# Hovercraft! Documentation

*Release 1.0*

**Lennart Regebro**

**May 24, 2017**

# Contents

Contents:

CHAPTER 1

# Introduction

## GUI tools are limiting

I used to do presentations with typical slideshow software, such as OpenOffice/LibreOffice Impress, but these tools felt restricted and limiting. I need to do a lot of reorganizing and moving around, and that might mean changing things from bullet lists to headings to text to pictures and back to bullet lists over again. This happens through the whole process. I might realize something that was just a bullet point needs to be a slide, or that a set of slides for time reasons need to be shortened down to bullet points. Much of the reorganization comes from seeing what fits on one slide and what does not, and how I need to pace the presentation, and to some extent even what kinda of pictures I can find to illustrate what I try to say, and if the pictures are funny or not.

**Presentation software should give you complete freedom to reorganize your presentation on every level, not only by reorganizing slides.**

The solution for me and many others, is to use a text-markup language, like reStructuredText, Markdown or similar, and then use a tool that generates an HTML slide show from that.

**Text-markup gives you the convenience and freedom to quickly move parts around as you like.**

I chose reStructuredText, because I know it and because it has a massive feature set. When I read the documentations of other text-markup langages it was not obvious if they has the features I needed or not.

## Pan, rotate and zoom

The tools that exist to make presentations from text-markup will make slideshows that has a sequence of slides from left to right. But the fashion now is to have presentations that rotate and zoom in and out. One open source solution for that is impress.js.

**With impress.js you can make modern cool presentations.**

But impress.js requires you to write your presentation as HTML, which is annoying, and the markup isn't flexible enough to let you quickly reorganize things from bullet points to headings etc.

You also have to position each slide separately, and if you insert a new slide in the middle, you have to reposition all the slides that follow.

## Hovercraft!

So what I want is a tool that takes the power, flexibility and convenience of reStructuredText and allows me to generate pan, rotate and zoom presentations with impress.js, without having to manually reposition each slide if I reorganize a little bit of the presentation. I couldn't find one, so I made Hovercraft.

Hovercraft's power comes from the combination of reStructuredText's convenience with the cool of impress.js, together with a flexible and powerful solution to position the slides.

There are four ways to position slides:

1. Absolute positioning: You simply add X and Y coordinates to a slide, in pixels. Doing only this will not be fun, but someone might need it.
2. Relative positioning to last slide: By specifying x and/or y with with a starting r,you specify the distance from the previous slide. By using this form of positioning you can insert a slide, and the other slides will just move to make space for the new slide.
3. Relative positiong to any slide: You can reference any *previous* slide by its id and specify the position relative to it. This will work for all positioning fields. However, you should not use `r` as a slide id since the positioning might not behave as you expect.
4. Automatically: If you don't specify any position the slide will have the same settings as the previous slide. With a relative positioning, this means the slide will move as long as the previous slide moved. This defaults to moving 1600px to the right, which means that if you supply no positions at all anywhere in the presentation, you get the standard slide-to-the-left presentation.
5. With an SVG path: In this last way of positioning, you can take an SVG path from an SVG document and stick it into the presentation, and that slide + all slides following that has no explicit positioning will be positioned on that path. This can be a bit fiddly to use, but can create awesome results, such as positioning the slides as snaking Python or similar.

Hovercraft! also includes impress-console, a presenter console that will show you your notes, slide previews and the time, essential tools for any presentation.

CHAPTER 2

# Using Hovercraft!

You can either use Hovercraft! to generate the presentation as HTML in a target directory, or you can let Hovercraft! serve the presentation from its builtin webserver.

The latter have several benefits. One is that most webbrowsers will be very reluctant to open popup-windows from pages served from the file system. This is a security measure which can be changed, but it's easier to just point the browser to http://localhost:8000 instead.

The second benefit is that Hovercraft! will monitor the source files for the presentation, and if they are modified Hovercraft! will generate the presentation again automatically. That way you don't have to run Hovercraft! everytime you save a file, you only need to refresh the browser.

## Parameters

```
hovercraft [-h] [-t TEMPLATE] [-c CSS] [-j JS] [-a] [-s] [-n] [-p PORT]
<presentation> [<targetdir>]
```

Positional arguments:

**<presentation>** The path to the reStructuredText presentation file.

**<targetdir>** The directory where the presentation is saved. Will be created if it does not exist. If you do not specify a targetdir Hovercraft! will instead start a webserver and serve the presentation from that server.

Optional arguments:

**-h, --help** Show this help.

**-t TEMPLATE, --template TEMPLATE** Specify a template. Must be a .cfg file, or a directory with a `template.cfg` file. If not given it will use a default template.

**-c CSS, --css CSS** An additional CSS file for the presentation to use. See also the `:css:` settings of the presentation.

**-j JS, --js JS** An additional Javascript file for the presentation to use. Added as a js-body script. See also the `:js-body:` settings of the presentation.

**-a, --auto-console** Open the presenter console automatically. This is useful when you are rehearsing and making sure the presenter notes are correct. You can also set this by having `:auto-console: true` first in the presentation.

**-s, --skip-help** **Do not show the initial help popup. You can also set** this by having `:skip-help: true` first in the presentation.

**-n, --skip-notes** Do not include presenter notes in the output.

**-N, --slide-numbers** Show the current slide number on the slide itself and in the presenter console. You can also set this by having `slide-numbers: true` in the presentation preamble.

**-p PORT, --port PORT** The address and port that the server uses. Ex 8080 or 127.0.0.1:9000. Defaults to 0.0.0.0:8000.

## Built in templates

There are two templates that come with Hovercraft! One is called `default` and will be used unless you specify a template. This is the template you will use most of the time.

The second is called `simple` and it doesn't have a presenter console. This template is especially useful if you combine it with the `--skip-notes` parameter to prepare a version of your presentation to be put online.

CHAPTER 3

# Making presentations

## A note on terminology

Traditionally a presentation is made up of slides. Calling them "slides" is not really relevant in an impress.js context, as they can overlap and doesn't necessarily slide. The name "steps" is better, but it's also more ambiguous. Hence impress.js uses the terms "slide" and "step" as meaning the same thing, and so does Hovercraft!

## Hovercraft! syntax

Presentations are reStructuredText files. If you are reading this documentation from the source code, then you are looking at a reStructuredText document already.

It's fairly simple, you underline headings to mark them as headings:

```
This becomes a h1
=================

And this a h2
-------------
```

The different ways of underlining them doesn't mean anything, instead the order of them is relevant, so the first type of underline encountered in the file will make a level 1 heading, the second type a level 2 heading and so on. In this file = is used for level 1, and - for level 2.

You can also mark text as *italic* or **bold**, with `*single asterixes*` or `**double asterixes**` respectively.

You can also have bullet lists:

```
* Bullet 1

  * Bullet 1.1

* Bullet 2
```

```
* Bullet 3
```

And numbered lists:

```
1. Item 1

    1.1. Item 1.1

2. Item 2

3. Item 3
```

You can include images:

```
.. image:: path/to/image.png
    :height: 600px
    :width: 800px
```

As you see you can also specify height and width and loads of other parameters, but they are all optional.

And you can mark text as being preformatted. You do that by ending the previous row with double colons, or have a row of double colons by itself:

```
::

    This code here will be preformatted
     and shown with a  monospaced font
    and    all    spaces     preserved.
```

If you want to add source code, you can use the `code` directive, and get syntax highlighting:

```
.. code:: python

    def some_example_code(foo):
        return foo * foo
```

The syntax highlighting is done by Pygments and supports lots and lots of languages.

You are also likely to want to put a title on the presentation. You do that by having a `.. title::` statement before the first slide:

```
.. title:: This is the presentation title
```

That is the most important things you'll need to know about reStructuredText for making presentations. There is a lot more to know, and a lot of advanced features like links, footnotes, and more. It is in fact advanced enough so you can write a whole book in it, but for all that you need to read the documentation.

If you add a `math` directive then hovercraft! will add a link to the MathJax CDN so that this:

```
.. math:: e^{i \pi} + 1 = 0
```

will be rendered by the MathJax javascript library. The math directive can also be used as a "role" with the equations inlined with the text flow. Note that if you use the math statement, by default the MathJax library will be loaded from the internet, meaning that your presentation will need network connectivity to work, which can be a problem when presenting and conferences, which often have bad network connectivity.

This can be solved by specifying a local copy of mathjax with the –mathjax command line.

## External files

Any image file referenced in the presentation by a relative path will be copied to the target directory, keeping it's relative path to the presentation. The same goes for images or fonts referenced in any CSS files used by the presentation or the template.

Images or fonts referenced by absolute paths or URI's will not be copied.

## Styling your Presentation

The css that is included by the default template are three files.

- `impressConsole.css` contains the CSS needed for the presenter console to work,
- `highlight.css` contains a default style for code syntax highlighting, as that otherwise would be a lot of work. If you don't like the default colors or styles in the highlighting, this is the file you should copy and modify.
- `hovercraft.css`, which only includes the bare minimum: It hides the impress.js fallback message, the presenter notes, and sets up a useful default of having a step width be 1000 pixels wide.

For this reason you want to include your own CSS to style your slides. To include a CSS file you add a `:css:`-field at the top of the presentation:

```
:css: css/presentation.css
```

You can also optionally specify that the css should be only valid for certain CSS media:

```
:css-screen,projection: css/presentation.css
:css-print: css/print.css
```

You can specify any number of css files in this way. You can also add one extra CSS-file via a command-line parameter:

> hovercraft –css=my_extra.css presentationfile.rst outdir/

In a similar fashion you can add Javascript files to either header or body:

```
:js-header: js/firstjsfile.js
:js-body: js/secondjsfile.js
```

You can also add one extra Javascript-file via a command-line parameter:

> hovercraft –js=my_extra.js presentationfile.rst outdir/

If you want static content, content that doesn't move with each slide; for example a header, footer, your company logo or a slide background pattern, then you can insert that content with the header and footer commands:

```
.. header::

   .. image:: images/company-logo.png


.. footer::

    "How to use Hovercraft", Yern Busfern, ImaginaryCon 2017
```

The header will be located in the resulting HTML before the first slide and the footer will be located after the last slide. However, they will be displayed statically on every slide, and you will have to position them with CSS. By default the

header will be displayed behind the slides and the footer in front of the slides, so the header is useful for background designs and the footer for designs that should be in the foreground.

It doesn't matter where in the presentation you add these commands, I would recommend that you add them before the first slide.

### Styling a specific slide

If you want to have specific styling for a specific slide, it is a good idea to give that slide a unique ID:

```
:id: the-slide-id
```

You can then style that slide specifically with:

```
div#the-slide-id {
    /* Custom CSS here */
}
```

If you don't give it a specific ID, it will get an ID based on its sequence number. And that means the slide's ID will change if you insert or remove slides that came before it, and in that case your custom stylings of that slide will stop working.

### Portable presentations

Since Hovercraft! generates HTML5 presentations, you can use any computer that has a modern browser installed to view or show the presentation. This allows you both to put up the presentation online and to use a borrowed computer for your conference or customer presentation.

When you travel you don't know what equipment you have to use when you show your presentaton, and it's surprisingly common to encounter a projector that refuses to talk to your computer. It is also very easy to forget your dongle if you have a MacBook, and there have even been cases of computers going completely black and dead when you connect them to a projector, even though all other computers seem to work fine.

The main way of making sure your presentation is portable is to try it on different browsers and different computers. But the latter can be unfeasible, not everyone has both Windows, Linux and OS X computers at home. To help make your presentations portable it is a good idea to define your own @font-face's and use them, so you are sure that the target browser will use the same fonts as you do. Hovercraft! will automatically find @font-face definitions and copy the font files to the target directory.

## impress.js fields

The documentation on impress.js is contained as comments in the demo html file. It is not always very clear, so here comes a short summary for convenience.

The different data fields that impress.js will use in 0.5.3, which is the current version, are the following:

- **data-transition-duration**: The time it will take to move from one slide to another. Defaults to 1000 (1 second). This is only valid on the presentation as a whole.
- **data-perspective**: Controls the "perspective" in the 3d effects. It defaults to 500. Setting it to 0 disables 3D effects.
- **data-x**: The horizontal position of a slide in pixels. Can be negative.
- **data-y**: The vertical position of a slide in pixels. Can be negative.

- **data-scale**: Sets the scale of a slide, which is what creates the zoom. Defaults to 1. A value of 4 means the slide is four times larger. In short: Lower means zooming in, higher means zooming out.
- **data-rotate-z**: The rotation of a slide in the x-axis, in degrees. This will cause the slide to be rotated clockwise or counter-clockwise.
- **data-rotate**: The same as **data-rotate-z**.
- **data-rotate-x**: The rotation of a slide in the x-axis, in degrees. This means you are moving the slide in a third dimension compared with other slides. This is generally cool effect, if used right.
- **data-rotate-y**: The rotation of a slide in the x-axis, in degrees.
- **data-z**: This controls the position of the slide on the z-axis. Setting this value to -3000 means it's positioned -3000 pixels away. This is only useful when you use **data-rotate-x** or **data-rotate-y**, otherwise it will only give the impression that the slide is made smaller, which isn't really useful.

## Hovercraft! specialities

Hovercraft! has some specific ways it uses reStructuredText. First of all, the reStructuredText "transition" is used to mark the separation between different slides or steps. A transition is simply a line with four or more dashes:

```
----
```

You don't have to use dashes, you can use any of the characters used to underline headings, `= - ` : . ' " ~ ^ _ * + #`. And just as width headings, using different characters indicates different "levels". In this way you can make a hierarchical presentation, with steps and substeps. However, impress.js does not support that, so this is only useful if you make your own templates that uses another Javascript library, for example Reveal.js. If you have more than one transition level with the templates included with Hovercraft, the resulting presentation may behave strangely.

All reStructuredText fields are converted into attributes on the current tag. Most of these will typically be ignored by the rendering to HTML, but there is two places where the tags will make a difference, and that is by putting them first in the document, or first on a slide.

Any fields you put first in a document will be rendered into attributes on the main impress.js `<div>`. The only ones that Hovercraft! will use are `data-transition-duration`, `skip-help`, `auto-console` and `slide-numbers`.

Any fields you put first in a slide will be rendered into attributes on the slide `<div>`. This is used primarily to set the position/zoom/rotation of the slide, either with the `data-x`, `data-y` and other impress.js settings, or the `hovercraft-path` setting, more on that later.

Hovercraft! will start making the first slide when it first encounters either a transition or a header. Everything that comes before that will belong to the presentation as a whole.

A presentation can therefore look something like this:

```
:data-transition-duration: 2000
:skip-help: true

.. title: Presentation Title

----

This is the first slide
=======================

Here comes some text.
```

```
----

:data-x: 300
:data-y: 2000

This is the second slide
========================

#. Here we have

#. A numbered list

#. It will get correct

#. Numbers automatically
```

## Relative positioning

Hovercraft! gives you the ability to position slides relative to each other. You do this by starting the coordinates with "r". This will position the slide 500 pixels to the right and a thousand pixels above the previous slide:

```
:data-x: r500
:data-y: r-1000
```

Relative paths allow you to insert and remove slides and have other slides adjust automatically. It's generally the most useful way of positioning.

## Automatic positioning

If you don't specify an attribute, the slide settings will be the same as the previous slide. This means that if you used relative positioning, the next slide will move the same distance.

This gives a linear movement, and your slides will end up in a straight line.

By default the movement is 1600 pixels to the right, which means that if you don't position any slides at all, you get a standard presentation where the slides will simply slide from right to left.

## SVG Paths

Hovercraft! supports positioning slides along an SVG path. This is handy, as you can create a drawing in a software that supports SVG, and then copy-paste that drawings path into your presentation.

You specify the SVG path with the `:hovercraft-path:` field. For example:

```
:hovercraft-path: m275,175 v-150 a150,150 0 0,0 -150,150 z
```

Every following slide that does not have any explicit positioning will be placed on this path.

There are some things you need to be careful about when using SVG paths.

### Relative and absolute coordinates

SVG coordinates can either be absolute, with a reference to the page origin; or relative, which is in reference to the last point. Hovercraft! can handle both, but what it can not handle very well is a mixture of them.

Specifically, if you take an SVG path that starts with a relative movement and extract that from the SVG document, you will lose the context. All coordinates later must then also be relative. If you have an absolute coordinate you then suddenly regain the context, and everything after the first absolute coordinate will be misplaced compared to the points that come before.

Most notable, the open source software "Inkscape" will mix absolute and relative coordinates, if you allow it to use relative coordinates. You therefore need to go into it's settings and uncheck the checkbox that allows you to use relative coordinates. This forces Inkscape to save all coordinates as absolute, which will work fine.

### Start position

By default the start position of the path, and hence the start position of the first slide, will be whatever the start position would have been if the slide had no positioning at all. If you want to change this position then just include `:data-x:` or `:data-y:` fields. Both relative and absolute positioning will work here.

In all cases, the first `m` or `M` command of the SVG path is effectively ignored, but you have to include it anyway.

### SVG transforms

SVG allows you to draw up path and then transform it. Hovercraft! has no support for these transforms, so before you extract the path you should make sure the SVG software doesn't use transforms. In Inkscape you can do this by the "Simplify" command.

### Other SVG shapes

Hovercraft! doesn't support other SVG shapes, just the path. This is because organising slides in squares, etc, is quite simple anyway, and the shapes can be made into paths. Usually in the software you will have to select the shape and tell your software to make it into a path. In Inkscape, transforming an object into a path will generally mean that the whole path is made of CubicBezier curves, which are unnecessarily complex. Using the "Simplify" command in Inkscape is usually enough to make the shapes into paths.

### Shape-scaling

Hovercraft! will scale the path so that all the slides that need to fit into the path will fit into the path. If you therefore have several paths in your presentation, they will **not** keep their relative sizes, but will be resized so the slides fit. If you need to have the shapes keep their relative sizes, you need to combine them into one path.

## Examples

To see how to use Hovercraft! in practice, there are three example presentations included with Hovercraft!

**hovercraft.rst** The demo presentation you can see at http://regebro.github.com/hovercraft

**tutorial.rst** A step by step guide to the features of Hovercraft!

**positions.rst** An explanation of how to use the positioning features.

CHAPTER 4

# Designing your presentations

There are several tricks to making presentations. I certainly do not claim to be an expert, but here are some beginners hints.

## Take it easy

Don't go too heavy on the zoom. Having a difference between two slides in scale of more than 5 is rarely going to look good. It would make for a nice cool zooming effect if it did, but this is not what browsers are designed for, so it won't.

And the 3D effects can be really cool if used well. But not all the time, it gets tiring for the audience.

Try, if you can, to use the zoom and 3D effects when they make sense in the presentation. You can for example mention the main topics on one slide, and then zoom in on each topic when you discuss it in more detail. That way the effects help clarify the presentation, rather than distract from it.

## Custom fonts

Browsers tend to render things subtly differently.

They also have different default fonts, and different operating systems have different implementations of the same fonts. So to make sure you have as much control over the design as possible, you should always include fonts with the presentation. A good source for free fonts are Google Webfonts. Those fonts are free and open source, so you can use them with no cost and no risk of being sued. They can also be downloaded or included online.

### Online vs Downloaded

If you are making a presentation that is going to run on your computer at a conference or customer meeting, always download the fonts and have them as a part of the presentation. Put them in a folder called `fonts` under the folder where your presentation is.

You also need to define the font-family in your CSS. Font Squirrel's webfont generator will provide you with a platform-independent toolkit for generating both the varius font formats and the CSS.

If the presentation is online only, you can put an `@include`-statement in your CSS to include Googles webfonts directly:

```
@import url(http://fonts.googleapis.com/css?
↪family=Libre+Baskerville|Racing+Sans+One|Satisfy);
```

But don't use this for things you need to show on your computer, as it requires you to have internet access.

## Test with different browsers

If you are putting the presentation online, test it with a couple of major browsers, to make sure nothing breaks and that everything still looks good. Not only are there subtle differences in how things may get positioned, different browsers are also good at different things.

I've tested some browsers, all on Ubuntu, and it is likely that they behave differently on other operating systems, so you have to try for yourself.

### Firefox

Firefox 18 is quite slow to use with impress.js, especially for 3D stuff, so it can have very jerky movements from slide to slide. It does keep text looking good no matter how much you zoom. On the other hand, it refuses to scale text infinitely, so if you scale too much characters will not grow larger, they will instead start moving around.

Firefox 19 is better, but for 3D stuff it's still a bit slow.

### Chrome

Chrome 24 is fast, but will not redraw text in different sizes, but will instead create an image of them and rescale them, resulting in the previous slide having a fuzzy pixelated effect.

### Epiphany

Epiphany 3.4.1 is comparable to Firefox 19, possibly a bit smoother, and the text looks good. But it has bugs in how it handles 3D data, and the location bar is visible in fullscreen mode, making it less suitable for any sort of presentation.

CHAPTER 5

# Templates

Luckily, for most cases you don't need to create your own template, as the default template is very simple and most things you need to do is doable with css. However, I don't want Hovercraft! to set up a wall where it isn't flexible enough for your needs, so I added support to make your own templates.

You need to create your own template if you are unsatisfied with the HTML that Hovercraft! generates, for example if you need to use another version of HTML or if the reStructuredText you are using isn't being rendered in a way that is useful for you. Although if you aren't happy with the HTML generated from the reStructuredText that could very well be a bug, so open an issue on Github for discussion.

Hovercraft! generates presentations by converting the reStructuredText into XML and then using XSLT to translate the XML into HTML.

Templates are directories with a configuration file, a template XSL file, and any number of CSS, JS and other resource files.

## The template configuration file

The configuration file is normally called template.cfg, but if you have several configuration files in one template directory, you can specify which one to use by specifying the full path to the configuration file. However, if you just specify the template directory, `template.cfg` will be used.

Template files are in configparser format, which is an extended ini-style format. They are very simple, and have only one section, `[hovercraft]`. Any other sections will be ignored. Many of the parameters are lists that often do not fit on one line. In that case you can split the line up over several lines, but indenting the lines. The amount of indentation doesn't make any difference, except aesthetically.

The parameters in the `[hovercraft]` section are:

- `template` The name of the xsl template.
- `css` A list of CSS filenames separated by whitespace. These files will get included in the final file with "all" as the media specification.

- `css-<media>` A list of CSS filenames separated by whitespace. These files will get included in the final file with the media given in the parameter. So the files listed for the parameter "css-print" will get "print" as their media specification and a key like "css-screen,print" will return media "screen,print".
- `js-header` A list of filenames separated by whitespace. These files will get included in the target file as header script links.
- `js-body` A list of filenames separated by whitespace. These files will get included in the target file as script links at the end of the file. The files impress.js, impressConsole.js and hovercraft.js typically need to be included here.
- `resources` A list of filenames separated by whitespace that will be copied to the target directory, but nothing else is done with them. Images and fonts used by CSS will be copied anyway, but other resources may be added here.
- `resource-directories` A list of directory names separated by whitespace. These will be treated like `resources` above, ie only copied to the target directory. The directory contents will be copied recursively, but hidden files (like files starting with a `.` are ignored.

An example:

```
[hovercraft]
template = template.xsl

css = css/screen.css
      css/impressConsole.css

css-print = css/print.css

js-header = js/dateinput.js

js-body = js/impress.js
          js/impressConsole.js
          js/hovercraft.js

resources = images/back.png
            images/forward.png
            images/up.png
            images/down.png
```

## The template file

The file specified with the `template` parameters is the actual XSLT template that will perform the translation from XML to HTML.

Most of the time you can just copy the default template file in `hovercraft/templates/default/template.xsl` and modify it. XSLT is very complex, but modifying the templates HTML is quite straightforward as long as you don't have to touch any of the `<xsl:...>` tags.

Also, the HTML that is generated is XHTML compatible and quite straightforward, so for the most case all you would need to generate another version of HTML, for example strict XHTML, would be to change the doctype.

But if you need to add or change the main generated HTML you can add and change HTML statements in this main file as you like. See for example how the little help-popup is added to the bottom of the HTML.

If you want to change the way the reStructuredText is rendered things get slightly more complex. The XSLT rules that convert the reStructuredText XML into HTML are contained in a separate file, `reST.xsl`. For the most part you can just include it in the template file with the following code:

```
<xsl:import href="resource:templates/reST.xsl" />
```

The `resource:` part here is not a part of XSLT, but a part of Hovercraft! It tells the XSLT translation that the file specified should not be looked up on the file system, but as a Python package resource. Currently the `templates/reST.xsl` file is the only XSLT resource import available.

If you need to change the way reStructuredText is rendered you need to make a copy of that file and modify it. You then need to make a copy of the main template and change the reference in it to your modified XSLT file.

None of the XSLT files need to be copied to the target, and should not be listed as a resource in the template configuration file.