
Pay some attention to the
box inside the box

Vanity slide

Jarno Elovirta

A DITA hobbyist

@jelovirt www.elovirta.com jelovirt@gmail.com

www.wunderdog.fi

Agenda

- Why did you/they do it that way
- What's new
- What do we have in mind

Tweet #ditaotday for questions

The box inside the box

At high level, a transtype in DITA-OT is split into parts, preprocessing and transtype specific conversions.

The preprocessing part has been growing over time

Technology stack

- Why do we use Java to process documents when we could use XSLT for everything? Why Java in the first place?
- XSLT tried to be immutable and in some case you can't afford that.
- XSLT 3.0 has streamable features, but DITA-OT got started in XSLT 1.0 era and SAX is sometimes faster.
- SAX and DOM can be made to use less memory, especially SAX.
- In the end, you can go fully either way.

Which XML APIs

- Plenty to choose from, why SAX and DOM.
- Usually SAX vs. DOM, but historically it was StringReader and StringWriter.
- SAX allows mapping based processing, implemented with XMLFilter. Code reuse becomes easier and each mapping operation is easier to test.
- DOM came with the batteries, even if it's ill-suited in every platform
- You have to pick one and once you've picked it's not easy to change to the other one.

Why all the IO

- OT preprocessing is a series of Ant targets where each targets modifies the content files or reads from them. Reading and writing XML is IO intensive.
- Processing DITA requires that some things are done in a particular order and in particular blocks. Thus we need multiple pass-throughs.
- The biggest obstacle in trying to improve this are the current extension points. Without them we'd be able to optimize the reads more.
- Memory is cheap, but that's no excuse to use it all. We can't afford to deliberately leave some users out.

If you don’t have anything nice to say, don’t say anything at all.

Ant

What changed for DITA 1.3

- Most of the changes only affect preprocessing
- Overall architecture of preprocessing didn't have to change that much.
- The order of steps was changed because processing mandated it.

Same topic fragments

- Simple SAX filter
- Done first only for @conref, then for @href

Branch filtering

- Implemented as a separate filtering process, not part of DITAVAL filtering
- Internally a two step process: branch duplication and filtering
- For convenience, map merge was moved to take place first
- Required to take place before key resolution

Scoped keys

- DITA 1.2 implementation read keys during initial parse phase
- Instead of a simple map, a scope tree is constructed
- Scope qualified key references are implemented by bi-directional cascading
- Like branch filtering, scoped keys create new resources
- DITA-OT 2.2 implementation doesn't cover every corner case

Future work

- Finish DITA 1.3 support
- Adding new libraries and updating existing ones
- Remove extension points that don't make sense anymore and/or are in the way of progress.
- Move as much of the code into preprocessing as possible.
- Split plugins into separate Github repos

Possible optimizations

- Abstract temporary file read/save to JAXP Source/Result
- Generalize element names and rewrite XSLT stylesheets to use base element names
- Combine modules to allow more piping

Discussion