active_download thread safe.
- static ticpp::Document ∗ doc

  XML Document for the download statistics.
- static double ratio

  Upload amount divided by download amount.
- static double downloaded

  Total amount of bytes downloaded this session.
- static double uploaded

  Total amount of bytes uploaded this session.
- static int d_pid = -1

  Download thread pid.
- static double max_upspeed

  Max upload speed.
- static double max_downspeed

  Max download speed.

4.1.1 Detailed Description

Manages all up- and downloads. Also responsible for playing streams.

4.1.2 Function Documentation

4.1.2.1 void **DownloadManager::add (Download** ∗ *download*)

Adds a download to the list.

Parameters

download,:	The download to be added.

Definition at line 564 of file DownloadManager.cpp.

4.1.2.2 std::string **DownloadManager::buildXML** ( )

Builds the XML response for the HTTP server, providing download statistics.

Definition at line 323 of file DownloadManager.cpp.

4.1.2.3 void **DownloadManager::calculateDownloadAmount** ( )

Calculates the download amount.

Definition at line 49 of file DownloadManager.cpp.

4.1.2.4 void **DownloadManager::calculateRatio** ( )

Calculates the upload/download ratio.

Definition at line 42 of file DownloadManager.cpp.

4.1.2.5 void **DownloadManager::calculateUploadAmount** ( )

Calculates the upload amount.

Definition at line 56 of file DownloadManager.cpp.

4.1.2.6 void **DownloadManager::clearList** ( )

Removes all downloads from the list

Definition at line 893 of file DownloadManager.cpp.

4.1.2.7 void $*$ DownloadManager::dispatch (void $*$ *arg*)

Dispatches swift::Channel::evbase to get into the main loop to download files.

Parameters

arg,:	Unused argument of p'thread_create.

Definition at line 421 of file DownloadManager.cpp.

4.1.2.8 void DownloadManager::downloadFirstInList ( )

Downloads first element in the download list.

Definition at line 617 of file DownloadManager.cpp.

4.1.2.9 Download DownloadManager::getActiveDownload ( )

Only for testing purposes. Returns the active_download variable. Method is not thread safe, so refrain from using it. Instead, acces active_download directly and lock it with the _active_download_mutex.

Definition at line 66 of file DownloadManager.cpp.

4.1.2.10 struct DownloadManager::Amount DownloadManager::getDownload-Amount ( ) [read]

Returns the downloaded amount in kb, MB or GB.

Definition at line 706 of file DownloadManager.cpp.

4.1.2.11 std::vector$<$ Download $>$ DownloadManager::getDownloads ( )

Returns a copy of the vector downloads.

Definition at line 21 of file DownloadManager.cpp.

4.1.2.12 int DownloadManager::getIndexFromHash (const std::string *download_hash*)

Finds the index of a download in the list given a root hash.

Parameters

download_-hash,:	The root hash of a download to be searched.

Definition at line 693 of file DownloadManager.cpp.

4.1.2.13 double DownloadManager::getMaxDownSpeed ( )

Returns the maximum download speed in kb/s.

Definition at line 751 of file DownloadManager.cpp.

4.1.2.14 double DownloadManager::getMaxUpSpeed ( )

Returns the maximum download speed in kb/s.

Definition at line 758 of file DownloadManager.cpp.

4.1.2.15 double DownloadManager::getRatio ( )

Returns the upload/download ratio.

Definition at line 744 of file DownloadManager.cpp.

4.1.2.16 struct DownloadManager::Amount DownloadManager::getUploadAmount ( ) [read]

Returns the uploaded amount in kb, MB or GB.

Definition at line 725 of file DownloadManager.cpp.

4.1.2.17 void DownloadManager::init ( )

Initialises the download manager.

Definition at line 6 of file DownloadManager.cpp.

4.1.2.18 void DownloadManager::limitDownSpeeds (double *speed*)

Sets the maximum download speed in kb/s.

Parameters

speed,:	speed in kb/s.

Definition at line 802 of file DownloadManager.cpp.

4.1.2.19 void DownloadManager::limitUpSpeeds (double *speed*)

Sets the maximum upload speed in kb/s.

Parameters

speed,:	speed in kb/s.

Definition at line 820 of file DownloadManager.cpp.

4.1.2.20 void **DownloadManager::pauseAllDownloads** ( )

Pause all the downloads.

Definition at line 932 of file DownloadManager.cpp.

4.1.2.21 void **DownloadManager::pauseDownload** (const std::string *download_hash*)

Pauses a download with the given root hash.

Parameters

download_-hash,:	The root hash of the download.

Definition at line 437 of file DownloadManager.cpp.

4.1.2.22 void **DownloadManager::removeFromDisk** (const std::string *download_hash*)

Removes a download from the hard disk given a root hash.

Parameters

download_-hash,:	The root hash of the download to be removed.

Definition at line 871 of file DownloadManager.cpp.

4.1.2.23 void **DownloadManager::removeFromList** (const std::string *download_hash*)

Removes a download from the list given a root hash.

Parameters

download_-hash,:	The root hash of the download to be removed.

Definition at line 838 of file DownloadManager.cpp.

4.1.2.24 void DownloadManager::resumeAllDownloads ( )

Resume all the downloads.

Definition at line 945 of file DownloadManager.cpp.

4.1.2.25 int DownloadManager::resumeDownload (std::string *download_hash*)

Resumes a paused download given a root hash.

Parameters

download_-hash,:	The root hash of the download.

Definition at line 455 of file DownloadManager.cpp.

4.1.2.26 void DownloadManager::setActiveDownload (Download ∗ *download*)

Sets the active download.

Parameters

download,:	The download to be the active download.

Definition at line 33 of file DownloadManager.cpp.

4.1.2.27 void DownloadManager::setMaxDownSpeed (double *speed*)

Sets the maximum download speed in kb/s.

Parameters

speed,:	speed in byte/s.

Definition at line 766 of file DownloadManager.cpp.

4.1.2.28 void DownloadManager::setMaxUpSpeed (double *speed*)

Sets the maximum upload speed in kb/s.

Parameters

speed,:	speed in byte/s.

Definition at line 784 of file DownloadManager.cpp.

4.1.2.29 int **DownloadManager::startDownload** (const std::string *download_hash*)

Starts a download given a root hash.

Parameters

download_-hash,:	The root hash of the download.

Definition at line 502 of file DownloadManager.cpp.

4.1.2.30 void **DownloadManager::startStream** (std::string *tracker,* std::string *hash*)

Starts the streaming thread.

Parameters

tracker,:	The tracker from which we stream content.

Definition at line 974 of file DownloadManager.cpp.

4.1.2.31 void ∗ **DownloadManager::startStreamThread** (void ∗ *arg*)

Starts streaming.

Parameters

arg,:	Unused argument of pthread_create.

Definition at line 959 of file DownloadManager.cpp.

4.1.2.32 void **DownloadManager::startUploads** ( )

Starts all Uploads located in the public folder.

Definition at line 672 of file DownloadManager.cpp.

4.1.2.33 void **DownloadManager::stopStream** ( )

Stops streaming.

Definition at line 917 of file DownloadManager.cpp.

4.1.2.34 void **DownloadManager::switchDownload** (std::string *hash*)

Switches active download to another download with the given root hash.

Parameters

hash,:	The root hash of the download to be switched to.

Definition at line 533 of file DownloadManager.cpp.

4.1.2.35 void **DownloadManager::updateDownloadStatistics** ( )

Updates the download statistics.

Definition at line 73 of file DownloadManager.cpp.

4.1.2.36 void **DownloadManager::upload** (std::string *filename*)

Uploads a file.

Parameters

filename,:	Name of the file to be uploaded.

Definition at line 640 of file DownloadManager.cpp.

4.1.3 Variable Documentation

4.1.3.1 **Download∗ DownloadManager::active_download** `[static]`

The currently active download.

Definition at line 41 of file DownloadManager.h.

4.1.3.2 pthread_mutex_t **DownloadManager::active_download_mutex** `[static]`

Mutex to make active_download thread safe.

Definition at line 53 of file DownloadManager.h.

4.1.3.3 int **DownloadManager::d_pid** = -1 `[static]`

Download thread pid.

Definition at line 68 of file DownloadManager.h.

4.1.3.4 ticpp::Document∗ **DownloadManager::doc** `[static]`

XML Document for the download statistics.

Definition at line 56 of file DownloadManager.h.

4.1.3.5 double DownloadManager::downloaded [static]

Total amount of bytes downloaded this session.

Definition at line 62 of file DownloadManager.h.

4.1.3.6 std::vector<Download> DownloadManager::downloads [static]

Vector containing all downloads.

Definition at line 35 of file DownloadManager.h.

4.1.3.7 struct event DownloadManager::evcompl [static]

Event needed to start download loop.

Definition at line 38 of file DownloadManager.h.

4.1.3.8 double DownloadManager::max_downspeed [static]

Max download speed.

Definition at line 74 of file DownloadManager.h.

4.1.3.9 double DownloadManager::max_upspeed [static]

Max upload speed.

Definition at line 71 of file DownloadManager.h.

4.1.3.10 pthread_mutex_t DownloadManager::mutex [static]

Mutex to make downloads vector thread safe.

Definition at line 50 of file DownloadManager.h.

4.1.3.11 double DownloadManager::ratio [static]

Upload amount divided by download amount.

Definition at line 59 of file DownloadManager.h.

4.1.3.12 pthread_t DownloadManager::streaming_thread [static]

Thread needed to play streams.

Definition at line 44 of file DownloadManager.h.

4.1.3.13 pthread_t DownloadManager::thread `[static]`

Thread needed to download and upload.

Definition at line 47 of file DownloadManager.h.

4.1.3.14 double DownloadManager::uploaded `[static]`

Total amount of bytes uploaded this session.

Definition at line 65 of file DownloadManager.h.

4.2 HttpServer Namespace Reference

Functions

- static void sendXMLResponse (std::string msg, struct evhttp_request ∗req, struct evbuffer ∗buf)
- static void sendResponse (struct evhttp_request ∗req, struct evbuffer ∗buf, const char ∗message)
- static void handleRequest (struct evhttp_request ∗req, void ∗arg)
- int init ()

Variables

- struct event_base ∗ base

4.2.1 Detailed Description

Http request handler. Acts as controller of the webserver subsystem.

4.2.2 Function Documentation

4.2.2.1 static void HttpServer::handleRequest (struct evhttp_request ∗ *req*, void ∗ *arg*) `[static]`

The HTTP GET request handler.

Parameters

req,:	The request struct from libevent.
arg,:	Ignored argument.

Definition at line 273 of file HttpServer.cpp.

4.2.2.2 int HttpServer::init ( )

Initialises the web server.

Definition at line 419 of file HttpServer.cpp.

4.2.2.3 static void **HttpServer::sendResponse** (struct evhttp_request ∗ *req*, struct evbuffer ∗ *buf*, const char ∗ *message*) `[static]`

Sends the HTTP response.

Parameters

req,:	The received HTTP request.
buf,:	Buffer used to send the reply.
message,:	The message to be sent.

Definition at line 36 of file HttpServer.cpp.

4.2.2.4 static void **HttpServer::sendXMLResponse** (std::string *msg*, struct evhttp_request ∗ *req*, struct evbuffer ∗ *buf*) `[static]`

Sends the HTTP XML response.

Parameters

msg,:	The xml message to be sent.
req,:	The received HTTP request.
buf,:	Buffer used to send the reply.

Definition at line 13 of file HttpServer.cpp.

4.2.3 Variable Documentation

4.2.3.1 struct event_base∗ **HttpServer::base**

Definition at line 4 of file HttpServer.cpp.

4.3 SearchEngine Namespace Reference

Classes

- struct result

  Struct holding the data of a search result.

Functions

- void ∗ startDispersy (void ∗arg)
- void clearSearchResults ()
- std::string buildSearchXML ()
- void search (std::string search_term)
- std::string getResults ()
- struct result getResultWithHash (std::string hash)
- struct result getResultWithName (std::string name)
- void init ()

Variables

- static pthread_t dispersy_thread

  Thread to run dispersy module in.
- static pthread_mutex_t dispersy_mutex

  Mutex to protect dispersy thread.
- static std::vector< struct result > search_results

  Vector of all search results.
- static ticpp::Document ∗ searchdoc

  Document used to build the list of search results.
- PyObject ∗ p_module_name
- PyObject ∗ p_module
- PyObject ∗ p_main
- PyObject ∗ p_function
- PyObject ∗ p_args
- PyObject ∗ p_main_value
- PyObject ∗ p_function_value
- PyObject ∗ p_result_string
- PyGILState_STATE gstate
- pthread_mutex_t gstate_mutex
- PyThreadState ∗ save

4.3.1 Detailed Description

Interface to Dispersy module. Makes use of Dispersy methods to find files online.

4.3.2 Function Documentation

4.3.2.1 std::string SearchEngine::buildSearchXML ( )

Builds an XML string from the vector of results.

Definition at line 17 of file SearchEngine.cpp.

4.3.2.2 void **SearchEngine::clearSearchResults** ( )

Clears the list of search results.

Definition at line 13 of file SearchEngine.cpp.

4.3.2.3 std::string **SearchEngine::getResults** ( )

Returns the current list of results.

Definition at line 57 of file SearchEngine.cpp.

4.3.2.4 struct **SearchEngine::result SearchEngine::getResultWithHash** (std::string *hash*) `[read]`

Returns the result with a cetain hash.

Definition at line 108 of file SearchEngine.cpp.

4.3.2.5 struct **SearchEngine::result SearchEngine::getResultWithName** (std::string *name*) `[read]`

Returns the result with a cetain filename.

Definition at line 125 of file SearchEngine.cpp.

4.3.2.6 void **SearchEngine::init** ( )

Init function to set up python calls and start dispersy.

Definition at line 191 of file SearchEngine.cpp.

4.3.2.7 void **SearchEngine::search** (std::string *search_term*)

Search.

Definition at line 158 of file SearchEngine.cpp.

4.3.2.8 void ∗ **SearchEngine::startDispersy** (void ∗ *arg*)

Call the main() from DispersyInterface.

Definition at line 142 of file SearchEngine.cpp.

4.3.3 Variable Documentation

4.3.3.1 pthread_mutex_t SearchEngine::dispersy_mutex [static]

Mutex to protect dispersy thread.

Definition at line 23 of file SearchEngine.h.

4.3.3.2 pthread_t SearchEngine::dispersy_thread [static]

Thread to run dispersy module in.

Definition at line 20 of file SearchEngine.h.

4.3.3.3 PyGILState_STATE SearchEngine::gstate

Definition at line 8 of file SearchEngine.cpp.

4.3.3.4 pthread_mutex_t SearchEngine::gstate_mutex

Definition at line 9 of file SearchEngine.cpp.

4.3.3.5 PyObject* SearchEngine::p_args

Definition at line 6 of file SearchEngine.cpp.

4.3.3.6 PyObject * SearchEngine::p_function

Definition at line 5 of file SearchEngine.cpp.

4.3.3.7 PyObject * SearchEngine::p_function_value

Definition at line 6 of file SearchEngine.cpp.

4.3.3.8 PyObject * SearchEngine::p_main

Definition at line 5 of file SearchEngine.cpp.

4.3.3.9 PyObject * SearchEngine::p_main_value

Definition at line 6 of file SearchEngine.cpp.

4.3.3.10 PyObject * SearchEngine::p_module

Definition at line 5 of file SearchEngine.cpp.

4.3.3.11 PyObject∗ SearchEngine::p_module_name

Definition at line 5 of file SearchEngine.cpp.

4.3.3.12 PyObject∗ SearchEngine::p_result_string

Definition at line 7 of file SearchEngine.cpp.

4.3.3.13 PyThreadState∗ SearchEngine::save

Definition at line 10 of file SearchEngine.cpp.

4.3.3.14 std::vector<struct result> SearchEngine::search_results `[static]`

Vector of all search results.

Definition at line 26 of file SearchEngine.h.

4.3.3.15 ticpp::Document∗ SearchEngine::searchdoc `[static]`

Document used to build the list of search results.

Definition at line 29 of file SearchEngine.h.

4.4 Settings Namespace Reference

Functions

- void init (std::string download_dir)
- void loadSettings (std::string filename)
- void saveSettings (std::vector< std::string > settings)
- void setDownloadDirectory (std::string dir)
- void setIP (std::string ip)
- void setMaxUpSpeed (double speed)
- void setMaxDownSpeed (double speed)
- std::vector< std::string > & split (const std::string &str, char delim, std::vector< std::string > &elems)
- std::vector< std::string > split (const std::string &str, char delim)
- std::string getIP ()
- std::string getDownloadDirectory ()
- bool directoryExists (std::string path)
- double getMaxUpSpeed ()
- double getMaxDownSpeed ()
- std::string replaceSubstring (std::string str, const std::string from, const std::string to)

Variables

- static std::string ip_address

  Keeps track of the current ip address.
- static std::string download_directory

  Keeps track of the current download directory.
- static pthread_mutex_t mutex

  Mutex to make download_directory variable thread safe.
- static pthread_mutex_t max_mutex

  Mutex to make max speed variables thread safe.
- static double max_download_speed

  Maximum download speed.
- static double max_upload_speed

  Maximum upload speed.
- static std::string settings_file

  Settings file.

4.4.1 Detailed Description

General Settings manager.

4.4.2 Function Documentation

4.4.2.1 bool **Settings::directoryExists** (std::string *path*)

Definition at line 95 of file Settings.cpp.

4.4.2.2 std::string **Settings::getDownloadDirectory** ( )

Returns the current download directory.

Parameters

ip,:	The ip to be set.

Definition at line 196 of file Settings.cpp.

4.4.2.3 std::string **Settings::getIP** ( )

Returns the current ip.

Definition at line 150 of file Settings.cpp.

4.4.2.4 double **Settings::getMaxDownSpeed** ( )

Returns the maximum download speed.

Definition at line 157 of file Settings.cpp.

4.4.2.5 double **Settings::getMaxUpSpeed** ( )

Returns the maximum upload speed.

Definition at line 167 of file Settings.cpp.

4.4.2.6 void **Settings::init** (std::string *download_dir*)

Initialises Utils.

Definition at line 206 of file Settings.cpp.

4.4.2.7 void **Settings::loadSettings** (std::string *filename*)

Loads the settings file.

Parameters

filename,:	Settings file to be loaded.

Definition at line 29 of file Settings.cpp.

4.4.2.8 std::string **Settings::replaceSubstring** (std::string *str,* const std::string *from,* const std::string *to*)

Replace a substring in a string.

Definition at line 248 of file Settings.cpp.

4.4.2.9 void **Settings::saveSettings** (std::vector< std::string > *settings*)

Saves the settings.

Definition at line 77 of file Settings.cpp.

4.4.2.10 void **Settings::setDownloadDirectory** (std::string *dir*)

Sets the directory where swift will download files to.

Parameters

dir,:	The directory where swift will download files to.

Definition at line 117 of file Settings.cpp.

4.4.2.11 void Settings::setIP (std::string *ip*)

Sets the current ip.

Parameters

ip,:	The ip to be set.

Definition at line 136 of file Settings.cpp.

4.4.2.12 void Settings::setMaxDownSpeed (double *speed*)

Sets the maximum download speed.

Definition at line 177 of file Settings.cpp.

4.4.2.13 void Settings::setMaxUpSpeed (double *speed*)

Sets the maximum upload speed.

Definition at line 186 of file Settings.cpp.

4.4.2.14 std::vector< std::string > & Settings::split (const std::string & *str,* char *delim,* std::vector< std::string > & *elems*)

Used to tokenize strings.

Definition at line 6 of file Settings.cpp.

4.4.2.15 std::vector< std::string > Settings::split (const std::string & *str,* char *delim*)

Used to tokenize strings.

Definition at line 20 of file Settings.cpp.

4.4.3 Variable Documentation

4.4.3.1 std::string Settings::download_directory `[static]`

Keeps track of the current download directory.

Definition at line 34 of file Settings.h.

4.4.3.2 std::string Settings::ip_address [static]

Keeps track of the current ip address.

Definition at line 31 of file Settings.h.

4.4.3.3 double Settings::max_download_speed [static]

Maximum download speed.

Definition at line 43 of file Settings.h.

4.4.3.4 pthread_mutex_t Settings::max_mutex [static]

Mutex to make max speed variables thread safe.

Definition at line 40 of file Settings.h.

4.4.3.5 double Settings::max_upload_speed [static]

Maximum upload speed.

Definition at line 46 of file Settings.h.

4.4.3.6 pthread_mutex_t Settings::mutex [static]

Mutex to make download_directory variable thread safe.

Definition at line 37 of file Settings.h.

4.4.3.7 std::string Settings::settings_file [static]

Settings file.

Definition at line 49 of file Settings.h.

Chapter 5

Class Documentation

5.1 AlreadyDownloadingException Class Reference

`#include <Exceptions.h>`

Public Member Functions

- AlreadyDownloadingException () throw ()
- ∼AlreadyDownloadingException () throw ()
- virtual const char ∗ what () const throw ()

5.1.1 Detailed Description

Exception thrown when a download is added which was already added before.

Definition at line 40 of file Exceptions.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AlreadyDownloadingException::AlreadyDownloadingException ( ) throw () `[inline]`

Definition at line 43 of file Exceptions.h.

5.1.2.2 AlreadyDownloadingException::∼AlreadyDownloadingException ( ) throw () `[inline]`

Definition at line 44 of file Exceptions.h.

5.1.3 Member Function Documentation

5.1.3.1 virtual const char∗ **AlreadyDownloadingException::what** ( ) const throw () `[inline, virtual]`

Definition at line 46 of file Exceptions.h.

The documentation for this class was generated from the following file:

- include/Exceptions.h

5.2 DownloadManager::Amount Struct Reference

```
#include <DownloadManager.h>
```

Public Attributes

- double amount
- std::string unit

5.2.1 Detailed Description

Definition at line 29 of file DownloadManager.h.

5.2.2 Member Data Documentation

5.2.2.1 double **DownloadManager::Amount::amount**

Definition at line 30 of file DownloadManager.h.

5.2.2.2 std::string **DownloadManager::Amount::unit**

Definition at line 31 of file DownloadManager.h.

The documentation for this struct was generated from the following file:

- include/DownloadManager.h

5.3 CannotResumeException Class Reference

```
#include <Exceptions.h>
```

Public Member Functions

- CannotResumeException () throw ()
- ∼CannotResumeException () throw ()
- virtual const char ∗ what () const throw ()

5.3.1 Detailed Description

Exception thrown when something went wrong while trying to resume a download.

Definition at line 70 of file Exceptions.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 CannotResumeException::CannotResumeException ( ) throw () [inline]

Definition at line 73 of file Exceptions.h.

5.3.2.2 CannotResumeException::∼CannotResumeException ( ) throw () [inline]

Definition at line 74 of file Exceptions.h.

5.3.3 Member Function Documentation

5.3.3.1 virtual const char∗ CannotResumeException::what ( ) const throw () [inline, virtual]

Definition at line 76 of file Exceptions.h.

The documentation for this class was generated from the following file:

- include/Exceptions.h

5.4 Download Class Reference

```
#include <Download.h>
```

Classes

- struct downloadStats

  Struct for holding download statistics.
- struct time

  Struct for holding time data.

Public Member Functions

- void retry ()
- void start ()
- void stop ()
- void pause ()
- void resume ()
- const int getID ()
- const int getStatus ()
- std::string getTrackerAddress ()
- std::string getFilename ()
- std::string getRootHash ()
- struct downloadStats getStatistics ()
- void calculateSpeeds ()
- void calculatePeers ()
- void setDownloadSpeed (double speed)
- void setUploadSpeed (double speed)
- void setProgress (double percentage)
- bool isComplete ()
- void setSeeders (int amount)
- void setPeers (int amount)
- void calculateEstimatedTime ()
- void setID (int id)
- void setStatus (int status)
- void limitDownSpeed (double speed)
- void limitUpSpeed (double speed)
- Download (std::string tracker, std::string root_hash, std::string filename)
- ∼Download ()

Protected Attributes

- pthread_mutex_t _mutex

  Mutex to prevent download thread and main thread from accessing same data at the same time.
- pthread_mutex_t _transfer_mutex

  Mutex to prevent download thread and main thread from accessing _transfer at the same time.
- volatile int _status

  Current status of the download.
- std::string _filename

  Name of the download.
- std::string _tracker

  Trackers seeding this download.
- std::string _root_hash

  Root hash needed to start swift download.

- downloadStats _stats

    Struct holding the statistics of the download.
- swift::FileTransfer ∗ _transfer

    Swift FileTransfer needed to access certain data about swift downloads.

5.4.1 Detailed Description

Data structure to store all data regarding swift downloads. Makes use of swift methods.

Definition at line 33 of file Download.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 **Download::Download** (std::string *tracker*, std::string *root_hash*, std::string *filename*) `[inline]`

Constructor.

Definition at line 126 of file Download.h.

5.4.2.2 **Download::∼Download** ( ) `[inline]`

Destructor.

Definition at line 146 of file Download.h.

5.4.3 Member Function Documentation

5.4.3.1 void **Download::calculateEstimatedTime** ( )

Calculates estimated download time.

Definition at line 211 of file Download.cpp.

5.4.3.2 void **Download::calculatePeers** ( )

Calculates number of peers.

Definition at line 127 of file Download.cpp.

5.4.3.3 void **Download::calculateSpeeds** ( )

Calculates current download and upload speeds.

Definition at line 114 of file Download.cpp.

5.4.3.4 std::string Download::getFilename ( )

Getter for the filename.

Definition at line 352 of file Download.cpp.

5.4.3.5 const int Download::getID ( )

Getter for the download ID.

Definition at line 301 of file Download.cpp.

5.4.3.6 std::string Download::getRootHash ( )

Getter for the root hash.

Definition at line 359 of file Download.cpp.

5.4.3.7 Download::downloadStats Download::getStatistics ( ) [read]

Getter for the download statistics.

Definition at line 321 of file Download.cpp.

5.4.3.8 const int Download::getStatus ( )

Getter for the status.

Definition at line 311 of file Download.cpp.

5.4.3.9 std::string Download::getTrackerAddress ( )

Getter for the tracker address.

Definition at line 345 of file Download.cpp.

5.4.3.10 bool Download::isComplete ( )

Determines whether the download is complete

Definition at line 178 of file Download.cpp.

5.4.3.11 void Download::limitDownSpeed (double *speed*)

Limits download speed.

Definition at line 273 of file Download.cpp.

5.4.3.12 void **Download::limitUpSpeed** (double *speed*)

Limits upload speed.

Definition at line 260 of file Download.cpp.

5.4.3.13 void **Download::pause** ( )

Pauses downloading and uploading.

Definition at line 70 of file Download.cpp.

5.4.3.14 void **Download::resume** ( )

Resumes a paused download.

Definition at line 90 of file Download.cpp.

5.4.3.15 void **Download::retry** ( )

Deletes the downloaded content and try again from the beginning.

Definition at line 29 of file Download.cpp.

5.4.3.16 void **Download::setDownloadSpeed** (double *speed*)

Setter for download speed.

Parameters

speed,:	The speed in Kb/sec.

Definition at line 140 of file Download.cpp.

5.4.3.17 void **Download::setID** (int *id*)

Setter for download ID.

Parameters

id,:	The download id given by swift::Open.

Definition at line 246 of file Download.cpp.

5.4.3.18 void **Download::setPeers** (int *amount*)

Setter for amount of peers.

Parameters

amount,:	The amount of peers to be set.

Definition at line 199 of file Download.cpp.

5.4.3.19 void **Download::setProgress** (double *percentage*)

Setter for download progress.

Parameters

percentage,:	The percentage to be set between 0 and 100.

Definition at line 166 of file Download.cpp.

5.4.3.20 void **Download::setSeeders** (int *amount*)

Setter for amount of seeders.

Parameters

amount,:	The amount of seeders to be set.

Definition at line 186 of file Download.cpp.

5.4.3.21 void **Download::setStatus** (int *status*)

Setter for status

Parameters

status,:	The new status to be set.

Definition at line 287 of file Download.cpp.

5.4.3.22 void **Download::setUploadSpeed** (double *speed*)

Setter for upload speed.

Parameters

speed,:	The speed in Kb/sec.

Definition at line 153 of file Download.cpp.

5.4.3.23 void Download::start ( )

Starts downloading and uploading.

Definition at line 39 of file Download.cpp.

5.4.3.24 void Download::stop ( )

Stops the download and removes all content from disk.

Definition at line 6 of file Download.cpp.

5.4.4 Member Data Documentation

5.4.4.1 std::string Download::_filename [protected]

Name of the download.

Definition at line 45 of file Download.h.

5.4.4.2 pthread_mutex_t Download::_mutex [protected]

Mutex to prevent download thread and main thread from accessing same data at the same time.

Definition at line 36 of file Download.h.

5.4.4.3 std::string Download::_root_hash [protected]

Root hash needed to start swift download.

Definition at line 51 of file Download.h.

5.4.4.4 downloadStats Download::_stats [protected]

Struct holding the statistics of the download.

Definition at line 86 of file Download.h.

5.4.4.5 volatile int Download::_status [protected]

Current status of the download.

Definition at line 42 of file Download.h.

5.4.4.6 std::string Download::_tracker [protected]

Trackers seeding this download.

Definition at line 48 of file Download.h.

5.4.4.7 swift::FileTransfer* Download::_transfer [protected]

Swift FileTransfer needed to access certain data about swift downloads.

Definition at line 89 of file Download.h.

5.4.4.8 pthread_mutex_t Download::_transfer_mutex [protected]

Mutex to prevent download thread and main thread from accessing _transfer at the same time.

Definition at line 39 of file Download.h.

The documentation for this class was generated from the following files:

- include/Download.h
- src/Download.cpp

5.5 DownloadManagerTest Class Reference

Protected Member Functions

- virtual ~DownloadManagerTest ()
- virtual void SetUp ()
- virtual void TearDown ()

5.5.1 Detailed Description

This is the class for testing DownloadManager.cpp. The setup clear the download list. The teardown also clears the download list and removes all downloaded files.

Definition at line 13 of file DownloadManagerTest.cpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 virtual DownloadManagerTest::~DownloadManagerTest ( ) [inline, protected, virtual]

Definition at line 16 of file DownloadManagerTest.cpp.

5.5.3 Member Function Documentation

5.5.3.1 virtual void DownloadManagerTest::SetUp ( ) `[inline, protected, virtual]`

Definition at line 18 of file DownloadManagerTest.cpp.

5.5.3.2 virtual void DownloadManagerTest::TearDown ( ) `[inline, protected, virtual]`

Definition at line 22 of file DownloadManagerTest.cpp.

The documentation for this class was generated from the following file:

- test/testsuite/DownloadManagerTest.cpp

5.6 Download::downloadStats Struct Reference

Struct for holding download statistics.

`#include <Download.h>`

Public Attributes

- int id

  Download id needed to check the stats.
- double download_speed

  Current download speed in kb/s.
- double upload_speed

  Current upload speed in kb/s.
- double download_percentage

  Download progress in percentage.
- int seeders

  Number of seeders uploading this file.
- int peers

  Number of peers connected to us for this file.
- struct time estimated

  Estimated time left for download to finish.

5.6.1 Detailed Description

Struct for holding download statistics.

Definition at line 62 of file Download.h.

5.6.2 Member Data Documentation

5.6.2.1 double **Download::downloadStats::download_percentage**

Download progress in percentage.

Definition at line 73 of file Download.h.

5.6.2.2 double **Download::downloadStats::download_speed**

Current download speed in kb/s.

Definition at line 67 of file Download.h.

5.6.2.3 struct **time Download::downloadStats::estimated**

Estimated time left for download to finish.

Definition at line 82 of file Download.h.

5.6.2.4 int **Download::downloadStats::id**

Download id needed to check the stats.

Definition at line 64 of file Download.h.

5.6.2.5 int **Download::downloadStats::peers**

Number of peers connected to us for this file.

Definition at line 79 of file Download.h.

5.6.2.6 int **Download::downloadStats::seeders**

Number of seeders uploading this file.

Definition at line 76 of file Download.h.

5.6.2.7 double **Download::downloadStats::upload_speed**

Current upload speed in kb/s.

Definition at line 70 of file Download.h.

The documentation for this struct was generated from the following file:

- include/Download.h

5.7 DownloadTest Class Reference

Protected Member Functions

- virtual ∼DownloadTest ()
- virtual void SetUp ()
- virtual void TearDown ()

Protected Attributes

- Download ∗ download
- std::string tracker
- std::string root_hash
- std::string filename
- int id

5.7.1 Detailed Description

This is the class for testing Download.cpp. The setup creates a new Download with a hard coded tracker, hash and filename. The teardown removes any files that get created.

Definition at line 11 of file DownloadTest.cpp.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 virtual **DownloadTest::∼DownloadTest ( )** `[inline, protected, virtual]`

Definition at line 20 of file DownloadTest.cpp.

5.7.3 Member Function Documentation

5.7.3.1 virtual void **DownloadTest::SetUp ( )** `[inline, protected, virtual]`

Definition at line 22 of file DownloadTest.cpp.

5.7.3.2 virtual void **DownloadTest::TearDown ( )** `[inline, protected, virtual]`

Definition at line 29 of file DownloadTest.cpp.

5.7.4 Member Data Documentation

5.7.4.1 Download∗ DownloadTest::download [protected]

Definition at line 14 of file DownloadTest.cpp.

5.7.4.2 std::string DownloadTest::filename [protected]

Definition at line 17 of file DownloadTest.cpp.

5.7.4.3 int DownloadTest::id [protected]

Definition at line 18 of file DownloadTest.cpp.

5.7.4.4 std::string DownloadTest::root_hash [protected]

Definition at line 16 of file DownloadTest.cpp.

5.7.4.5 std::string DownloadTest::tracker [protected]

Definition at line 15 of file DownloadTest.cpp.

The documentation for this class was generated from the following file:

- test/testsuite/DownloadTest.cpp

5.8 DownloadWhileStreamingException Class Reference

```
#include <Exceptions.h>
```

Public Member Functions

- DownloadWhileStreamingException () throw ()
- ∼DownloadWhileStreamingException () throw ()
- virtual const char ∗ what () const throw ()

5.8.1 Detailed Description

Exception thrown when a download attempt is made during streaming.

Definition at line 25 of file Exceptions.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 DownloadWhileStreamingException::DownloadWhileStreamingException ( ) throw () [inline]

Definition at line 28 of file Exceptions.h.

5.8.2.2 DownloadWhileStreamingException::∼DownloadWhileStreamingException ( ) throw () [inline]

Definition at line 29 of file Exceptions.h.

5.8.3 Member Function Documentation

5.8.3.1 virtual const char∗ DownloadWhileStreamingException::what ( ) const throw () [inline, virtual]

Definition at line 31 of file Exceptions.h.

The documentation for this class was generated from the following file:

- include/Exceptions.h

5.9 FileNotFoundException Class Reference

```
#include <Exceptions.h>
```

Public Member Functions

- FileNotFoundException () throw ()
- ∼FileNotFoundException () throw ()
- virtual const char ∗ what () const throw ()

5.9.1 Detailed Description

Exception thrown when a file is not found.

Definition at line 10 of file Exceptions.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 FileNotFoundException::FileNotFoundException ( ) throw () [inline]

Definition at line 13 of file Exceptions.h.

5.9.2.2 FileNotFoundException::∼FileNotFoundException ( ) throw () [inline]

Definition at line 14 of file Exceptions.h.

5.9.3 Member Function Documentation

5.9.3.1 virtual const char∗ FileNotFoundException::what ( ) const throw () [inline, virtual]

Definition at line 16 of file Exceptions.h.

The documentation for this class was generated from the following file:

- include/Exceptions.h

5.10 HTTPServerTest Class Reference

Protected Member Functions

- virtual ∼HTTPServerTest ()
- void search ()
- std::vector< struct SearchEngine::result > toVector (std::string response)
- virtual void SetUp ()
- virtual void TearDown ()

Static Protected Member Functions

- static size_t getResponse (char ∗response, size_t size, size_t count, void ∗stream)

Protected Attributes

- CURL ∗ easyHandle
- CURLcode res
- std::string response

5.10.1 Detailed Description

This is the test class for HttpServer.cpp. The setup initialises the Http Client in C++. and also clears the search engine list and downloads list. The teardown is empty.

Definition at line 16 of file HTTPServerTest.cpp.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 virtual **HTTPServerTest::∼HTTPServerTest ( )** `[inline, protected, virtual]`

Definition at line 23 of file HTTPServerTest.cpp.

5.10.3 Member Function Documentation

5.10.3.1 static size_t **HTTPServerTest::getResponse (** char ∗ ***response,*** size_t ***size,*** size_t ***count,*** void ∗ ***stream*** **)** `[inline, static, protected]`

Returns the Http response.

Definition at line 28 of file HTTPServerTest.cpp.

5.10.3.2 void **HTTPServerTest::search ( )** `[inline, protected]`

Do a search in SearchEngine.cpp.

Definition at line 37 of file HTTPServerTest.cpp.

5.10.3.3 virtual void **HTTPServerTest::SetUp ( )** `[inline, protected, virtual]`

Definition at line 84 of file HTTPServerTest.cpp.

5.10.3.4 virtual void **HTTPServerTest::TearDown ( )** `[inline, protected, virtual]`

Definition at line 94 of file HTTPServerTest.cpp.

5.10.3.5 std::vector<struct **SearchEngine::result**> **HTTPServerTest::toVector (** std::string ***response*** **)** `[inline, protected]`

Turns an XML string into a vector of results.

Definition at line 47 of file HTTPServerTest.cpp.

5.10.4 Member Data Documentation

5.10.4.1 CURL∗ **HTTPServerTest::easyHandle** `[protected]`

Definition at line 19 of file HTTPServerTest.cpp.

5.10.4.2 CURLcode HTTPServerTest::res [protected]

Definition at line 20 of file HTTPServerTest.cpp.

5.10.4.3 std::string HTTPServerTest::response [protected]

Definition at line 21 of file HTTPServerTest.cpp.

The documentation for this class was generated from the following file:

- test/testsuite/HTTPServerTest.cpp

5.11 InvalidIPException Class Reference

```
#include <Exceptions.h>
```

Public Member Functions

- InvalidIPException () throw ()
- ∼InvalidIPException () throw ()
- virtual const char ∗ what () const throw ()

5.11.1 Detailed Description

Exception thrown when the format of the IP address is incorrect.

Definition at line 55 of file Exceptions.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 InvalidIPException::InvalidIPException ( ) throw () [inline]

Definition at line 58 of file Exceptions.h.

5.11.2.2 InvalidIPException::∼InvalidIPException ( ) throw () [inline]

Definition at line 59 of file Exceptions.h.

5.11.3 Member Function Documentation

5.11.3.1 virtual const char∗ InvalidIPException::what ( ) const throw () [inline, virtual]

Definition at line 61 of file Exceptions.h.

The documentation for this class was generated from the following file:

- include/Exceptions.h

5.12 SearchEngine::result Struct Reference

Struct holding the data of a search result.

```
#include <SearchEngine.h>
```

Public Attributes

- std::string filename
- std::string tracker
- std::string hash

5.12.1 Detailed Description

Struct holding the data of a search result.

Definition at line 32 of file SearchEngine.h.

5.12.2 Member Data Documentation

5.12.2.1 std::string **SearchEngine::result::filename**

Definition at line 33 of file SearchEngine.h.

5.12.2.2 std::string **SearchEngine::result::hash**

Definition at line 35 of file SearchEngine.h.

5.12.2.3 std::string **SearchEngine::result::tracker**

Definition at line 34 of file SearchEngine.h.

The documentation for this struct was generated from the following file:

- include/SearchEngine.h

5.13 SearchEngineTest Class Reference

Protected Member Functions

- std::vector< struct SearchEngine::result > toVector (std::string response)
- virtual ~SearchEngineTest ()
- virtual void SetUp ()
- virtual void TearDown ()

5.13.1 Detailed Description

This class tests the SearchEngine. It doesn not test the search function because it is mocked The setup clears the list of search results The tear down is empty

Definition at line 12 of file SearchEngineTest.cpp.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 virtual **SearchEngineTest::~SearchEngineTest ( )** `[inline, protected, virtual]`

Definition at line 55 of file SearchEngineTest.cpp.

5.13.3 Member Function Documentation

5.13.3.1 virtual void **SearchEngineTest::SetUp ( )** `[inline, protected, virtual]`

Definition at line 57 of file SearchEngineTest.cpp.

5.13.3.2 virtual void **SearchEngineTest::TearDown ( )** `[inline, protected, virtual]`

Definition at line 62 of file SearchEngineTest.cpp.

5.13.3.3 std::vector<struct **SearchEngine::result**> **SearchEngineTest::toVector (** std::string *response* **)** `[inline, protected]`

Turns an XML string into a vector of results

Definition at line 18 of file SearchEngineTest.cpp.

The documentation for this class was generated from the following file:

- test/testsuite/SearchEngineTest.cpp

5.14 Stream Class Reference

```
#include <Stream.h>
```

Public Member Functions

- ∼Stream ()
- void start ()
- void stop ()
- void init ()
- void setTracker (std::string tracker)
- void setRoothash (std::string hash)
- void beginStreaming ()
- bool readStreaming ()
- const int getStatus ()
- std::string getTrackerAddress ()
- std::string getRootHash ()
- struct event ∗ getEvent ()
- void setStatus (int status)

Static Public Member Functions

- static Stream ∗ getInstance ()

Protected Attributes

- std::string _tracker

  Trackers seeding this stream.
- std::string _hash

  Root hash of the stream.
- pthread_mutex_t _mutex

  Mutex to prevent download thread and main thread from accessing same data at the same time.
- struct event _evclose

  Event used by libevent to stop a stream.
- volatile bool _streaming

  Boolean to determine whether a stream is opened or not.

Private Member Functions

- Stream ()

Static Private Attributes

- static Stream ∗ _instance

  Singleton instance of Stream class.

5.14.1 Detailed Description

Singleton class for downloading and binding streams. Makes use of swift methods.

Definition at line 17 of file Stream.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 **Stream::Stream ( )** `[inline, private]`

Constructor, made private to implement Singleton pattern.

Definition at line 41 of file Stream.h.

5.14.2.2 **Stream::∼Stream ( )** `[inline]`

Destructor.

Definition at line 51 of file Stream.h.

5.14.3 Member Function Documentation

5.14.3.1 void **Stream::beginStreaming ( )**

Sets stream state to "begin streaming".

Definition at line 16 of file Stream.cpp.

5.14.3.2 event ∗ **Stream::getEvent ( )** `[read]`

Returns the event to start the libevent loop.

Definition at line 79 of file Stream.cpp.

5.14.3.3 **Stream ∗ Stream::getInstance ( )** `[static]`

Returns Singleton instance of own class.

Definition at line 86 of file Stream.cpp.

5.14.3.4 std::string Stream::getRootHash ( )

Returns the root hash.

Definition at line 56 of file Stream.cpp.

5.14.3.5 const int Stream::getStatus ( )

5.14.3.6 std::string Stream::getTrackerAddress ( )

Returns the tracker address.

Definition at line 49 of file Stream.cpp.

5.14.3.7 void Stream::init ( )

Initialises stream state and mutex.

Definition at line 8 of file Stream.cpp.

5.14.3.8 bool Stream::readStreaming ( )

Returns stream state.

Definition at line 26 of file Stream.cpp.

5.14.3.9 void Stream::setRoothash (std::string *hash*)

Sets the root hash of the stream.

Parameters

hash,:	The root hash to be set.

Definition at line 72 of file Stream.cpp.

5.14.3.10 void Stream::setStatus (int *status*)

5.14.3.11 void Stream::setTracker (std::string *tracker*)

Sets the tracker address of the stream.

Parameters

tracker,:	The tracker address to be set.

Definition at line 64 of file Stream.cpp.

5.14.3.12 void Stream::start ( )

Starts the stream.

Definition at line 116 of file Stream.cpp.

5.14.3.13 void Stream::stop ( )

Stops the stream.

Definition at line 96 of file Stream.cpp.

5.14.4 Member Data Documentation

5.14.4.1 struct event Stream::_evclose [protected]

Event used by libevent to stop a stream.

Definition at line 29 of file Stream.h.

5.14.4.2 std::string Stream::_hash [protected]

Root hash of the stream.

Definition at line 23 of file Stream.h.

5.14.4.3 Stream ∗ Stream::_instance [static, private]

Singleton instance of Stream class.

Definition at line 36 of file Stream.h.

5.14.4.4 pthread_mutex_t Stream::_mutex [protected]

Mutex to prevent download thread and main thread from accessing same data at the same time.

Definition at line 26 of file Stream.h.

5.14.4.5 volatile bool Stream::_streaming [protected]

Boolean to determine whether a stream is opened or not.

Definition at line 32 of file Stream.h.

5.14.4.6 std::string Stream::_tracker [protected]

Trackers seeding this stream.

Definition at line 20 of file Stream.h.

The documentation for this class was generated from the following files:

- include/Stream.h
- src/Stream.cpp

5.15 StreamTest Class Reference

Protected Member Functions

- virtual ∼StreamTest ()
- virtual void SetUp ()
- virtual void TearDown ()

5.15.1 Detailed Description

This is the class that tests Stream.cpp. The setup sets the tracker. The teardown stops the stream if it is still active.

Definition at line 12 of file StreamTest.cpp.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 virtual StreamTest::∼StreamTest ( ) `[inline, protected, virtual]`

Definition at line 15 of file StreamTest.cpp.

5.15.3 Member Function Documentation

5.15.3.1 virtual void StreamTest::SetUp ( ) `[inline, protected, virtual]`

Definition at line 17 of file StreamTest.cpp.

5.15.3.2 virtual void StreamTest::TearDown ( ) `[inline, protected, virtual]`

Definition at line 22 of file StreamTest.cpp.

The documentation for this class was generated from the following file:

- test/testsuite/StreamTest.cpp

5.16 Download::time Struct Reference

Struct for holding time data.

```
#include <Download.h>
```

Public Attributes

- int days
- int hours
- int minutes
- int seconds

5.16.1 Detailed Description

Struct for holding time data.

Definition at line 54 of file Download.h.

5.16.2 Member Data Documentation

5.16.2.1 int **Download::time::days**

Definition at line 55 of file Download.h.

5.16.2.2 int **Download::time::hours**

Definition at line 56 of file Download.h.

5.16.2.3 int **Download::time::minutes**

Definition at line 57 of file Download.h.

5.16.2.4 int **Download::time::seconds**

Definition at line 58 of file Download.h.

The documentation for this struct was generated from the following file:

- include/Download.h

Chapter 6

File Documentation

6.1 include/Download.h File Reference

```
#include <iostream> #include <ctime> #include <string>×
#include <cstdio> #include <cstdlib> #include <float.h>
#include "swift.h" #include "Settings.h"
```

Classes

- class Download
- struct Download::time

  Struct for holding time data.
- struct Download::downloadStats

  Struct for holding download statistics.

Defines

- #define SECONDS_PER_MINUTE (60)
- #define SECONDS_PER_HOUR (SECONDS_PER_MINUTE $*$ SECONDS_PE-R_MINUTE)
- #define SECONDS_PER_DAY (SECONDS_PER_HOUR $*$ 24)

Enumerations

- enum Status { READY, PAUSED, DOWNLOADING, UPLOADING, STOPPED, SWITCHING, STATUS_SIZE }

  All states a Download can be in.

6.1.1 Define Documentation

6.1.1.1 #define SECONDS_PER_DAY (SECONDS_PER_HOUR ∗ 24)

Definition at line 16 of file Download.h.

6.1.1.2 #define SECONDS_PER_HOUR (SECONDS_PER_MINUTE ∗ SECONDS_PER_MINUTE)

Definition at line 15 of file Download.h.

6.1.1.3 #define SECONDS_PER_MINUTE (60)

Definition at line 14 of file Download.h.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum Status

All states a Download can be in.

Enumerator:

READY

PAUSED

DOWNLOADING

UPLOADING

STOPPED

SWITCHING

STATUS_SIZE

Definition at line 19 of file Download.h.

6.2 include/DownloadManager.h File Reference

```
#include <vector> #include <string> #include <dirent.h>
#include <iostream> #include <sstream> #include <cstdio> x
#include <cstdlib> #include <pthread.h> #include <float.-
h> #include "Stream.h" #include "Download.h" #include
"swift.h" #include "ticpp.h" #include "Exceptions.h"
```

Classes

- struct DownloadManager::Amount

Namespaces

- namespace DownloadManager

Defines

- #define UNLIMITED_SPEED 0

Functions

- void DownloadManager::calculateRatio ()
- void DownloadManager::calculateDownloadAmount ()
- void DownloadManager::calculateUploadAmount ()
- void DownloadManager::setActiveDownload (Download ∗download)
- Download DownloadManager::getActiveDownload ()
- std::vector< Download > DownloadManager::getDownloads ()
- void DownloadManager::init ()
- void DownloadManager::startStream (std::string tracker, std::string hash)
- void DownloadManager::stopStream ()
- void DownloadManager::pauseAllDownloads ()
- void DownloadManager::resumeAllDownloads ()
- void ∗ DownloadManager::startStreamThread (void ∗arg)
- void DownloadManager::updateDownloadStatistics ()
- std::string DownloadManager::buildXML ()
- void ∗ DownloadManager::dispatch (void ∗arg)
- int DownloadManager::resumeDownload (std::string download_hash)
- int DownloadManager::startDownload (const std::string download_hash)
- int DownloadManager::getIndexFromHash (const std::string download_hash)
- struct Amount DownloadManager::getDownloadAmount ()
- struct Amount DownloadManager::getUploadAmount ()
- double DownloadManager::getRatio ()
- double DownloadManager::getMaxUpSpeed ()
- double DownloadManager::getMaxDownSpeed ()
- void DownloadManager::setMaxUpSpeed (double speed)
- void DownloadManager::setMaxDownSpeed (double speed)
- void DownloadManager::limitUpSpeeds (double speed)
- void DownloadManager::limitDownSpeeds (double speed)
- void DownloadManager::upload (std::string filename)
- void DownloadManager::startUploads ()
- void DownloadManager::downloadFirstInList ()
- void DownloadManager::add (Download ∗download)
- void DownloadManager::pauseDownload (const std::string download_hash)
- void DownloadManager::switchDownload (std::string hash)
- void DownloadManager::removeFromList (const std::string download_hash)
- void DownloadManager::removeFromDisk (const std::string download_hash)
- void DownloadManager::clearList ()

Variables

- static std::vector< Download > DownloadManager::downloads

  Vector containing all downloads.
- static struct event DownloadManager::evcompl

  Event needed to start download loop.
- static Download $*$ DownloadManager::active_download

  The currently active download.
- static pthread_t DownloadManager::streaming_thread

  Thread needed to play streams.
- static pthread_t DownloadManager::thread

  Thread needed to download and upload.
- static pthread_mutex_t DownloadManager::mutex

  Mutex to make downloads vector thread safe.
- static pthread_mutex_t DownloadManager::active_download_mutex

  Mutex to make active_download thread safe.
- static ticpp::Document $*$ DownloadManager::doc

  XML Document for the download statistics.
- static double DownloadManager::ratio

  Upload amount divided by download amount.
- static double DownloadManager::downloaded

  Total amount of bytes downloaded this session.
- static double DownloadManager::uploaded

  Total amount of bytes uploaded this session.
- static int DownloadManager::d_pid = -1

  Download thread pid.
- static double DownloadManager::max_upspeed

  Max upload speed.
- static double DownloadManager::max_downspeed

  Max download speed.

6.2.1 Define Documentation

6.2.1.1 #define UNLIMITED_SPEED 0

Definition at line 21 of file DownloadManager.h.

6.3 include/Exceptions.h File Reference

`#include <exception> #include <string>`

Classes

- class FileNotFoundException
- class DownloadWhileStreamingException
- class AlreadyDownloadingException
- class InvalidIPException
- class CannotResumeException

6.4 include/HttpServer.h File Reference

`#include <iostream> #include <cstdio> #include <cstdlib> × #include <string.h> #include <event2/event.h> #include <event2/http.h> #include <event2/buffer.h> #include <event2/util.- h> #include <event2/event-config.h> #include <event2/thread.- h> #include "Exceptions.h" #include "Download.h" #include "Stream.h" #include "DownloadManager.h" #include "Search- Engine.h" #include "swift.h" #include "Settings.h"`

Namespaces

- namespace HttpServer

Functions

- static void HttpServer::sendXMLResponse (std::string msg, struct evhttp_request ∗req, struct evbuffer ∗buf)
- static void HttpServer::sendResponse (struct evhttp_request ∗req, struct evbuffer ∗buf, const char ∗message)
- static void HttpServer::handleRequest (struct evhttp_request ∗req, void ∗arg)
- int HttpServer::init ()

6.5 include/SearchEngine.h File Reference

`#include <vector> #include <string> #include <iostream> #include <pthread.h> #include "Exceptions.h" #include "- Settings.h" #include "ticpp.h"`

Classes

- struct SearchEngine::result

  Struct holding the data of a search result.

Namespaces

- namespace SearchEngine

Functions

- void ∗ SearchEngine::startDispersy (void ∗arg)
- void SearchEngine::clearSearchResults ()
- std::string SearchEngine::buildSearchXML ()
- void SearchEngine::search (std::string search_term)
- std::string SearchEngine::getResults ()
- struct result SearchEngine::getResultWithHash (std::string hash)
- struct result SearchEngine::getResultWithName (std::string name)
- void SearchEngine::init ()

Variables

- static pthread_t SearchEngine::dispersy_thread

  Thread to run dispersy module in.
- static pthread_mutex_t SearchEngine::dispersy_mutex

  Mutex to protect dispersy thread.
- static std::vector< struct result > SearchEngine::search_results

  Vector of all search results.
- static ticpp::Document ∗ SearchEngine::searchdoc

  Document used to build the list of search results.

6.6 include/Settings.h File Reference

```
#include <cstdio> #include <cstdlib> #include <iostream> ×
#include <sstream> #include <fstream> #include <sys/types.-
h> #include <sys/stat.h> #include <unistd.h> #include
<netinet/in.h> #include <arpa/inet.h> #include <ifaddrs.-
h> #include <string.h> #include <vector> #include "-
Exceptions.h"
```

Namespaces

- namespace Settings

Defines

- #define DEFAULT_SETTINGS_FILE "/mtd_down/widgets/user/SamyGO/SamyGO/.p2psettings"
- #define DEFAULT_DOWNLOAD_DIR "/dtv/usb/sda1/Downloads"
- #define DEFAULT_IP "127.0.0.1"
- #define DEFAULT_PORT 7758
- #define DHT_PORT 9999

Functions

- void Settings::init (std::string download_dir)
- void Settings::loadSettings (std::string filename)
- void Settings::saveSettings (std::vector< std::string > settings)
- void Settings::setDownloadDirectory (std::string dir)
- void Settings::setIP (std::string ip)
- void Settings::setMaxUpSpeed (double speed)
- void Settings::setMaxDownSpeed (double speed)
- std::vector< std::string > & Settings::split (const std::string &str, char delim, std::vector< std::string > &elems)
- std::vector< std::string > Settings::split (const std::string &str, char delim)
- std::string Settings::getIP ()
- std::string Settings::getDownloadDirectory ()
- bool Settings::directoryExists (std::string path)
- double Settings::getMaxUpSpeed ()
- double Settings::getMaxDownSpeed ()
- std::string Settings::replaceSubstring (std::string str, const std::string from, const std::string to)

Variables

- static std::string Settings::ip_address

  Keeps track of the current ip address.
- static std::string Settings::download_directory

  Keeps track of the current download directory.
- static pthread_mutex_t Settings::mutex

  Mutex to make download_directory variable thread safe.
- static pthread_mutex_t Settings::max_mutex

  Mutex to make max speed variables thread safe.
- static double Settings::max_download_speed

  Maximum download speed.
- static double Settings::max_upload_speed

  Maximum upload speed.
- static std::string Settings::settings_file

  Settings file.

6.6.1 Define Documentation

6.6.1.1 #define DEFAULT_DOWNLOAD_DIR "/dtv/usb/sda1/Downloads"

Definition at line 21 of file Settings.h.

6.6.1.2 #define DEFAULT_IP "127.0.0.1"

Definition at line 22 of file Settings.h.

6.6.1.3 #define DEFAULT_PORT 7758

Definition at line 23 of file Settings.h.

6.6.1.4 #define DEFAULT_SETTINGS_FILE "/mtd_down/widgets/user/SamyGO/SamyG-O/.p2psettings"

Definition at line 20 of file Settings.h.

6.6.1.5 #define DHT_PORT 9999

Definition at line 24 of file Settings.h.

6.7 include/Stream.h File Reference

```
#include <iostream>  #include <string>  #include <ctime>
#include <cstdio> #include <cstdlib> #include "Settings.-
h" #include "swift.h"
```

Classes

- class Stream

6.8 src/Download.cpp File Reference

```
#include "Download.h"
```

6.9 src/DownloadManager.cpp File Reference

```
#include "DownloadManager.h"
```

Functions

- template<class Type >
  void fillXMLValue (Type value, ticpp::Element ∗tag)
- void buildSizeTag (int index, ticpp::Element ∗download_tag)
- void buildCompletedTag (int index, ticpp::Element ∗download_tag)
- void buildStatusTag (int index, ticpp::Element ∗download_tag)
- void buildNameTag (int index, ticpp::Element ∗download_tag)
- void buildDSpeedTag (int index, ticpp::Element ∗download_tag)
- void buildUSpeedTag (int index, ticpp::Element ∗download_tag)
- void buildProgressTag (int index, ticpp::Element ∗download_tag)
- void buildRatioTag (ticpp::Element ∗download_tag)
- void buildUploadAmountTag (ticpp::Element ∗download_tag)
- void buildDownloadAmountTag (ticpp::Element ∗download_tag)
- void buildSeedersTag (int index, ticpp::Element ∗download_tag)
- void buildPeersTag (int index, ticpp::Element ∗download_tag)
- void buildHashTag (int index, ticpp::Element ∗download_tag)
- void buildTimeTag (int index, ticpp::Element ∗download_tag)
- void downloadCallback (int fd, short event, void ∗arg)
- bool fileExists (const std::string filename)

6.9.1 Function Documentation

6.9.1.1 void **buildCompletedTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the COMPLETED tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the COMPLETED tag is nested.

Definition at line 124 of file DownloadManager.cpp.

6.9.1.2 void **buildDownloadAmountTag** (ticpp::Element ∗ *download_tag*)

Builds the DOWNLOADAMOUNT tag for the statistics XML document.

Parameters

download_-tag,:	The DOWNLOAD tag in which the DOWNLOADAMOUNT tag is nested.

Definition at line 229 of file DownloadManager.cpp.

6.9.1.3 void **buildDSpeedTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the DSPEED tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the DSPEED tag is nested.

Definition at line 163 of file DownloadManager.cpp.

6.9.1.4 void **buildHashTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the HASH tag for the statistics XML document.

Parameters

index,:	index of the download from which the root hash is needed.
download_-tag,:	The DOWNLOAD tag in which the HASH tag is nested.

Definition at line 272 of file DownloadManager.cpp.

6.9.1.5 void **buildNameTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the NAME tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the NAME tag is nested.

Definition at line 150 of file DownloadManager.cpp.

6.9.1.6 void **buildPeersTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the PEERS tag for the statistics XML document.

Parameters

index,:	index of the download from which the no. of peers is needed.
download_-tag,:	The DOWNLOAD tag in which the PEERS tag is nested.

Definition at line 259 of file DownloadManager.cpp.

6.9.1.7 void **buildProgressTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the PROGRESS tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the PROGRESS tag is nested.

Definition at line 189 of file DownloadManager.cpp.

6.9.1.8 void **buildRatioTag** (ticpp::Element ∗ *download_tag*)

Builds the RATIO tag for the statistics XML document.

Parameters

download_-tag,:	The DOWNLOAD tag in which the RATIO tag is nested.

Definition at line 201 of file DownloadManager.cpp.

6.9.1.9 void **buildSeedersTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the SEEDERS tag for the statistics XML document.

Parameters

index,:	index of the download from which the no. of seeders is needed.
download_-tag,:	The DOWNLOAD tag in which the SEEDERS tag is nested.

Definition at line 246 of file DownloadManager.cpp.

6.9.1.10 void **buildSizeTag** (int *index*, ticpp::Element ∗ *download_tag*)

Builds the SIZE tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the SIZE tag is nested.

Definition at line 111 of file DownloadManager.cpp.

6.9.1.11 void **buildStatusTag** (int *index,* ticpp::Element ∗ *download_tag*)

Builds the STATUS tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the STATUS tag is nested.

Definition at line 137 of file DownloadManager.cpp.

6.9.1.12 void **buildTimeTag** (int *index,* ticpp::Element ∗ *download_tag*)

Builds the TIME tags for the statistics XML document.

Parameters

index,:	index of the download from which the estimated times are needed.
download_-tag,:	The DOWNLOAD tag in which the TIME tags are nested.

Definition at line 285 of file DownloadManager.cpp.

6.9.1.13 void **buildUploadAmountTag** (ticpp::Element ∗ *download_tag*)

Builds the UPLOADAMOUNT tag for the statistics XML document.

Parameters

download_-tag,:	The DOWNLOAD tag in which the UPLOADAMOUNT tag is nested.

Definition at line 213 of file DownloadManager.cpp.

6.9.1.14 void **buildUSpeedTag** (int *index,* ticpp::Element ∗ *download_tag*)

Builds the USPEED tag for the statistics XML document.

Parameters

index,:	index of the download from which the size is needed.
download_-tag,:	The DOWNLOAD tag in which the USPEED tag is nested.

Definition at line 176 of file DownloadManager.cpp.

6.9.1.15 void **downloadCallback** (int *fd,* short *event,* void ∗ *arg*)

Callback to keep all downloads and uploads running.

Parameters

fd,:	The file descriptor used by swift.
event,:	The event it gets from libevent.
arg,:	Unused argument from libevent.

Definition at line 369 of file DownloadManager.cpp.

6.9.1.16 bool **fileExists** (const std::string *filename*)

Checks whether a file exists.

Parameters

filename,:	Name of the file to be checked.

Definition at line 627 of file DownloadManager.cpp.

6.9.1.17 template<class Type > void **fillXMLValue** (Type *value,* ticpp::Element ∗ *tag*)

Fills in XML tags with the specified value.

Parameters

value,:	The value to be set.
tag,:	The tag to be filled with the value.

Definition at line 101 of file DownloadManager.cpp.

6.10 src/HttpServer.cpp File Reference

```
#include "HttpServer.h"
```

Namespaces

- namespace HttpServer

Functions

- static std::string searchRequest (std::string search_term)
- static std::string addRequest (std::string hash)

- static std::string downloadRequest (std::string hash)
- static std::string uploadRequest (std::string filename)
- static std::string stopRequest (std::string hash)
- static std::string removeRequest (std::string hash)
- static std::string pauseRequest (std::string hash)
- static std::string resumeRequest (std::string hash)
- static std::string streamRequest (std::string hash)
- static std::string settingsRequest (std::vector< std::string > result)
- static std::string clearRequest ()

Variables

- struct event_base * HttpServer::base

6.10.1 Function Documentation

6.10.1.1 static std::string **addRequest** (std::string *hash*) `[static]`

Handler for the /add request.

Definition at line 70 of file HttpServer.cpp.

6.10.1.2 static std::string **clearRequest** ( ) `[static]`

Handler for the /clear request.

Definition at line 252 of file HttpServer.cpp.

6.10.1.3 static std::string **downloadRequest** (std::string *hash*) `[static]`

Handler for the /download request.

Definition at line 93 of file HttpServer.cpp.

6.10.1.4 static std::string **pauseRequest** (std::string *hash*) `[static]`

Handler for the /pause request.

Definition at line 173 of file HttpServer.cpp.

6.10.1.5 static std::string **removeRequest** (std::string *hash*) `[static]`

Handler for the /remove request.

Definition at line 155 of file HttpServer.cpp.

6.10.1.6 static std::string **resumeRequest** (std::string *hash*) `[static]`

Handler for the /resume request.

Definition at line 191 of file HttpServer.cpp.

6.10.1.7 static std::string **searchRequest** (std::string *search_term*) `[static]`

Handler for the /search request.

Definition at line 50 of file HttpServer.cpp.

6.10.1.8 static std::string **settingsRequest** (std::vector< std::string > *result*) `[static]`

Handler for the /settings request.

Definition at line 234 of file HttpServer.cpp.

6.10.1.9 static std::string **stopRequest** (std::string *hash*) `[static]`

Handler for the /stop request.

Definition at line 137 of file HttpServer.cpp.

6.10.1.10 static std::string **streamRequest** (std::string *hash*) `[static]`

Handler for the /stream request.

Definition at line 213 of file HttpServer.cpp.

6.10.1.11 static std::string **uploadRequest** (std::string *filename*) `[static]`

Handler for the /upload request.

Definition at line 117 of file HttpServer.cpp.

6.11 src/Main.cpp File Reference

```
#include <iostream> #include "ticpp.h" #include "swift.-
h" #include "DownloadManager.h" #include "HttpServer.h" ×
#include "SearchEngine.h" #include "Settings.h" #include
<stdio.h> #include <string.h>
```

Functions

- bool InstallHTTPGateway (struct event_base ∗evbase, swift::Address addr, uint32_t chunk_size, double ∗maxspeed)
- bool InstallStatsGateway (struct event_base ∗evbase, swift::Address addr)
- int main ()

6.11.1 Function Documentation

6.11.1.1 bool **InstallHTTPGateway** (struct event_base ∗ ***evbase,*** swift::Address ***addr,*** uint32_t ***chunk_size,*** double ∗ ***maxspeed***)

Define the InstallHTTPGateway method in httpgw.cpp.

6.11.1.2 bool **InstallStatsGateway** (struct event_base ∗ ***evbase,*** swift::Address ***addr***)

6.11.1.3 int **main** ( )

Application main loop.

Definition at line 42 of file Main.cpp.

6.12 test/testsuite/Main.cpp File Reference

```
#include <iostream> #include "curl.h" #include "easy.h"×
#include "gtest.h" #include "swift.h" #include "Download-
Manager.h"#include "HttpServer.h"#include "SearchEngine.-
h"#include "Settings.h"#include <pthread.h>
```

Functions

- bool InstallHTTPGateway (struct event_base ∗evbase, swift::Address addr, uint32_t chunk_size, double ∗maxspeed)
- void ∗ serverCallback (void ∗args)
- int main (int argc, char ∗∗argv)

Variables

- pthread_t server_thread

6.12.1 Function Documentation

6.12.1.1 bool InstallHTTPGateway (struct event_base ∗ *evbase*, swift::Address *addr*, uint32_t *chunk_size*, double ∗ *maxspeed*)

Define the InstallHTTPGateway method in httpgw.cpp.

6.12.1.2 int main (int *argc*, char ∗∗ *argv*)

Definition at line 27 of file Main.cpp.

6.12.1.3 void∗ serverCallback (void ∗ *args*)

Definition at line 22 of file Main.cpp.

6.12.2 Variable Documentation

6.12.2.1 pthread_t server_thread

Definition at line 15 of file Main.cpp.

6.13 src/SearchEngine.cpp File Reference

`#include "SearchEngine.h" #include <Python.h>`

Namespaces

- namespace SearchEngine

Variables

- PyObject ∗ SearchEngine::p_module_name
- PyObject ∗ SearchEngine::p_module
- PyObject ∗ SearchEngine::p_main
- PyObject ∗ SearchEngine::p_function
- PyObject ∗ SearchEngine::p_args
- PyObject ∗ SearchEngine::p_main_value
- PyObject ∗ SearchEngine::p_function_value
- PyObject ∗ SearchEngine::p_result_string
- PyGILState_STATE SearchEngine::gstate
- pthread_mutex_t SearchEngine::gstate_mutex
- PyThreadState ∗ SearchEngine::save

6.14 src/Settings.cpp File Reference

```
#include "Settings.h"
```

6.15 src/Stream.cpp File Reference

```
#include "Stream.h"
```

Functions

- void closeCallback (int fd, short event, void ∗arg)

6.15.1 Function Documentation

6.15.1.1 void **closeCallback** (int *fd,* short *event,* void ∗ *arg*)

Libevent loop for streaming files, also used to close stream.

Parameters

fd,:	The file descriptor used by swift.
event,:	The event it gets from libevent.
arg,:	Argument taken by callback.

Definition at line 39 of file Stream.cpp.

6.16 test/testsuite/DownloadManagerTest.cpp File Reference

```
#include "DownloadManager.h" #include "Download.h" #include
"Stream.h" #include "Settings.h" #include "gtest.h" #include
<string>
```

Classes

- class DownloadManagerTest

Functions

- void testDownloadsAreEqual (Download dl1, Download dl2)
- TEST_F (DownloadManagerTest, setDirectoryTrivial)
- TEST_F (DownloadManagerTest, getIndexFromHashTrivial)
- TEST_F (DownloadManagerTest, getIndexFromHashNonexistent)

- TEST_F (DownloadManagerTest, addDownloadTrivial)
- TEST_F (DownloadManagerTest, addDownloadExists)
- TEST_F (DownloadManagerTest, downloadFirstInListTrivial)
- TEST_F (DownloadManagerTest, downloadFirstInListAlreadyDownloading)
- TEST_F (DownloadManagerTest, clearListTrivial)
- TEST_F (DownloadManagerTest, clearListEmpty)
- TEST_F (DownloadManagerTest, getActiveDownloadTrivial)
- TEST_F (DownloadManagerTest, startDownloadTrivial)
- TEST_F (DownloadManagerTest, startDownloadNonexistent)
- TEST_F (DownloadManagerTest, removeFromListTrivial)
- TEST_F (DownloadManagerTest, removeFromListActiveDownload)
- TEST_F (DownloadManagerTest, removeFromListNonexistent)
- TEST_F (DownloadManagerTest, switchDownloadTrivial)
- TEST_F (DownloadManagerTest, switchDownloadSame)
- TEST_F (DownloadManagerTest, switchDownloadNonexistent)
- TEST_F (DownloadManagerTest, startStreamTrivial)
- TEST_F (DownloadManagerTest, startStreamWithActiveDownload)
- TEST_F (DownloadManagerTest, stopStreamTrivial)
- TEST_F (DownloadManagerTest, stopStreamWithActiveDownload)

6.16.1 Function Documentation

6.16.1.1 TEST_F (DownloadManagerTest , setDirectoryTrivial)

Trivial test for setDirectory.

Definition at line 63 of file DownloadManagerTest.cpp.

6.16.1.2 TEST_F (DownloadManagerTest , getIndexFromHashTrivial)

Trivial test for getIndexFromHash.

Definition at line 77 of file DownloadManagerTest.cpp.

6.16.1.3 TEST_F (DownloadManagerTest , getIndexFromHashNonexistent)

Try getting a download with a hash that doesn't exist.

Definition at line 90 of file DownloadManagerTest.cpp.

6.16.1.4 TEST_F (DownloadManagerTest , addDownloadTrivial)

Trivial test for addDownload.

Definition at line 100 of file DownloadManagerTest.cpp.

6.16.1.5 TEST_F (DownloadManagerTest , addDownloadExists)

Try downloading something that is already being download.

Definition at line 116 of file DownloadManagerTest.cpp.

6.16.1.6 TEST_F (DownloadManagerTest , downloadFirstInListTrivial)

Trivial test for downloadFirstInList.

Definition at line 134 of file DownloadManagerTest.cpp.

6.16.1.7 TEST_F (DownloadManagerTest , downloadFirstInListAlreadyDownloading)

Try downloading the first download when already downloading the first.

Definition at line 166 of file DownloadManagerTest.cpp.

6.16.1.8 TEST_F (DownloadManagerTest , clearListTrivial)

Trivial test for clearList.

Definition at line 185 of file DownloadManagerTest.cpp.

6.16.1.9 TEST_F (DownloadManagerTest , clearListEmpty)

Try clearing list when it is already empty.

Definition at line 200 of file DownloadManagerTest.cpp.

6.16.1.10 TEST_F (DownloadManagerTest , getActiveDownloadTrivial)

Trivial test for getActiveDownload.

Definition at line 212 of file DownloadManagerTest.cpp.

6.16.1.11 TEST_F (DownloadManagerTest , startDownloadTrivial)

Trivial test for startDownload.

Definition at line 229 of file DownloadManagerTest.cpp.

6.16.1.12 TEST_F (DownloadManagerTest , startDownloadNonexistent)

Try downloading a file that doesn't exist.

Definition at line 252 of file DownloadManagerTest.cpp.

6.16.1.13 TEST_F (DownloadManagerTest , removeFromListTrivial)

Trivial test for removeFromList.

Definition at line 263 of file DownloadManagerTest.cpp.

6.16.1.14 TEST_F (DownloadManagerTest , removeFromListActiveDownload)

Remove the active download.

Definition at line 285 of file DownloadManagerTest.cpp.

6.16.1.15 TEST_F (DownloadManagerTest , removeFromListNonexistent)

try removing nonexistent file.

Definition at line 311 of file DownloadManagerTest.cpp.

6.16.1.16 TEST_F (DownloadManagerTest , switchDownloadTrivial)

Tivial test for switchDownload.

Definition at line 321 of file DownloadManagerTest.cpp.

6.16.1.17 TEST_F (DownloadManagerTest , switchDownloadSame)

Try switching to the same download.

Definition at line 345 of file DownloadManagerTest.cpp.

6.16.1.18 TEST_F (DownloadManagerTest , switchDownloadNonexistent)

Switch to nonexistent download.

Definition at line 369 of file DownloadManagerTest.cpp.

6.16.1.19 TEST_F (DownloadManagerTest , startStreamTrivial)

Trivial test for starting a stream.

Definition at line 386 of file DownloadManagerTest.cpp.

6.16.1.20 TEST_F (DownloadManagerTest , startStreamWithActiveDownload)

Start a stream while there is an active download.

Definition at line 395 of file DownloadManagerTest.cpp.

6.16.1.21 TEST_F (DownloadManagerTest , stopStreamTrivial)

Trivial test for stopping a stream.

Definition at line 415 of file DownloadManagerTest.cpp.

6.16.1.22 TEST_F (DownloadManagerTest , stopStreamWithActiveDownload)

Stop a stream while there was an active download.

Definition at line 427 of file DownloadManagerTest.cpp.

6.16.1.23 void testDownloadsAreEqual (Download *dl1,* Download *dl2*)

Check whether two downloads are the same.

Definition at line 51 of file DownloadManagerTest.cpp.

6.17 test/testsuite/DownloadTest.cpp File Reference

```
#include "Download.h"#include "gtest.h"#include <string>×
#include <iostream>
```

Classes

- class DownloadTest

Functions

- TEST_F (DownloadTest, constructor)
- TEST_F (DownloadTest, setIDTrivial)
- TEST_F (DownloadTest, setIDNegative)
- TEST_F (DownloadTest, setDownloadSpeedTrivial)
- TEST_F (DownloadTest, setDownloadSpeedNegative)
- TEST_F (DownloadTest, setUploadSpeedTrivial)
- TEST_F (DownloadTest, setUploadSpeedNegative)
- TEST_F (DownloadTest, percentageTrivial)
- TEST_F (DownloadTest, percentageNegative)
- TEST_F (DownloadTest, percentageOver100)
- TEST_F (DownloadTest, seedersTrivial)
- TEST_F (DownloadTest, seedersNegative)
- TEST_F (DownloadTest, peersTrivial)
- TEST_F (DownloadTest, peersNegative)
- TEST_F (DownloadTest, statusTrivial)
- TEST_F (DownloadTest, statusWrong)

- TEST_F (DownloadTest, startTrivial)
- TEST_F (DownloadTest, startAlreadyStarted)
- TEST_F (DownloadTest, pauseTrivial)
- TEST_F (DownloadTest, pauseNeverStarted)
- TEST_F (DownloadTest, pauseAlreadyPaused)
- TEST_F (DownloadTest, resumeTrivial)
- TEST_F (DownloadTest, resumeAlreadyDownloading)
- TEST_F (DownloadTest, stopTrivial)
- TEST_F (DownloadTest, stopNotStarted)
- TEST_F (DownloadTest, stopWhilePaused)
- TEST_F (DownloadTest, stopAlreadyStopped)

6.17.1 Function Documentation

6.17.1.1 TEST_F (DownloadTest , constructor)

Test whether the constructor sets values properly.

Definition at line 52 of file DownloadTest.cpp.

6.17.1.2 TEST_F (DownloadTest , setIDTrivial)

Trivial test for setID.

Definition at line 63 of file DownloadTest.cpp.

6.17.1.3 TEST_F (DownloadTest , setIDNegative)

Try setting the ID to a negative number.

Definition at line 73 of file DownloadTest.cpp.

6.17.1.4 TEST_F (DownloadTest , setDownloadSpeedTrivial)

Trivial test for setDownloadSpeed.

Definition at line 89 of file DownloadTest.cpp.

6.17.1.5 TEST_F (DownloadTest , setDownloadSpeedNegative)

Try setting the downloadspeed to a negative number.

Definition at line 101 of file DownloadTest.cpp.

6.17.1.6 TEST_F (DownloadTest , setUploadSpeedTrivial)

Trivial test for setUploadSpeed.

Definition at line 116 of file DownloadTest.cpp.

6.17.1.7 TEST_F (DownloadTest , setUploadSpeedNegative)

Try setting upload speed to a negative number.

Definition at line 127 of file DownloadTest.cpp.

6.17.1.8 TEST_F (DownloadTest , percentageTrivial)

Trivial test for setPercentage.

Definition at line 142 of file DownloadTest.cpp.

6.17.1.9 TEST_F (DownloadTest , percentageNegative)

Try setting the percentage to a negative number.

Definition at line 153 of file DownloadTest.cpp.

6.17.1.10 TEST_F (DownloadTest , percentageOver100)

Try setting the percentage to a number above 100.

Definition at line 165 of file DownloadTest.cpp.

6.17.1.11 TEST_F (DownloadTest , seedersTrivial)

Trivial test for setSeeders.

Definition at line 179 of file DownloadTest.cpp.

6.17.1.12 TEST_F (DownloadTest , seedersNegative)

Try to set the seeders to a negative.

Definition at line 190 of file DownloadTest.cpp.

6.17.1.13 TEST_F (DownloadTest , peersTrivial)

Trivial test for setPeers.

Definition at line 205 of file DownloadTest.cpp.

6.17.1.14 TEST_F (DownloadTest , peersNegative)

Try setting the peers to a negative number.

Definition at line 216 of file DownloadTest.cpp.

6.17.1.15 TEST_F (DownloadTest , statusTrivial)

Trivial test for setStatus.

Definition at line 235 of file DownloadTest.cpp.

6.17.1.16 TEST_F (DownloadTest , statusWrong)

Try setting the status to a status that doesn't exist.

Definition at line 245 of file DownloadTest.cpp.

6.17.1.17 TEST_F (DownloadTest , startTrivial)

Trivial test for start.

Definition at line 258 of file DownloadTest.cpp.

6.17.1.18 TEST_F (DownloadTest , startAlreadyStarted)

Try starting a download that has already started.

Definition at line 267 of file DownloadTest.cpp.

6.17.1.19 TEST_F (DownloadTest , pauseTrivial)

Trivial test for pause.

Definition at line 278 of file DownloadTest.cpp.

6.17.1.20 TEST_F (DownloadTest , pauseNeverStarted)

Try pausing a download that has not started yet.

Definition at line 288 of file DownloadTest.cpp.

6.17.1.21 TEST_F (DownloadTest , pauseAlreadyPaused)

Try pausing a download that is already paused.

Definition at line 296 of file DownloadTest.cpp.

6.17.1.22 TEST_F (DownloadTest , resumeTrivial)

Trivial test for resume.

Definition at line 308 of file DownloadTest.cpp.

6.17.1.23 TEST_F (DownloadTest , resumeAlreadyDownloading)

Try resuming a download that is already downlaoding.

Definition at line 319 of file DownloadTest.cpp.

6.17.1.24 TEST_F (DownloadTest , stopTrivial)

Trivial test for stop.

Definition at line 333 of file DownloadTest.cpp.

6.17.1.25 TEST_F (DownloadTest , stopNotStarted)

Try stopping a download that never started.

Definition at line 342 of file DownloadTest.cpp.

6.17.1.26 TEST_F (DownloadTest , stopWhilePaused)

Stop a download that is paused.

Definition at line 350 of file DownloadTest.cpp.

6.17.1.27 TEST_F (DownloadTest , stopAlreadyStopped)

Stop a download that is already stopped.

Definition at line 359 of file DownloadTest.cpp.

6.18 test/testsuite/HTTPServerTest.cpp File Reference

```
#include <string>  #include "curl.h"  #include "easy.h"×
#include "HttpServer.h"  #include "gtest.h"  #include "-
SearchEngine.h" #include "DownloadManager.h"
```

Classes

- class HTTPServerTest

Functions

- TEST_F (HTTPServerTest, aliveTest)
- TEST_F (HTTPServerTest, searchTrivial)
- TEST_F (HTTPServerTest, searchEmpty)
- TEST_F (HTTPServerTest, downloadTrivial)
- TEST_F (HTTPServerTest, addNonexistent)
- TEST_F (HTTPServerTest, downloadNonexistent)
- TEST_F (HTTPServerTest, downloadTwice)
- TEST_F (HTTPServerTest, pauseResumePause)
- TEST_F (HTTPServerTest, pauseResumeStats)
- TEST_F (HTTPServerTest, removeTrivial)
- TEST_F (HTTPServerTest, removeActiveDownload)
- TEST_F (HTTPServerTest, removeNonexistent)
- TEST_F (HTTPServerTest, startStreamTrivial)
- TEST_F (HTTPServerTest, stopStreamTrivial)

6.18.1 Function Documentation

6.18.1.1 TEST_F (HTTPServerTest , aliveTest)

Test whether the server is running.

Definition at line 105 of file HTTPServerTest.cpp.

6.18.1.2 TEST_F (HTTPServerTest , searchTrivial)

Test whether a search returns results.

Definition at line 119 of file HTTPServerTest.cpp.

6.18.1.3 TEST_F (HTTPServerTest , searchEmpty)

Test whether an empty searh is handled properly.

Definition at line 141 of file HTTPServerTest.cpp.

6.18.1.4 TEST_F (HTTPServerTest , downloadTrivial)

Test whether you can add and download a file after a search.

Definition at line 165 of file HTTPServerTest.cpp.

6.18.1.5 TEST_F (HTTPServerTest , addNonexistent)

Test whether adding a nonexistent download is handled properly.

Definition at line 198 of file HTTPServerTest.cpp.

6.18.1.6 TEST_F (HTTPServerTest , downloadNonexistent)

Test whether downloading a nonexistent download is handled properly.

Definition at line 210 of file HTTPServerTest.cpp.

6.18.1.7 TEST_F (HTTPServerTest , downloadTwice)

Try downloading the same file twice.

Definition at line 222 of file HTTPServerTest.cpp.

6.18.1.8 TEST_F (HTTPServerTest , pauseResumePause)

Test the infamous pause-resume-pause segfault.

Definition at line 258 of file HTTPServerTest.cpp.

6.18.1.9 TEST_F (HTTPServerTest , pauseResumeStats)

Test the infamous pause-resume-stats segfault.

Definition at line 287 of file HTTPServerTest.cpp.

6.18.1.10 TEST_F (HTTPServerTest , removeTrivial)

Trivial test for remove.

Definition at line 317 of file HTTPServerTest.cpp.

6.18.1.11 TEST_F (HTTPServerTest , removeActiveDownload)

Remove the active download.

Definition at line 345 of file HTTPServerTest.cpp.

6.18.1.12 TEST_F (HTTPServerTest , removeNonexistent)

Try Removing a nonexistent download.

Definition at line 375 of file HTTPServerTest.cpp.

6.18.1.13 TEST_F (HTTPServerTest , startStreamTrivial)

Start a stream.

Definition at line 390 of file HTTPServerTest.cpp.

6.18.1.14 TEST_F (HTTPServerTest , stopStreamTrivial)

Stop a stream.

Definition at line 406 of file HTTPServerTest.cpp.

6.19 test/testsuite/SearchEngineMock.cpp File Reference

```
#include "SearchEngine.h"
```

6.20 test/testsuite/SearchEngineTest.cpp File Reference

```
#include "SearchEngine.h"  #include "gtest.h"  #include "-
Exceptions.h" #include <string>
```

Classes

- class SearchEngineTest

Functions

- TEST_F (SearchEngineTest, getResultsTrivial)
- TEST_F (SearchEngineTest, getResultsEmpty)
- TEST_F (SearchEngineTest, getResultWithNameTrivial)
- TEST_F (SearchEngineTest, getResultWithNameNonexistent)
- TEST_F (SearchEngineTest, getResultWithHashTrivial)
- TEST_F (SearchEngineTest, getResultWithHashNonexistent)

6.20.1 Function Documentation

6.20.1.1 TEST_F (SearchEngineTest , getResultsTrivial)

Trivial test for getResults

Definition at line 70 of file SearchEngineTest.cpp.

6.20.1.2 TEST_F (SearchEngineTest , getResultsEmpty)

Get resuts when there are none

Definition at line 81 of file SearchEngineTest.cpp.

6.20.1.3 TEST_F (SearchEngineTest , getResultWithNameTrivial)

Trivial test for getResultWithName

Definition at line 95 of file SearchEngineTest.cpp.

6.20.1.4 TEST_F (SearchEngineTest , getResultWithNameNonexistent)

Try to get a result by name that doesn't exist

Definition at line 106 of file SearchEngineTest.cpp.

6.20.1.5 TEST_F (SearchEngineTest , getResultWithHashTrivial)

Trivial test for getResultWithHash

Definition at line 118 of file SearchEngineTest.cpp.

6.20.1.6 TEST_F (SearchEngineTest , getResultWithHashNonexistent)

Try to get a result by name that doesn't exist

Definition at line 130 of file SearchEngineTest.cpp.

6.21 test/testsuite/StreamTest.cpp File Reference

```
#include "Download.h" #include "Stream.h" #include "gtest.-
h" #include <string> #include <iostream>
```

Classes

- class StreamTest

Functions

- TEST_F (StreamTest, getInstance)
- TEST_F (StreamTest, startStreamTrivial)
- TEST_F (StreamTest, startTwice)
- TEST_F (StreamTest, startStopStart)
- TEST_F (StreamTest, stopTrivial)
- TEST_F (StreamTest, stopTwice)

6.21.1 Function Documentation

6.21.1.1 TEST_F (StreamTest , getInstance)

Check whether Stream always returns the same instance.

Definition at line 34 of file StreamTest.cpp.

6.21.1.2 TEST_F (StreamTest , startStreamTrivial)

Trivial test for start.

Definition at line 45 of file StreamTest.cpp.

6.21.1.3 TEST_F (StreamTest , startTwice)

Start stream twice.

Definition at line 54 of file StreamTest.cpp.

6.21.1.4 TEST_F (StreamTest , startStopStart)

Start, stop and start the stream again.

Definition at line 64 of file StreamTest.cpp.

6.21.1.5 TEST_F (StreamTest , stopTrivial)

Trivial test for stop.

Definition at line 77 of file StreamTest.cpp.

6.21.1.6 TEST_F (StreamTest , stopTwice)

Stop the stream twice.

Definition at line 87 of file StreamTest.cpp.

Bibliography

[1] Blaise Barney and Lawrence Livermore. *libevent – an event notification library.* 2012. URL: `http://libevent.org/`.

[2] QEMU community. *QEMU wiki.* 2012. URL: `http://wiki.qemu.org/Main_Page`.

[3] SamyGO community. *SamyGO.* 2012. URL: `http://www.samygo.tv/`.

[4] Ubuntu community. *Ubuntu wiki.* 2012. URL: `https://wiki.ubuntu.com/`.

[5] ARM Holdings. *ARM.* 2012. URL: `http://www.arm.com/`.

[6] Aaron Kaluszka. "Distributed Hash Tables". In: (2010).

[7] Nick Mathewson and Niels Provos. *POSIX Threads Programming.* 2012. URL: `https://computing.llnl.gov/tutorials/pthreads/`.

[8] Dr. Ir. J.A. Pouwelse, Dr. V. Grischenko, and A. Bakker. *Swift, the multiparty transport protocol.* 2012. URL: `http://libswift.org/`.

[9] J. A. Pouwelse et al. "TRIBLER: a social-based peer-to-peer system". In: *Concurrency and Computation: Practice and Experience* 20.2 (2008), pp. 127–138. ISSN: 1532-0634. DOI: `10.1002/cpe.1189`. URL: `http://dx.doi.org/10.1002/cpe.1189`.

[10] Boudewijn Schoon. "Dispersy: Distributed Permission System". In: (2010). URL: `http://www.scribd.com/doc/81170037/Boudewijn-Dispersy-Documentation-DRAFT-2010`.

[11] P. Shetty. *QEMU TAP network setup.* 2012. URL: `http://technology-shettyprasad.blogspot.nl/2009/01/qemu-tap-network-setup.html`.

[12] The netfilter core team. *The netfilter.org "iptables" project.* 2012. URL: `http://www.netfilter.org/projects/iptables/index.html`.

[13] Tribler team. *4th Generation of P2P.* 2012. URL: `http://www.tribler.org/trac/wiki/4thGenerationP2P`.

[14] Niels Zeilemaker and Johan Pouwelse. "Tribler: P2P Search, Share and Stream". In: (2012).