idoc User Guide

ILS Contributors

December 2, 2017

Contents

Chapter 1

idoc User Guide

Prerex

http://www.independentlearning.science/tiki/None: Someone wanted to see what this looked like, so it's here. Normally you can just leave the prerex block out when there are none.

Introduction

idoc is a language and a program. The language is what was used to write this document. The program is what rendered it (in both LaTeX and HTML5). It has a number of features that are not found in other humane markup languages. The most critical of these is support for allowing prerequisite resolution, though this is technically implemented by a static checker after parsing. idoc was originally based on asciidoc, but has since diverged significantly.

1.1 Goals

Get the Source!

You will probably want to examine the source for this document along with the rendered content. It can be found here.

idoc was created because no other existing markup language had the properties we needed for the website. We describe these properties and how the extant humane markup languages failed to have them.

Prerequisite resolution This is the big one. The website is designed around the idea of prerequisites, and so prerequisite resolution is a necessary thing idoc has to be able to do. It would have been *possible* to add this to Asciidoc, but not fun. It would have been *impossible* to add this to Markdown.

Typed Blocks A "strongly-typed" block is where one declares the purpose of a section of text explicitly. idoc uses the strongly-typed blocks to enable a bunch of cool features. One of those features is being able to take a doc (or a bunch of docs) and take all the theorems, lemmas, propositions, corollaries, definitions and examples, stick them all in a new document and basically generate a useful review sheet for you automatically. It also means that you don't manually have to describe the formatting of these things; idoc takes care of that for you.

Extensible Blocks On the other hand, having "extensible" blocks means that we can add new types of blocks – and new features to them – as needed. For example, if idoc is use a lot by computer science students, it might be necessary to add a block that allows you to download example code. It might even be cool to add that functionality to the already existing `code` blocks. These kind of additions are *impossible* in Markdown, *difficult* in Asciidoc, and *easy* in idoc.

Easy to Learn by Example While Asciidoc is a powerful markup language, its syntax can be somewhat irregular. It uses different types of delimiters to denote different types of blocks. Unless you have memorized what each style of delimiter means, this makes the markup hard to understand as a beginner. idoc was designed to be more regular – you should be able to figure out what is going on given just the source and the output. More importantly, you should be able to make *changes* to the source confidently, based on what you already see in it. Most edits to sites like Wikipedia are small ones; users like that don't want to learn the entirety of a markup language. They just want to get in, make their edit, and get out. Hopefully idoc allows them to do this.

Power with Limitations Asciidoc and Markdown allow you to insert inline HTML into your documents. While this sounds nice, it means that it's unsuitable for content on a shared website. This is because Javascript can be embedded in HTML, and that would mean we were letting people write and run untrusted programs on our user's computers. This is not acceptable. Such things also break the abstraction layer the markup languages provide, and makes it much more difficult to properly format the result of the markup language automatically. We prefer providing the features users want directly, instead of having them rely on such "escape hatches".

Haskell Implementation We are using the Haskell programming language to make the site because there is simply no better language out there. There does not exist (at the time of writing) a good implementation of Asciidoc for Haskell, and since Markdown is unsuitable for our purposes it did not matter *how* good the implementation of it was. Since we would have to write our own implementation of another markup language anyway, it was not a major extra investment to write an implementation of a new one instead.

Hopefully you can forgive us for forcing you to use an untested language!

1.2 Features

Basics

idoc is a *humane* markup language, which basically means it doesn't suck to write in it. For example, this is a **paragraph**. You just type like normal. They are separated by a blank line, as is every other **complex content** in idoc. Complex content is one of a paragraph, a list, or a block.

This is now a new paragraph. I can **italicize** text by surrounding it with with *underscores* (`_`) and **bold** it with **asterisks** (`*`). I can make it **monospace** (like "computer code") using `backticks` (`‘`).

Every idoc document must have a **title**, which is the first line of the document and looks like this: `= Title`. The space between the `=` and the title text is important. To make new **sections**, type `== Section Name`. Note that the section title must be separated from the previous content by a newline. For example,

```
I am a paragraph!  Hear me roar!

==New Section!

Another paragraph.  RAAAAAAAAAAAWR!
```

is correct, while

```
I am a paragraph!  Hear me roar!
==New Section!

Another paragraph.  RAAAAAAAAAAAWR!
```

would not display what you want correctly. Subsections are the same deal: use `=== Subsection Name`.

Any line that begins with `//` denotes a comment. They will not be displayed in the final document. These are not yet implemented, but will be soon.

// This is a comment line. It will not be rendered. Block // comments are not supported because they are a pain.

Nearly everything in idoc can be given an identifier (or **ID**) using the syntax `[[#idName]]`. This immediately follows the thing you wish to identify.

```
Nearly everything in idoc can be given an identifier using
 ↪  the syntax
`[[#idName]]`.  This immediately follows the thing you wish
 ↪  to
identify.
```

To see how to refer to IDs and link to them, check out the .

Lists

Lists are of fundamental importance when constructing rich documents. However, currently `idoc` does not recognize complex content inside lists, though this is subject to change. This means you cannot nest lists. You may only write a single paragraph. We support unordered lists, ordered lists and labelled (description) lists.

1. This is an unordered list. It can contain paragraph contents.
2. It will look like bullet points in the final render.
3. Third main item.

```
-This is an unordered list.  It can contain paragraph
 ↪  contents.

-It will look like bullet points in the final render.

-Third main item.
```

1. This is a numbered list.
2. This is the second guy.
3. And so on...

```
.This is a numbered list.

.This is the second guy.

.And so on...
```

First Item This is a list where the items have labels.

Second Item Another item.

And So On ...

```
:First Item::This is a list where the items have labels.

:Second Item::Another item.

:And So On::...
```

1.3 Blocks

Blocks are a fundamental construct in idoc. They all look the same, basically. They begin with `@blocktype` on a line by itself, separated by the previous content by a blank line. A block can also have a title, which comes *after* the declaration of block type; it looks like `#Block Title`. Titles are optional, and a default title will be chosen based on the block type if not included. Next, we have the attribute list, which I'll talk more on below. Basically it looks like `[key=value,key-no-value,key2=value-2,...]`, and gives a list of **attributes** and optionally their **values**. Finally, we have the actual block itself, which is enclosed by the delimiters `---`. The syntax inside the block is block dependent, and we'll explore them all below.

Math

Inline math is done just using normal LaTeX syntax by doing enclosing text in dollar signs. $f(x) = \exp(-x^2)$. Display mode is done by using a `math` block, like so:

$$f(x) = \int_x^\infty g(t)\mathrm{d}t$$

```
@math
#Look Ma', I Have Equations!
---
f(x) = \int_x^\infty g(t) \mathrm{d}t
---
[[#eqnFDefn]]{Basic Math}
```

Note that that d will not be upright as it should be. We'll fix that later. (Fixed it! Use `mathrm{d}`!) Also note that we added an *ID* to the equation. IDs can be added to many things. They always appear immediately following the thing they identify.

There is also a more specialized `eqnarray` block, which numbers equations. Note however that *unlike* the LaTeX eqnarray, you do not put a linebreak (\ ) at the end of each line. The ampersands (`&`) tell MathJax where to align line to. So, for example, the equations below have a "left side", an "equals sign" in the middle, and a "right side".

$$f(x) = \quad 2x + 3g(y) = \quad y^2 \implies (f \circ g)(x) = \quad 2x^2 + 3 \tag{1.1}$$

```
@eqnarray
---
f(x) &=& 2x + 3
g(y) &=& y^2
\implies (f \circ g) (x) &=& 2 x^2 + 3
---
```

Note that we can use the standard `label{}` macro within the `math` or `eqnarray` environments to assign IDs to individual equations. Then we can use `ref{}` or syntax to refer to them:

$$y = \qquad mx + b \tag{1.2}$$

```
@eqnarray
---
\label{importantEquation} y &=& mx + b
---
```

Refer to the *important equation* 1.2. Refer to . If you want to prevent an equation from being numbered, use `nonumber` as usual.

$$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} + \Lambda g_{\mu\nu} = \qquad \frac{8\pi G}{c^4}T_{\mu\nu}$$

```
@eqnarray
---
```

```
R_{\mu \nu} - \frac{1}{2} R g_{\mu \nu} + \Lambda g_{\mu
↪  \nu} &=& \frac{8 \pi G}{c^4} T_{\mu \nu} \nonumber
---
```

Axiom 1. *An axiom looks like this.*

$$x \cdot y = y \cdot x \tag{1.3}$$

```
@axiom
---
An axiom looks like this.

@eqnarray
---
x \cdot y = y \cdot x
---
---
```

Definition 1. a **number** is an element of an field.

```
[term=number]
@definition
---
a *number* is an element of an field.
---
[[#def-number]]{Definition: Number}
```

Theorem 2.

$$(x + y)^2 = x^2 + 2xy + y^2. \tag{1.4}$$

Proof. Use geometry.

□

```
@theorem
---

@eqnarray
```

```
---
(x + y)^2 = x^2 + 2 xy + y^2.
---

+++

Use geometry.
---
[[#thm-foil]]{Theorem: FOIL}
```

Lemma 3. *This is a lemma. It's pretty easy to prove.*
For $n > 2$ there exists no integer solutions to the equation:

$$x^n + y^n = z^n \tag{1.5}$$

Proof. It's trivial. □

```
@lemma
#Easy Lemma
---
This is a lemma.  It's pretty easy to prove.

For $n > 2$ there exists no integer solutions to the
↪  equation

@eqnarray
---
x^n + y^n = z^n
---

+++

It's trivial.
---
[[#lemEasyLemma]]{Lemma: Easy One}
```

Proposition 4. *Hey, um, do you want to go out with me next Saturday?*

```
@proposition
---
```

```
Hey, um, do you want to go out with me next Saturday?
---
```

Corollary 5. *If $x, y > 2$ are prime then $x + y$ is not prime.*

```
@corollary
---
If $x, y > 2$ are prime then $x + y$ is not prime.
---
```

Conjecture 6. *The product of any two primes is also prime.*

```
@conjecture
---
The product of any two primes is also prime.
---
```

Proof. Just intuit.

□

```
[theorem=lemEasyLemma]
@proof
---
Just intuit.
---
```

Emphasis Blocks

The following blocks are designed to emphasize certain things within the text. The first is the `quote` block, used for quotations.

> The internet is the most important invention since gravity.

```
[author=Albert Einstein]
@quote
#A Quote from the "Great One".
---
```

```
The internet is the most important invention since
↪ gravity.
---
[[#qEinstein]]{Einstein Quote}
```

Notice in the `quote` block we added an author attribution above as an *attribute* of the quote block contents. **Attribute lists** always come *just* before the thing they modify. In this case, we are modifying the "body" of the `quote` element, so it comes just before the body.

The next type of block is the `sidenote`. It is used to separate a paragraph from the main text, perhaps because it is important. Or perhaps it's unimportant... but then why is it there at all!?

Please be Aware:

This content will be rendered off to the side of the document. You are allowed any amount of complex content here. An equation: $F = \frac{dp}{dt}$.

```
@sidenote
#Please be Aware:
---
This content will be rendered off to the side of the
↪ document.  You are
allowed any amount of complex content here.  An equation:
↪ $F =
\frac{dp}{dt}$.
---
```

For more specialized blocks similar to `sidenotes`, use an `admonition` instead. They look nicer and are more eye catching. Here are some examples; notice how again we use an attribute list to specify the type of admonition.

This is a Warning Block

This will show a little warning symbol next to it.

```
@warning
#This is a Warning Block
---
This will show a little warning symbol next to it.
```

```
---
[[#warningBlock]]{Warning}
```

This is an Info Block

This will show a little info symbol next to it.

```
@info
#This is an Info Block
---
This will show a little info symbol next to it.
---
[[#infoBlock]]{Info}
```

This is a Caution Block

This will show a little caution symbol next to it.

```
@caution
#This is a Caution Block
---
This will show a little caution symbol next to it.
---
[[#cautionBlock]]{Caution}
```

This is a Tip Block

This will show a little lightbulb symbol next to it.

```
@tip
#This is a Tip Block
---
This will show a little lightbulb symbol next to it.
---
[[#tipBlock]]{Tip}
```

If you're worried about all the wasted space in the HTML output, don't be! It's only wasted because I didn't put text between each block. You would not normally have $1 + 4$ of these things in a row.

Pedagogical Blocks

These next blocks are useful when you want to offer extra information to the reader. In the case of a `connection`, that information may be more advanced than what the reader understands. In the case of an `intuition`, it's information that is technically contained in the main text, but is a powerful/neat way of thinking about the material. Please use good judgement when deciding when to use these.

A connection

Connection Prerex

http://www.independentlearning.science/tiki/Basics/Logic: Mostly only *first-order* logic will be used, but some *second-order* stuff is important for stuff later on, so we put it here.
http://www.independentlearning.science/tiki/Basics/Fun: And so on...
A `connection` is useful when you have content connected to the main text but which isn't super necessary, or requires more advanced techniques than the rest of the text. This is because connections are allowed to have their own prerequisites, in addition to the main doc's prereqs.

Rendering

If you're viewing this full-screen on a computer, you'll notice that unlike the other some other blocks we've seen, connections are rendered across the full screen. In general, you should *never* rely on the position of something relative to something else, or its size. This is because idoc has been designed to have pluggable themes, and the web renderer creates *responsive* html that is free to reorganize content to better match the devices size. Therefore, **whenever you refer to something, please please please use an ID**.

Connections are for when you want to connect with an outside idea, but don't want to have this interesting connection create new prerequisites for your doc. For example, in the article on topology, we might want to talk about how is difficult to prove two things are *not* homeomorphic, and how the fundamental group is useful because of its status as a topological invariant. However, we don't know if the readers actually know group theory. This would be the time you would use a connection – group theory is not needed to understand homeomorphisms, but it *is* needed to understand this small aside.

When the site is complete, connections will start out open or closed depending on if the reader has the necessary prereqs or not.

```
@connection
#A connection
---
@prerex
#Connection Prerex
---
/Basics/Logic{Mostly only _first-order_ logic will be
 ↪  used, but some _second-order_ stuff is important for
 ↪  stuff later on, so we put it here.}
/Basics/Fun{And so on...}
---

A `connection` is useful when you have content connected
 ↪  to the main text
but which isn't super necessary, or requires more advanced
 ↪  techniques
than the rest of the text.  This is because connections
 ↪  are allowed to have
their own prerequisites, in addition to the main doc's
 ↪  prereqs.

@info
#Rendering
 ---
If you're viewing this full-screen on a computer, you'll
 ↪  notice that
unlike the other some other blocks we've seen, connections
 ↪  are
rendered across the full screen.  In general, you should
 ↪  _never_ rely
on the position of something relative to something else,
 ↪  or its size.
This is because idoc has been designed to have pluggable
 ↪  themes, and
the web renderer creates _responsive_ html that is free to
 ↪  reorganize
content to better match the devices size.  Therefore,
 ↪  *whenever you
refer to something, please please please use an ID*.
---
[[#renderingInfo]]{Rendering Info}

Connections are for when you want to connect with an
 ↪  outside idea, but
```

```
don't want to have this interesting connection create new
prerequisites for your doc.  For example, in the article
<</Math/Topology/Homeomorphism#>>{on topology}, we might
↪  want to talk
about how is difficult to prove two things are _not_
↪  homeomorphic, and
how the fundamental group is useful because of its status
↪  as a
topological invariant.  However, we don't know if the
↪  readers actually
know group theory.  This would be the time you would use a
↪  connection --
group theory is not needed to understand homeomorphisms,
↪  but it _is_
needed to understand this small aside.

When the site is complete, connections will start out open
↪  or closed
depending on if the reader has the necessary prereqs or
↪  not.
---
[[#myConnection]]{A Connection}
```

But What *is* the Determinant?! – An Example Intuition Block

The determinant from a geometric point of view is used to talk about *volume*. If you grok this, then everything else will fall into place. For example, the reason a linear transformation is invertible if and only if its determinant is non-zero is because when it *is* zero, the map is going to map a set of basis vectors in your n-dimensional space into some smaller m-dimensional space. This m-dimensional space has zero "volume" compared to the n-dimensional one, and so you're basically losing a whole dimension. There is no way to then map linearly back to the bigger space. It would be like trying to find a linear function that maps the square to the cube!

```
@intuition
# But What _is_ the Determinant?! -- An Example Intuition
↪  Block
---
The determinant from a geometric point of view is used to
↪  talk about
_volume_.  If you grok this, then everything else will
↪  fall into
```

```
place.  For example, the reason a linear transformation is
 ↪  invertible
if and only if its determinant is non-zero is because when
 ↪  it _is_
zero, the map is going to map a set of basis vectors in
 ↪  your
$n$-dimensional space into some smaller $m$-dimensional
 ↪  space.  This
$m$-dimensional space has zero "volume" compared to the
$n$-dimensional one, and so you're basically losing a
 ↪  whole
dimension.  There is no way to then map linearly back to
 ↪  the bigger
space.  It would be like trying to find a linear function
 ↪  that maps
the square to the cube!
---
```

`examples` and `exercises` are also a key way for people to learn. The difference between the two is that essentially an exercise is an example without a solution.

Example

Prove that every integer greater than zero is positive.

The proof is pretty simple. Try to follow along.

First, assume not. But that's not the case. QED.

```
@example
---
Prove that every integer greater than zero is positive.

+++

The proof is pretty simple.  Try to follow along.

First, assume not.  But that's not the case.  QED.
---
[[#exExample]]{An Example}
```

Exercise

What is largest rational number that can be written as a fraction of 5 and 7?

```
@exercise
---
What is largest rational number that can be written as a
↪  fraction of
$5$ and $7$?
---
```

Links

Links come in three flavours. The first is your standard link, called an **out link** or **olink** in idoc. It's written like `this` and appears like this. Note that the `http(s)://` part of the link is *mandatory*. Olinks are restricted in idoc to appear in only very specific places, for example in `furtherreading` or `bibliography` blocks. This is because the text of a doc should be *self-contained*. You should not *need* to go to an outside source to understand a concept. However, that doesn't mean that outside sources are useless – Wikipedia is an excellent reference for topics you've just learned about. It just means you don't want to interrupt the flow of the document just anywhere. This is why idoc enforces this rule.

The second flavour is a link within the current doc, called an **internal link** or **ilink**. For example, when we used `ref`{} back in the Math section, we were using ilinks. Refering to anything on the page which has an ID is an ilink. They are written like , and appear like . That is, they are the `#` symbol followed by a valid ID.

The final flavour is the **back link**, or **blink**. This is a link to something contained within the prerequisite tree of the current doc. In their most general form, they are written like `<</Name/Of/Article#id-in-article>>`{`this right here`} and display like this right here. However, the `#` symbol and the ID following it can be left out if you just want to refer to the article as a whole. Notice that you are not allowed to link to another article that isn't within the prerequisite tree from just anywhere. This is by design; articles should only depend on the knowledge from previous articles. This restriction allows many of the features that the ILS website hopes to deliver, such as automatic prerequisite resolution for subjects you want to learn.

In all types of links, you are free to leave out the text inside the {}. This will just use the text inside the as the display text instead. Normally this is not what you want, however.

YouTube

We can embed YouTube videos if we think they are important. Please do not abuse this feature.

Images

We can embed important images too.

Bunnies make everything better.

1.4 Conclusion

That's about it for the basic syntax. We can fix the rest later.