Addressing File Format Compatibility in Word Processors

Peter Kelly & Louis Suárez-Potts

APACHECON EUROPE

CORINTHIA HOTEL
BUDAPEST, HUNGARY
— NOVEMBER 17-21, 2014 —

https://github.com/uxproductivity/Corinthia

So you want to write a word processor

1980s

Decide on feature set, come up with format that can express all your desired features. Aim for direct loading/mapping into memory, avoiding parsing for efficiency reasons. File format = internal data structure of your app.

1990s

Much the same, based on legacy approaches. e.g. MS Word's .doc.
Users expecting import/export capabilities as they move to newer software.

2000s

Design an XML-based format, and try to get it approved as a standard. Argue that everyone should support your “universal” standard. Lobby/bribe governments around the world to vote for it on standards committees.

2010s

Wow, there's a lot of formats out there. None has been universally accepted. I'm writing a new program, how do I deal with this huge mess?

https://github.com/uxproductivity/Corinthia

There will never be "the" standard

It was a nice idea. But it never happened. There's lots of reasons, but little point in rehashing old debates here.

We have to live with different formats, chosen for various reasons. But we want tools that can work with them. And we want to be able to create new tools which can work with all popular file formats without requiring lots of effort.

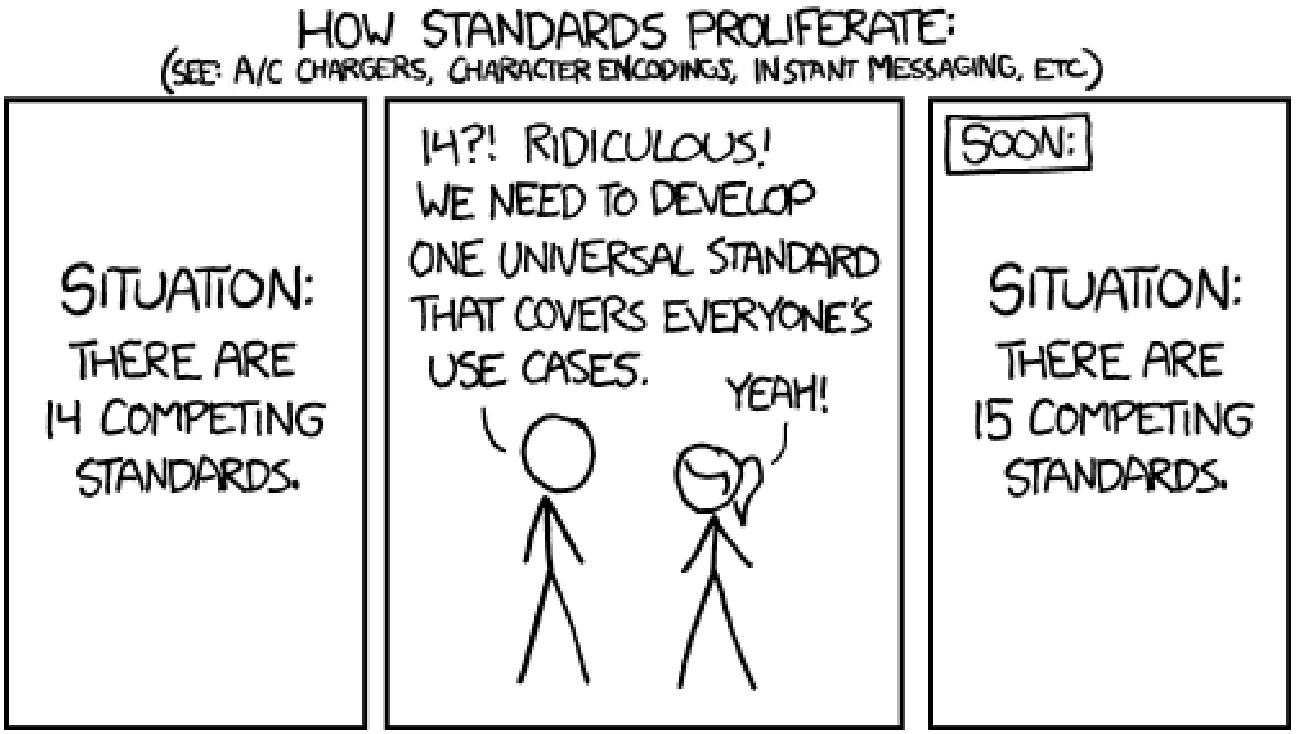
Idea: Open source project to build an easy-to-use library supporting many file formats. Let app developers focus on their apps.

How to support multiple formats?

We need *one* data model that the app can work with, and APIs to abstract over the differences between various file formats, so the app doesn't care which it's using

Data model = file format, more or less (the latter just specifies encoding)

So we need a new, "universal" data model/file format - back to square one?


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

http://xkcd.com/927/

https://github.com/uxproductivity/Corinthia

What format is most of the worlds online information in?

HTML/CSS has become by far the world's most common file format for storing rich text.

Millions of developers are familiar with it

Several widely-used rendering engines (e.g. WebKit) available that can easily be embedded in an app

Yet there's still a curious gap. Few word processors use HTML natively, and instead support only import/export to their own formats

"But HTML is only for web pages, not documents". I don't see these as inherently different things. It's a false distiction.

https://github.com/uxproductivity/Corinthia

Ok, so we'll pick HTML as the data model

Like all formats, HTML lacks certain features:

- Tabs
- Page breaks
- Header & footers
- Designed for continuous media, not paged media

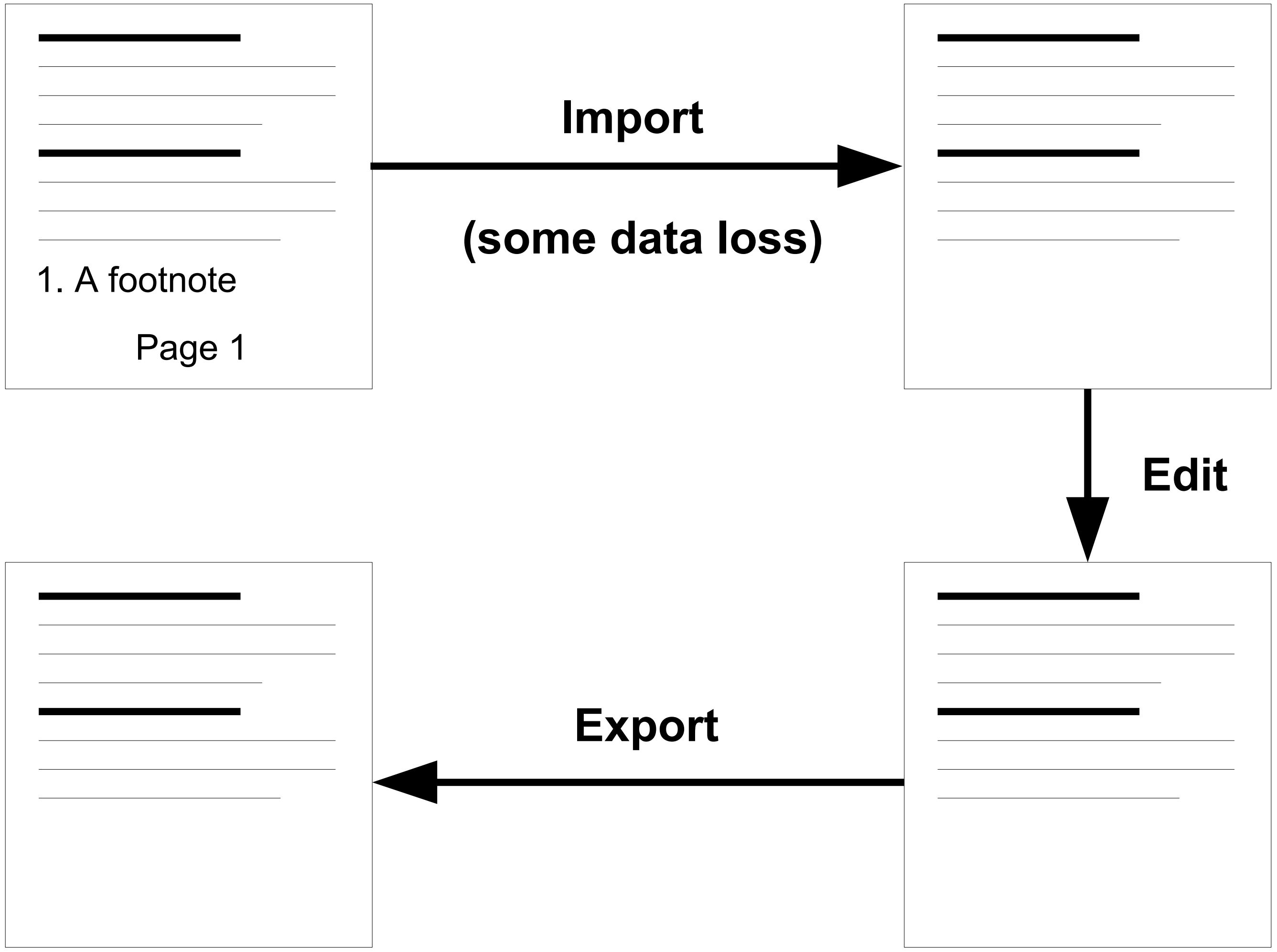
CSS3 improves on some of these features, but rendering support lacking

There will always be an "impedance mismatch" when translating between file formats. Data will get lost in the process. If we want perfect fidelity, no translator is going to provide that.

However:

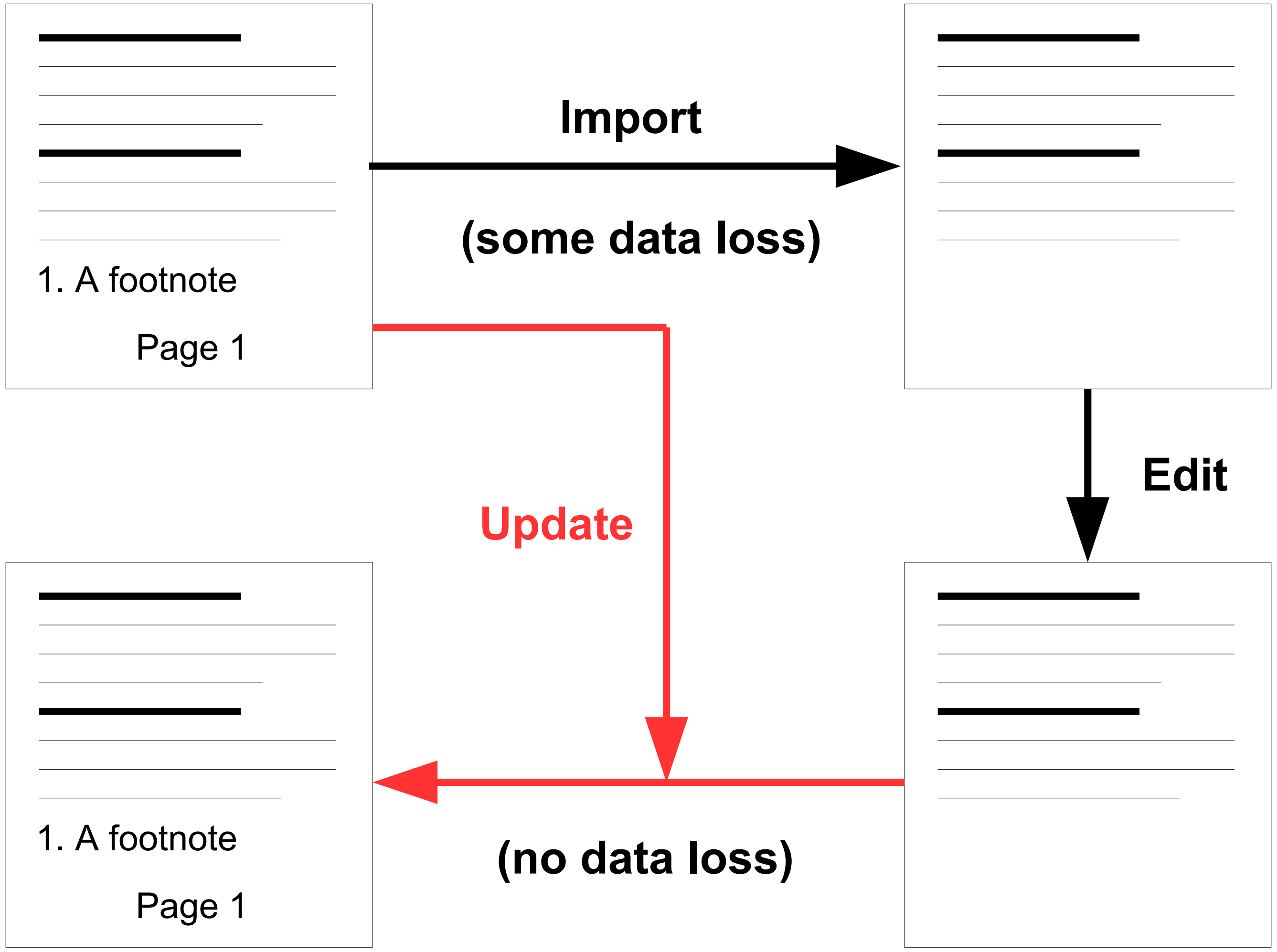
If we just want to *modify* a document - using HTML as an in-memory data model - then there's a way to avoid loss-on-save: **Bidirectional transformation**

Option 1: Import/Export



https://github.com/uxproductivity/Corinthia

Option 2: Bidirectional Transformation



https://github.com/uxproductivity/Corinthia

BDT operations

Source = Document in non-native format (e.g. OOXML)
View = Document in native format (e.g. HTML)

get(Source) → View

Create an abstract view containing a subset of the information in the source document. Maintain information allowing us to associate elements in the view with elements in the source.

update(Source,View') → Source'

Make a set of changes to the source document to make it consistent with a modified view. Information in the source that was not included in the original view is left untouched.

create(View) → Source

Special case of update with an empty source. Used for creating new documents.

https://github.com/uxproductivity/Corinthia

BDT laws

put(S,get(S)) = S

Constructing a view from a source, then immediately calling put on that same source should result in no changes being made to the source.

get(put(S,V)) = V

After updating a source based on a given view, and then constructing a new view from that source, the two views should be the same

get(create(V)) = V

If you create a new source from a given view, then create a view from that source, the two views should be the same

BDT references

Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce et. al. *Boomerang: Resourceful Lenses for String Data*. Technical Report MS-CIS-07-15 Department of Computer and Information Science University of Pennsylvania. November 2007.

http://www.cis.upenn.edu/~bcpierce/papers/boomerang.pdf

Benjamin Pierce. *Foundations for Bidirectional Programming*. ICMT2009 - International Conference on Model Transformation. June 2009.

http://www.cis.upenn.edu/~bcpierce/papers/icmt-2009-slides.pdf

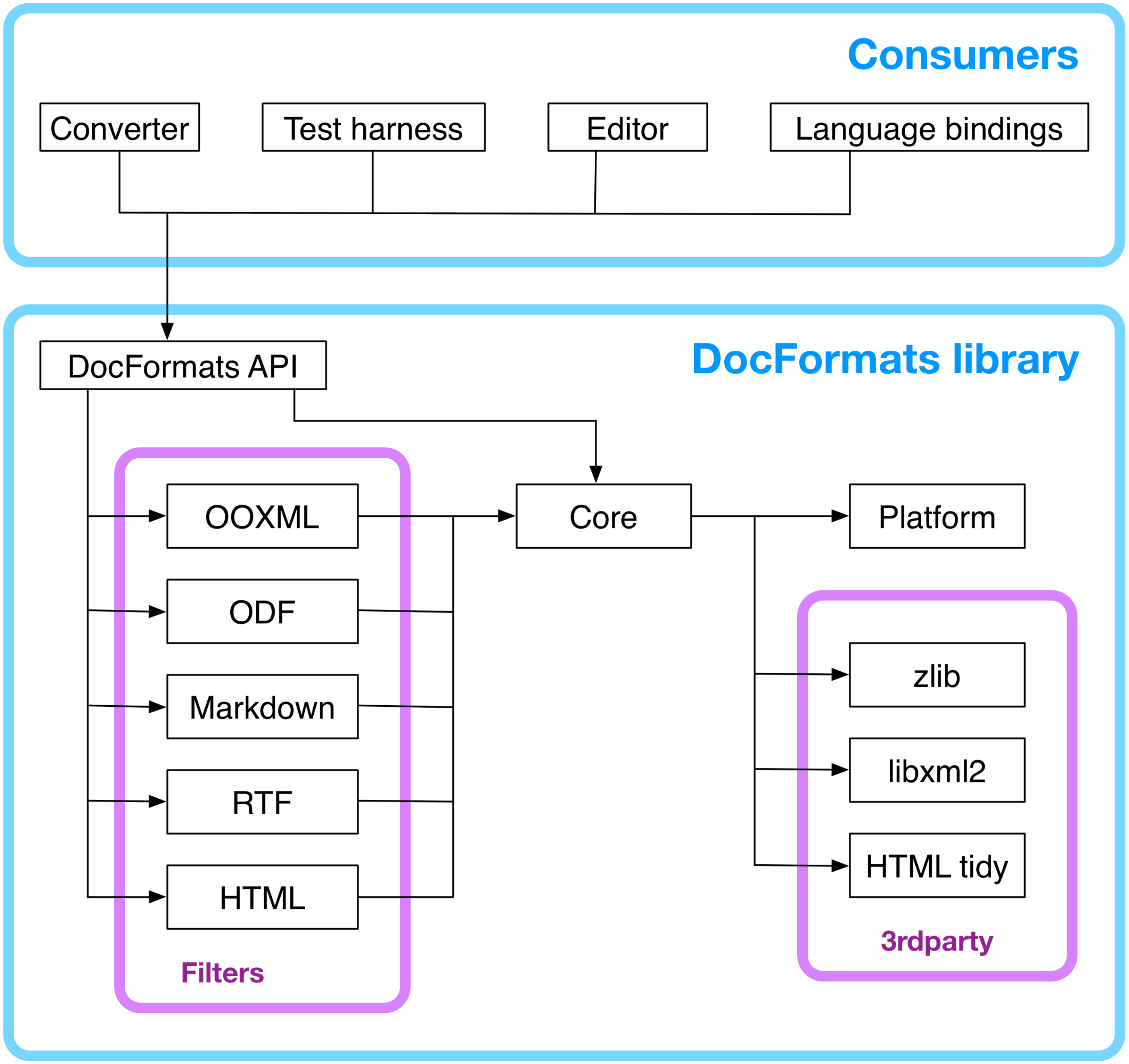
The DocFormats library

DocFormats is a component of Corinthia that implements file format conversion using bidirectional transformation. It is a C library with minimal dependencies.

Can be used for

- Loading and saving files to be used with the editor
  (the editor only understands HTML)
- One-off conversions with a command-line tool
- Batch conversions
- Any other situation when you need to convert or update file formats
  You can build apps that manipulate documents in HTML format, then use them with documents stored in docx – and in the future, other formats (e.g. ODF)

https://github.com/uxproductivity/Corinthia



https://github.com/uxproductivity/Corinthia

Demo

https://github.com/uxproductivity/Corinthia

Questions?

https://github.com/uxproductivity/Corinthia