rSlidy

Group 4

Elias Zeitfogel, Patrick Kasper, Karina Priebernig, Clemens Meinhart

Information Architecture and Web Usability
Project Paper

31 Jan 2014

Abstract

rSlidy is the responsive version of the original HTML Slidy by Dave Raggett. It has been compeletely rewritten to ensure responsiveness, performance and a good look and feel.

The introduction briefly describes where Slidy comes from, the basic features and what the main objectives are. In chapter 2 our considered approaches are documented before the features are explained in chapter 3. Chapter 4 explains how the migration from Slidy works and finally, chapter 5 covers the technical details.

A quick reference is attached in the appendix.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Everyone who already worked with Microsoft PowerPoint knows how chunky the presentations sometimes get. Not even on purpose it is just that there are so many features to push up each slide that it sometimes gets really annoying for the audience to watch the fancy fade-outs and star sprinkles. Less is more - that is why every presenter needs to focus on the content rather than distracting his or her audience. Besides that, PowerPoint is not portable when it is not saved as a PDF or as a ready to go zip file.

Luckily there is an alternative to Microsoft's presentation tool which is called Slidy.

1.1 Motivation

rSlidy is based on HTML Slidy[1] which was originally developed by Dave Raggett, a member of the W3C team.

The r in rSlidy come from the word responsive. While the original Slidy includes basic functionality like (just to name a few) standard key inputs for switching to the next slide, including bullet points for outlining the content or even a table of content which appears by pressing the Esc-key, rSlidy goes much further. Although it works like the old one, it is completely rewritten and can be considered as a completely new Slidy.

Combining HTML 5, CSS and JS technologies, it can be as easily controlled on a mobile device like a Smartphone or Tablet with touch gestures as on a PC with keyboard inputs.

1.2 Original Slidy

Slidy is an entirely web based alternative to PowerPoint that works in all common browsers. Each presentation is a single XHTML file and each slide is enclosed in a div tag. To run a presentation, the link to the HTML file needs to be opened within a browser.

Besides the functionality mentioned before, the basic features include also a time display in the footer to keep track on how much is already spent while giving a presentation. The features might seem handy and in fact they are, but there are some disadvantages also. It is not very responsive which means that users who want to access the presentation on their mobile devices are facing some troubles with the very main features. Since there is no keyboard which can be used for navigation, tapping the finger on the screen or sliding it to the left or right might work, but not in all cases. Depending on the mobile browser or the operating system, bugs may occur so that the usage is limited.

1.3 Objectives

Since it is always challenging working with the originally basic Slidy when it comes to compatibility among different browsers, rSlidy features a new state-of-the-art code. It is as usable on a PC's browser as on a mobile

[1] http://w3.org/Talks/Tools/Slidy2/

version. Because it is rebuilt from scratch, it can handle touch gestures as well as keyboard inputs. One main goal is covered by this fact.

Another objective is of course improving performance although the code has been rewritten. This goal has been achieved - rSlidy is backward compatible, older slides can be integrated without a loss of performance.

Chapter 2

Considered Approaches

This chapter is a developer's diary that narrates the learning process and experiments with technologies of the authors during the project.

As the proverb goes "The journey is its own reward.", we want to portray our search for the right technologies and list the considered ones that got dropped again. All of them had their up- and downsides, but we tried to keep external dependencies to an absolute minimum.

The initial objectives concentrated on proper rendering and controlling on mobile devices. Early tests proved that the original Slidy code barely ran on most mobile browsers. Hence, our first idea was to extract features from Slidy and add modern alternatives that are activated using compatibility tests and feature detection with Modernizr[1]. The experiments with it resulted in the unanimous decision to drop the original code. From this moment on we knew that a rebuild from scratch was necessary to reinstate Slidy and make it competitive and viable. As we discussed possible approaches we decided that we wanted modern technologies for modern browsers, and use methods that gracefully fail if they are not supported by the browser. That way Modernizr was not necessary anymore and got dropped too.

Pointer event handling caused difficulties until the very end. We tried to implement everything ourselves. But the browsers all handle the events differently, trigger various events in different orders, handle a click and drag or a slide inconsistently. Hacks are necessary to get all of them to work as intended. When we thought we had every problem solved, tests on mobile devices taught us better, because the mobile browsers Google Chrome, Mozilla Firefox, Samsung Internet (Default Browser on Samsung Android devices) and Dolphin are all far away from current standards and of course handled the events in another way again. We did not have an Iphone at our disposal, so no testing was done on Apple's smartphones.

In turn we tried to substitute either pointer event handling, gesture detection or execution with third party software. The first one was Touchy[2], a JavaScript plugin based on jQuery[3]. Touchy reads and provides event data. It offers gesture bindings with various options. But it brought compatibility issues along. Additionally, we wanted to avoid jQuery inclusion for better performance and less unnecessary overhead.

Another JavaScript called HandJS[4] followed Touchy. HandJS implements the new standard introduced by Microsoft called pointer events. A pointer event is a further abstraction of input events to unify input from mouse, pen or touch. With exception of Internet Explorer 10 and 11 modern browsers are not supporting this specification. This is where HandJS comes in. It claims to bring full pointer event functionality and a fall-back to mouse, so developers can write one event handling for all devices and browsers. We build rSlidy around this concept, and it worked perfectly fine on desktop browsers. However, the mobile ones did not trigger slide gestures properly. All those browsers fired different events. The velocity needed for slide gestures were far to high, although we reconfigured and tinkered on the triggers constantly. In the last week of development we

[1] http://modernizr.com/
[2] http://touchyjs.org/
[3] http://jquery.com/
[4] http://handjs.codeplex.com/

finally decided to replace it with hammer.js[5]. Hammer.js does not require jQuery, offers gesture triggers, events and event data. With this JavaScript library, everything worked smoothly.

rSlidy switches slides in the blink of an eye. We wanted something fancier, an animation where the slides scroll horizontally along when switched. In combination with the touch gesture slide we wanted a drag-effect. The slide should stick to the users finger while he or she drags along the screen. This effect cannot be achieved with CSS transitions, only with JavaScript animation. CSS transitions should be the fall-back, and instant switches are the default fall-back when a browser does not support the CSS transitions. But the JavaScript drew heavily on resources. The HTML file containing a slide show tends to grow rather large, and needs be loaded all at once. While it looked well on personal and notebooks computers, smartphones due to their limited hardware were forced to their knees. Again, we dropped this feature in favour of overall performance.

WebSockets are used for the remote control feature. The goal was to give rSlidy something special. Something that goes beyond simply making it work on a mobile platform. So we created a Master-Slave architecture which allows remote controlling other rSlidy clients. Previously ideas like these failed to a lack of options. The old saying goes that websites should be stateless meaning that once the content is loaded no further communication with the server should be required. This is a conflict of interest. But since new interactive websites generally keep talking with the server this was one issue less to care about. Usually websites use AJAX to communicate. This would mean that the server would need to have a queue and the clients poll for updates. This fact combined with AJAX not really being the fastest service meant it was not good for us. WebSockets on the other hand allow exactly what we want. Their only issue was that most browsers had them implemented but deactivated for security reasons. Since they seem to be solved by now the WebSocket standard is good to go allowing rSlidy very low response times without creating any heavy load on the connection which gives us the small special feature we were looking for.

[5] http://eightmedia.github.io/hammer.js/

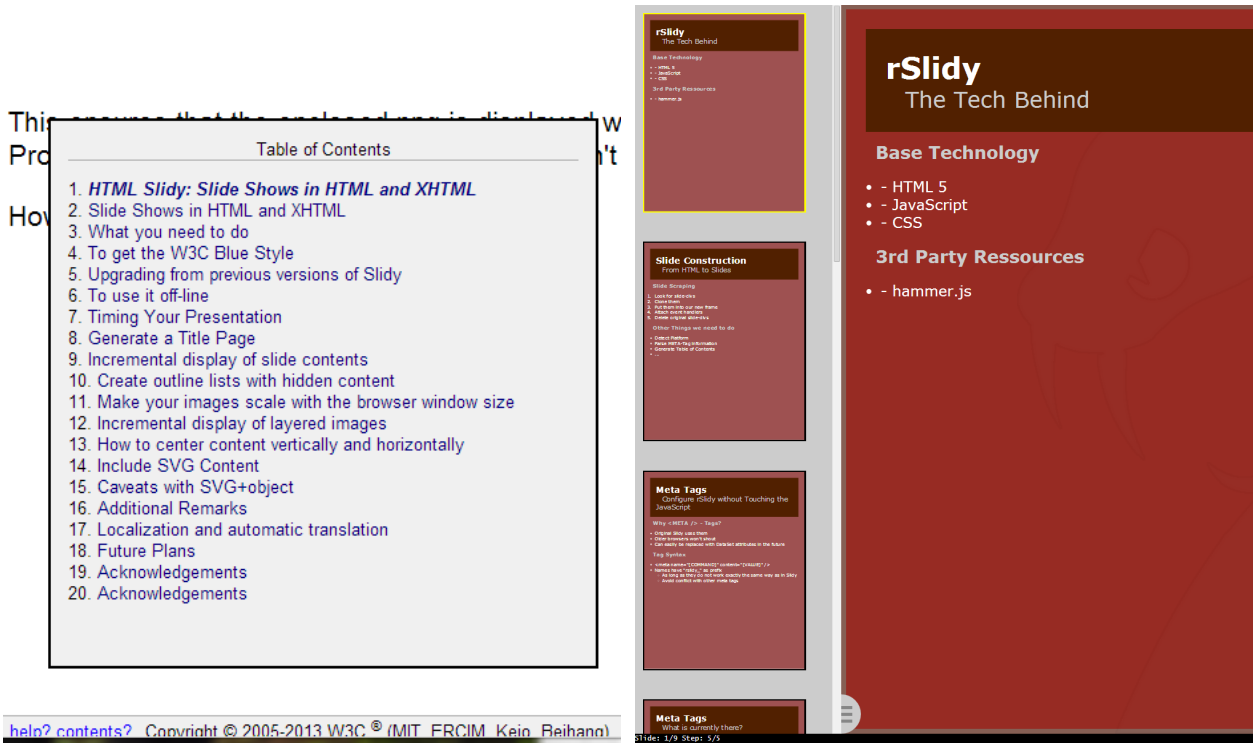
Chapter 3

Features

Based on the idea of the original Slidy, the rSlidy framework lists various features of its predecessor, extended by several new ones. Because Slidy was discontinued in 2010, its inner workings were far from state of the art. A alternation of its code would be time consuming and borderline pointless, therefore we decided to rebuild it from scratch. The objective was to feature new, state-of-the-art code, but still ensured backward compatibility to presentations written for Slidy. We implemented old and new features using current standards of web development, always mindful of supporting most modern browsers with considerable market shares of desktop and mobile devices. Dependencies were removed and avoided where it was possible, and the overall performance noticeably improved. This new framework reads and parses HTML, and converts it into a slide show, just like Slidy. It also uses meta tags and the same HTML-class selectors, mainly *slide* and *incremental* for this purpose. Instead of instant switching of slides rSlidy introduces CSS transitions for animated changing of slides, to evoke a more modern feel.

The status bar has also been conserved, displaying information about the progress of the slide show. While there is functionality for an automatically generated table of contents in Slidy, it is not very user friendly. Its approach is a simple list of headlines from the slides, that appears superimposed and covers the slide. The switch is a link in the status bar. Neither the toggle switch link in the status bar nor the links in the table of contents itself are easily usable on handheld touch devices because of their size (3.1). The rSlidy framework however features a larger menu button, in its default styling a grey semicircle with three horizontal lines on it. These lines are commonly used for menu buttons in modern designs and should be well known. The button is placed on the bottom left corner to ensure best anatomical accessibility for thumbs in case of touch input on smartphones. It can be replaced and redesigned by overriding its CSS styling. The table of contents consists of live previews of the slides instead of pure text, showing the presenter what to expect from each slide. The slide the user is currently viewing is highlighted with a yellow boarder. When activated, the menu slides in from the left and pushes the slide content to the left without covering it. This way, everything stays visible.

Slidy already featured mouse-click input for forward stepping and key input for various functions. A step in the context of Slidy and rSlidy is an incremental object, content that appears step by step on a slide. Controls are very important for a motivating and smooth usage, which is why we put great effort and emphasis on fluent and customizable user input options. In rSlidy the screen is divided in two halves. Mouse clicks or touch taps on the left half triggers a step back, mouse clicks or touch taps on the right half triggers a step forward - except when clicking or touching an object that offers interactivity, e.g. a hyperlink, an embedded video, game or iframe. For touch input rSlidy offers the slide gesture horizontally across the screen to browse whole slides forward and backward. Just like Slidy, rSlidy offers keyboard controls as well. Per default the left and right arrow keys switch full slides backward and forward, and the up and down keys trigger steps forward and backward. The escape key opens and closes the table of contents. If the user wants all input to be ignored for some time, e.g. to mark or copy-paste text from slides, the control key locks the slide show and provides this feature. The status bar will display the state "LOCKED" to remind the user to repress the control key in order to unlock the slide show and continue with it. The controls setup can be customized via meta tags, allowing the user to specify his or her own preferences for navigation input.



(a) Slidy Status Bar and Table of Contents

(b) rSlidy Status Bar and Table of Contents

Figure 3.1: Figure 3.1a shows the status bar and the table of contents as featured in Slidy. The button for the table of contents is the small "contents" link in the bar on the bottom. The status bar is too low positioned, resulting in a partial masking by the boarder of the browser window. The table of contents itself lies on top of the slide and covers it. The links in the table are hard to control using touch input. Figure 3.1a to the right displays the status bar and the table of contents as featured in rSlidy. Instead of text a live preview of the slides is presented. A yellow border marks the current slide. The menu slides in from the left, pushing the content of the current slide to the right and not covering it. It is toggled with the grey semicircle with the three horizontal lines in it, the common symbol for a menu. This button is large enough for touch input and placed to be well reachable with the left thumb.

Deep links are an important concept, that Slidy already provided. By typing "#(X)", where X is the slide number, behind the URL of the slide show in the navigation bar a user could navigate directly to said slide. This feature has been improved by rSlidy, featuring now deep links to the precision of steps. Using the syntax #[X,Y], where X is the number of the slide and Y is the number of the step on slide X, navigation to the slide and the exact step is possible.

The final feature, a purely optional but remarkable one, is remote control. Combined with a Websocket server one can telecontrol his or her slide show. The idea is to connect a handheld device as remote control and the projection device as presenting one. Then, each command entered by the remote control device is mirrored on the presenting device. Optionally the remote control can exclusively display elements like the status bar, the table of contents, the menu button or presenter's notes, while not displaying these elements on the presenting device, because they are of no concern for the audience.

Chapter 4

Migration from Slidy

This chapter will briefly introduce everything that is necessary to get good ol'Slidy or kSlidy working with the new rSlidy. It will further elaborate upon the similarities that still exist and new mechanics introduced with rework. Finally a step by step guide complete with code examples is attached to ease the transition from an older Slidy tool to the new rSlidy.

4.1 Requirements

The migration process described in this chapter is dependent on a few variables, which are crucial nonetheless. To enable a smooth switch to the new rSlidy it is foremost important that the target presentation was created using the original *Slidy* or *kSlidy*. Presentations from other Slidy deviates might also work but have not been tested yet. The new rSlidy ensures full compatibility to any presentations created with Slidy or kSlidy, including any meta-tags that were used for configuration purposes and CSS files that were already in place.

It is important to mention that, while rSlidy enables touch capabilities for a better mobile experience, the original CSS is not overwritten. If the CSS is not adaptive and responsive enough for mobile viewing the general experience might suffer. Thus, although the migrations progress is very quick and easy, one should still bare in mind optimizing the CSS for the mobile experience. The slides are placed in a sandbox and rendered with the provided CSS, rSlidy does not interfere.

Besides that the just a few more things need to be done. First, to get a current version of rSlidy[1], both the JavaScript component *rslidy.js* and the CSS file *rslidy.css*. The JavaScript file handles the technicalities and the CSS file is only responsible for transitions and inclusion of the status bar and the sophisticated table of contents and does not interfere with the CSS of the presentation itself.

The only external JavaScript variable that is used in rSlidy is *Hammer*[2]. Hammer is used to properly handle touch events both on mobile devices as well as on touch-enabled desktop devices and also supports emulation of touch gestures such as swipe and tap with mouse input.

4.2 Migration Steps

Following the requirements described in the section above, the migration process can be explained in a few single steps. The following listing also notes the example files used in the code guide in the next section in parentheses.

- Place the rslidy CSS file in the existing presentations CSS folder (rslidy.css).
- Place both rSlidy's JavaScript file and Hammer's JavaScript file in the existing presentations SCRIPTS folder (rslidy.js, hammer.js).

[1] rSlidy is availabe here: `https://github.com/pkasper/rSlidy`

[2] Hammer.js is available here: http://eightmedia.github.io/hammer.js/

css/	
	style.css
scripts/	
	slidy.js
presentation.html	

Table 4.1: An example starting point for migration with HTML presentation file and both the Slidy JavaScript and a generic CSS file for style.

css/	
	style.css
	rslidy.css
scripts/	
	slidy.js
presentation.html	

Table 4.2: The presentation folder with the newly included rSlidy CSS file.

- Open the original presentation's HTML file (presentation.html).
- Replace the existing inclusion of the Slidy or kSlidy JavaScript file with the inclusion of rSlidy's JavaScript file (rslidy.js).
- Include Hammer.js (hammer.js).
- Set optional meta-tags: Enabling animated transitions is recommended for a better viewing experience.

On a sidenote, more new meta-tags and their effects can be found in chapter 5.

4.3 Step by Step Guide with Code Examples

Assuming an original presentation folder structure similar to the one in table 4.1 the migration can be achieved as follows.

Place the rSlidy CSS file in the existing presentations CSS folder (rslidy.css)

The presentation folder should now look like in table 4.2

css/	
	style.css
	rslidy.css
scripts/	
	rslidy.js
	hammer.js
presentation.html	

Table 4.3: The presentation folder with the newly included JavaScript files. Note that the original slidy.js file has been removed since it is not needed anymore.

Place both rSlidy's JavaScript file and Hammer's JavaScript file in the existing presentations SCRIPTS folder (rslidy.js, hammer.js)

Since the original Slidy JavaScript file is not going be used anymore once the JavaScript variables in the presentation's HTML file are replaced, it can be deleted. After these first steps the folder structure of the presentation should be similar to table 4.3.

Open the original presentation's HTML file (presentation.html)

The original presenation's HTML file will probably look something like this:

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
3
4  <head>
5
6  <title>Example Presentation</title>
7
8  <link rel="stylesheet" href="css/style.css"/>
9
10 <script src="scripts/slidy.js" type="text/javascript"></script>
11
12 </head>
13
14 <body>
15 <div class="slide">
16 <h1>Example title</h1>
17 </div>
18
19  ...
```

Replace the existing inclusion of the Slidy or kSlidy JavaScript file with the inclusion of rSlidy's JavaScript file (rslidy.js)

Line ten in the code-snipped above shows the inclusion of the original Slidy JavaScript file. This include needs to be replaced with the new rSlidy JavaScript file resulting in a line similar to this one:

```
1  <script src="scripts/rslidy.js" type="text/javascript"></script>
```

Include Hammer.js (hammer.js)

Afterwards the only external dependcy, Hammer, needs to be included. This can be achieved with another JavaScript inclusion:

```
1  <script src="scripts/hammer.min.js" type="text/javascript"></script>
```

Set optional meta-tags

Finally some optional meta-tags can be set at will.

Enabling animated transitions is recommended for a better viewing experience. The meta-tag for this is called "rslidy_animated_transition" and needs to be set to true:

```
<meta name="rslidy_animated_transition" content="true"/>
```

Final Result

The resulting presentation's HTML file should look similar to this:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>

<title>Example Presentation</title>

<link rel="stylesheet" href="css/style.css"/>

<script src="scripts/rslidy.js" type="text/javascript"></script>
<script src="scripts/hammer.js" type="text/javascript"></script>

<meta name="rslidy_animated_transition" content="true"/>

</head>

<body>
<div class="slide">
<h1>Example title</h1>
</div>

...
```

Following these simple steps the migration from an old Slidy or kSlidy presentation has been successfully accomplished and the presentation can now be used on mobile devices or remote controlled.

Chapter 5

Technical Details

5.1 Base Technology

A core idea when developing rSlidy was to retain a maximum level of compatibility with other versions meaning the same technologies and languages were used. The attempt to use modern and up to date mechanics required the use of up to date versions of these technologies. Hence rSlidy currently has compatibility issues with old browsers. The base technologies used are:

- HTML 5
- JavaScript
- CSS 3

5.1.1 External JavaScript Libraries

One big goal during development was to remove all possible dependencies from rSlidy. The limited time-frame provided meant a compromise had to be accepted. In the current version the library Hammer.js [1] is being used. Hammer.js provides easy to use poly-fill handles for touch-interaction. Since a global standard is still on its way implementing a good functionality across Android, iOS, Windows Phone, etc would have meant to create and process the events accordingly to all current systems. This would have meant that different ideas would be sacrificed. Hence the decision was made to use external libraries here until a built in support can be provided or a global standard for touch inputs gets introduced.

5.2 Slide Generation

As already mentioned rSlidy takes normal HTML files as input. These files then get parsed when the entire code has been loaded. Just as older versions the system will look for -Tags with the class *slide*. Unlike older versions the layer is allowed to be member of multiple other classes too. The JavaScript functionality classList.contains() allows easy checks whether a node is member of a specific CSS-class or not.

When a slide has been found the entire node including its children gets cloned. These clones then get attached to the base frame rSlidy uses and event handlers get attached if that is needed. Afterwards the incremental-objects get parsed. Once again a class-attribute gets used. rSlidy will look for elements being member of the class *incremental*. Each slide contains its own set of increment objects allowing different progress states for each slide. Once the slide has been safely attached the old original node gets deleted. This has been done so that the old slides do not interfere with User-JavaScript or mess around with the layouts.

[1] http://eightmedia.github.io/hammer.js/

Inside the rSlidy frame each found slide gets its own layout-sandbox. This is just a simple div which is managed by the system allowing the user to play around with margins or various other CSS commands without interrupting functionality.

5.3 Table of Contents

Since rSlidy should be able to be used for presenting slides it was an important decision to actually show what is roughly on the slides when displaying a table of contents. A very simple mechanic relying on CSS 3 gets used to create the fully rendered preview.

During the slide generation (see 5.2) process the constructed clone not only gets attached to the rSlidy base frame but also to a sandbox frame belonging to the table of contents. Here all the normally hidden incremental steps are always visible. To create a scaled version of the original slide the CSS commands *transform:scale(0.3,0.3);* and *transform-origin: 0 0;* were used. Finally these new frames should react to click commands by jumping the presentation to the matching slide but should not allow any other events. This has been done by attaching the jump command to the sandbox frame and then using the CSS-command *pointer-events:none;* on its content. Now no mouse events get triggered on the preview slide and only the jump gets captured. This is an easy alternative to stopping propagation and cancelling bubbles.

5.4 More HTML Parsing

During the creation process a few other things need to be done. These allow passing some configuration options to the user without forcing JavaScript modifications, URL navigation, setting custom key-binds and more.

5.4.1 Parsing Meta Tags

<meta>-tags are used in the same way command-line parameters are used for other programs. During the initialisation the system will read all meta-tags found. If the name-attribute matches one of the implemented commands the contents of the content-attribute will be read and the appropriate functionality will be set. Currently all implemented commands come with a *rslidy_* prefix. This has been introduced to avoid conflicts with other systems that might be used as well. Which meta-commands are currently implemented and what they do can be found in the meta-commands section A.1 of the quick tutorial.

5.4.2 Platform Detection

Users behave differently on mobile devices and PC platforms. In addition PCs usually come with a lot more power allowing more eye-candy which should be turned off on mobiles. This is why rSlidy tries to detect which platform is currently used. This is done by reading the *navigator.userAgent* element. If a mobile browser is detected a simple variable will be set to true so all the other rSlidy functions can use it. As it was our goal to not force the user into anything this differentiation can be overwritten with a specific meta-command.

5.4.3 URL Navigation

When passing links or sometimes even whilst presenting it is important to jump to a very specific point on the slides. This can be done by adding something to the URL. The jump syntax is slide.html#[*Slide Number*,*Step Number*]. Upon loading this allows to jump to the specified slide and display the correct amount of incremental-steps. rSlidy does this by simply parsing the URL with a regular expression extracting the two values and jumps there once the slides have been constructed.

5.5 Touch Handling

As previously mentioned rSlidy currently uses Hammer.js to capture the touch input. The system will only react to tap and swipe commands and then passing the captured event to the controller. Past this point there is no difference whether the input came from a mouse or a finger. The functions to change slides and steps are globally available within rSlidy so all that needs to be done is to extract the type of event and where it occurred.

5.5.1 Tap

Tapping (or clicking) is generally used to proceed by a single step. To allow going a step forward and backward the screen has been split into two halves. A registered tap on the left side will move a step back hiding the last incremental element whilst a tap on the right side of the screen will show the next one. When all incremental objects are already displayed then the next step causes rSlidy to load the next slide. The same way a left-tap will load the previous slide if none of the incremental objects are displayed.

5.5.2 Swipe

Swiping (or holding the mouse and moving a certain distance) allows to move to the next or previous slide entirely ignoring the incremental step status. This allows faster navigation on mobile devices. When changing slides the previous incremental-status gets saved so when the user returns to this slide it will be in exactly the same state he left it. Swiping from left to right loads the previous slide whilst a right-to-left movement loads the next one. This was the most intuitive way as it looks like the user is pushing the pane to the side revealing the next slide.

5.6 Websocket Remote Control

Figure 5.1: The Remote Control popup after a successful server connection.

rSlidy allows remotely controlling other presentations via the WebSocket protocol[2]. This requires a Server to be up and running. The source-code for a very simple server can be found in the rSlidy package on github. At the moment this is not much more that a proof of concept! Everytime rSlidy loads it will attept to connect

[2] http://tools.ietf.org/html/rfc6455

to the server. If a connection could be established a popup will open showing different alternatives as seen in 5.1. The three options are listed below.

Master When connecting as master all slide and step changes are sent to the server and then being broadcast to all clients.

Slave A slave client will listen to broadcasts and execute the incoming commands to change slide or move a step forward or backward.

Autonomous Selecting this option closes the connection to the server and allows normal use. This is also the default mode if no connection could be established.

5.7 Future Versions

Some of the design decisions were not taken by the rSlidy development team. To ensure compatibility with other Slidy versions certain modules do no work in the optimal way.

5.7.1 class-Handles

Classes are supposed to be selectors for CSS. The old Slidy uses them to detect slide-divs and incremental objects. This obviously is not the ideal way. The solution deemed ideal is to use the HTML 5 dataset attribute[3]. This would remove the dual-usage of the class attribute and would also provide a lot more options to the user. A few possibilities are implementing various slide transitions or using different elements as slide-objects.

5.7.2 <meta>-Tags

Meta-Tags actually only have a very limited set of legal name-values within the default namespace. This causes the HTML-validation to fail if various commands have been set. Whilst this is not a big issue in general it is something that is not pretty and should be gotten rid of. As before the reason rSlidy uses this dirty method of doing things because of previous versions. Once again the dataset could be used maybe use entire set up custom tags[4] but this is currently just a draft and no browser supports it at the moment.

5.7.3 Remote Control

The WebSocket feature has a lot of potential but would also require a lot of work. Things that would need to be implemented is to stop input handling on clients, providing different views for master and slave. Hide things like the status bar or the table of contents for presentation slaves. All these options could be implemented via the specific Meta-tags or with the custom setting-tag mentioned above. Another important task is to write a prober server which does not use multithreaded blocking calls.

[3] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement.dataset
[4] http://w3c.github.io/webcomponents/spec/custom/

Appendix A

rSlidy Quick Manual

A.1 Installation

To use rSlidy you have to, of course, download it first. It would be best to extract it move the files and folders to the same folder your presentation is in. Once placed in your folder all you need to do to get the basic functionality is to reference the JavaScript files in your presentation. Add the following two lines of code to your HTML file inside the <head>-Tag

```
<script src="scripts/hammer.min.js" type="text/javascript"></script>
```

```
<script src="scripts/rslidy.js" type="text/javascript"></script>
```

Now that rSlidy is set up you can add slides to your presentation.

A.2 Adding Slides

To create a rSlidy-slide add the following line to your code.

```
<div class="slide"></div>
```

Everything within this tag will be treated as a slide. You can also transform existing -elements to a slide by adding "slide" to its class list. The content of this layer will be put in a sandbox-layer so your existing CSS styling will remain intact.

NOTE: Elements with position:fixed can escape this sandbox and can negatively impact the look and feel!

A.3 Defining Incremental Elements

rSlidy allows the user to define various elements on a slide which will then appear one after another when you give the command to move a step forward. This can be done by simply adding "incremental" to the elements class list. Almost all HTML elements support this.
-Tags will not work with incremental.

When you have defined your incremental elements you can open your presentation inside a browser. If rSlidy was able to extract the slides from your HTML file you should now see your presentation.

A.4 Controlling your Presentation

A.4.1 Mobile

On a mobile device the primary ways to interact with the presentation are tapping and swiping. Whilst tapping moves a step swiping will move an entire slide. If you tap on the left side of the screen rSlidy will go a step back and if you tap on the right side you will proceed by one step. A swiping motion will push the slide to the side and reveal the next (or previous) one. Tapping on the menu icon in the bottom left corner will open the table of contents allowing you to quickly jump to a specific slide by tapping on its preview.

NOTE: Any other gestures used on mobile devices like pinching to zoom will retain their normal functionality!

A.4.2 PC

When using a PC to view a rSlidy presentation multiple options to navigate through the presentation are available. When using the mouse the interaction works the same way as on mobile platforms. Additionally hotkeys have been introduced. The default keys are:

- **Arrow Up** Step back
- **Arrow Down** Step forward
- **Arrow left** Slide back
- **Arrow right** Slide forward
- **Esc** Open/Close table of content
- **Ctrl** Lock/Unlock screen

Locking the screen disables all possibility to change the current slide or the current step. This has been introduced to allow users to mark text without rSlidy interpreting their movement as gesture.

NOTE: All keys can be customized!

A.5 Customising your Presentation

A.5.1 CSS

rSlidy will not interfere with any CSS definitions you have defined in your HTML-file or any referenced CSS.

A.5.2 Meta-Tags

In order to customise the functionality of rSlidy without editing its source code you have to add specific <meta>-tags to your HTML. These commands allow a wide variety of manipulations. The command syntax:

```
1 <meta name="{NAME}" content="{VALUE}" />
```

The table A.1 shows the different commands and their effect.

Command	Values	Effect
rslidy_background_color	CSS colour value	Set the global background colour
rslidy_background_image	CSS image path	Sets a static background image. This image will NOT move with any slide and will always stay at the back. **NOTE:** This only works if your slides do not have a 100% filling background.
rslidy_border_color	CSS colour value	Defines the colour of the border around slides. Leave blank to remove borders
rslidy_css	Path	Additional way to reference a CSS file. Has no real use right now.
rslidy_animated_transition	true/false	Enable/Disable CSS transitions when slides change. Default: false
rslidy_title	String	Sets the presentation title. Default: content of <title>-tag
rslidy_keycodes	Keys in JSON	Sets custom key bindings. See below A.5.3 for details
rslidy_touch	true/false	Overrides the detected platform(PC/mobile). Default: ignored.
rslidy_statusbar	true/false	Show/Hide the status bar. Default: true

Table A.1: The meta-tag commands commands currently implemented in rSlidy

A.5.3 Custom Keybinds

Custom keys are defined as a simple JSON objects. The syntax is:

```
1 {"up":[38,87],"down":[40,83],"left":[37,65],"right":[39,68],"lock":[0,17],"nav
    ":[0,27]}
```

Every action can be bound to multiple keys. The example shows the default key bindings in addition WASD mirroring the arrow keys. Every entry must be an array even if there is only a single key to be defined. Keys are defined via their JavaScript key-codes. A full list can be found here http://cambiaresearch.com/articles/15/javascript-char-codes-key-codes. When a key is set the default will no longer be active. This means that the default keys can be unbound.

NOTE: JavaScript key codes differ from their normal ASCII value!

A.6 Improving and Modifying rSlidy

rSlidy is freely available to everyone on GitHub here[1].

[1] https://github.com/pkasper/rSlidy