



2022 DITA/Markdown Interoperability

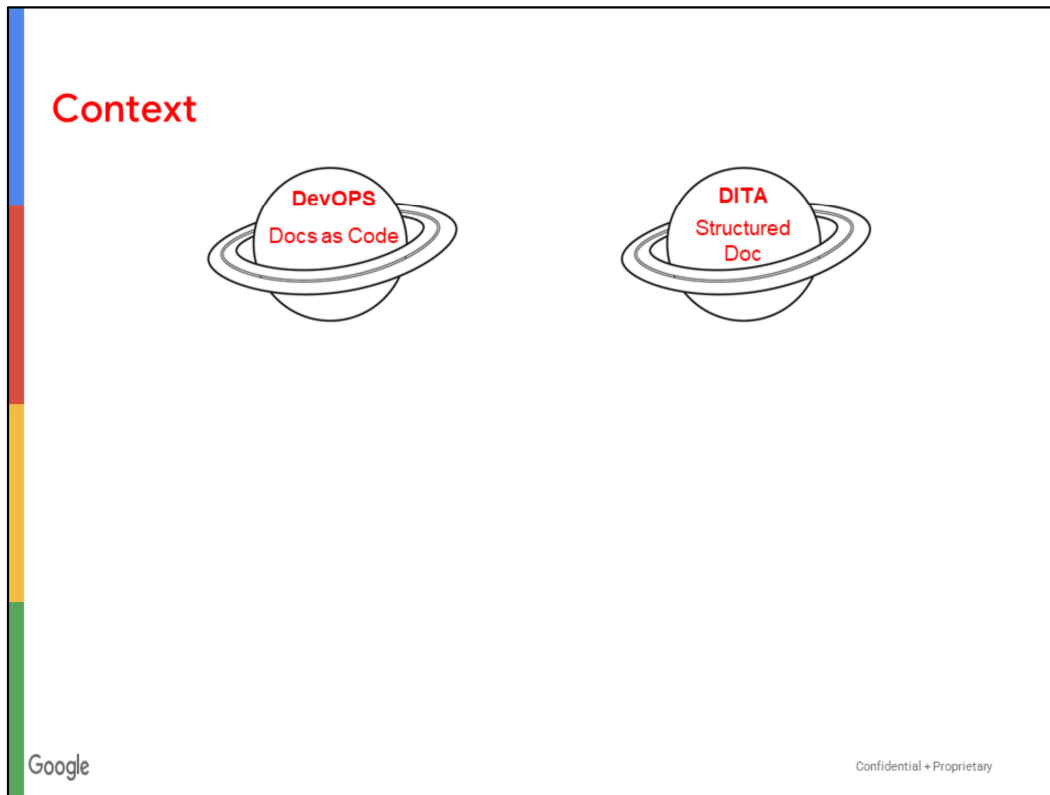
Stanley Doherty, Ph.D.
OASIS DITA Technical Committee
Google Cloud Platform

Boston DITA Users Group
August 10, 2022



Agenda

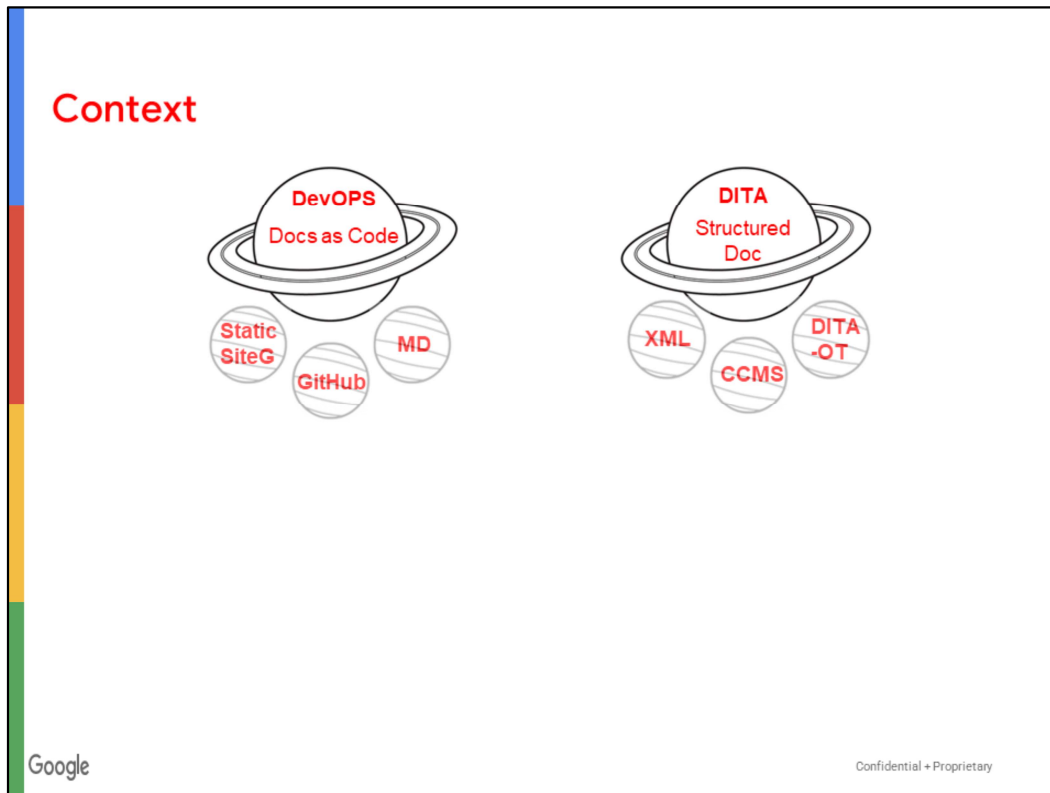
| | |
|------------------------|---|
| Context | Why are we talking about Markdown in a DITA user group? |
| Conversions | How do we convert Markdown to DITA? How do we convert DITA to Markdown? |
| Publishing | How can we publish HTML that references both DITA and Markdown sources? |
| Single sourcing | Is it possible to single-source Markdown content so that it can be published in <i>both</i> DITA and Docs-as-Code environments? |



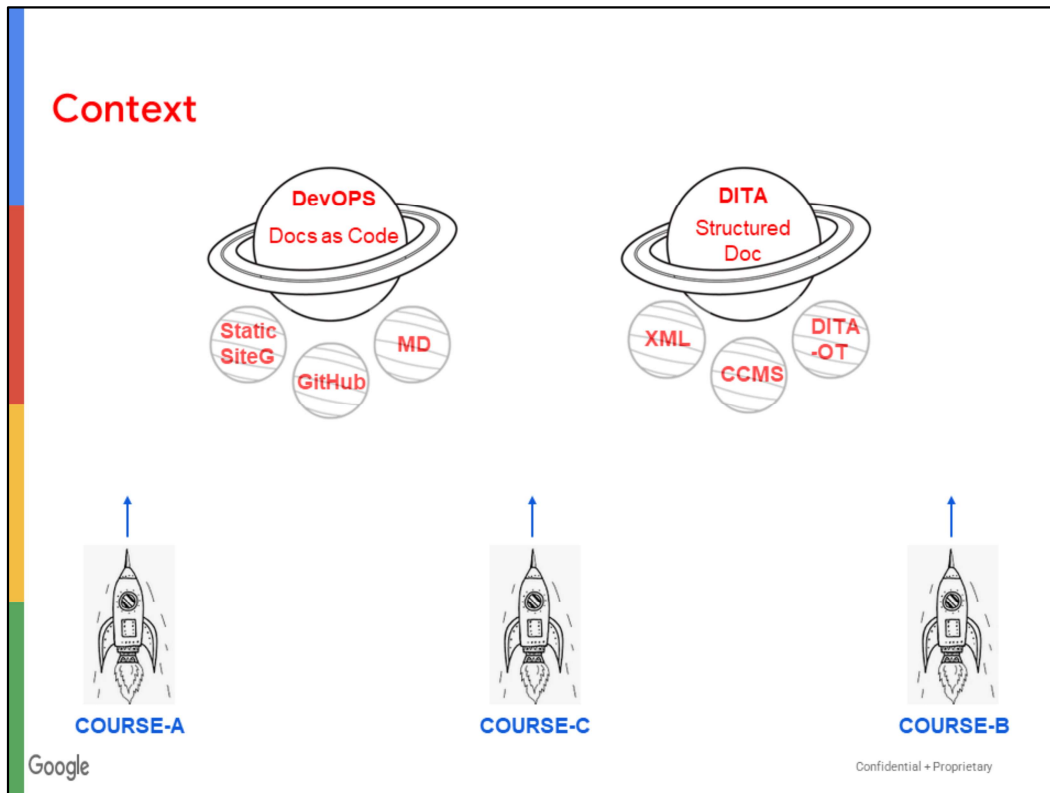
WARNING – stupid visual metaphor approaching at sub-warp speed!

Many content development teams find themselves trying to navigate between two very popular frameworks:

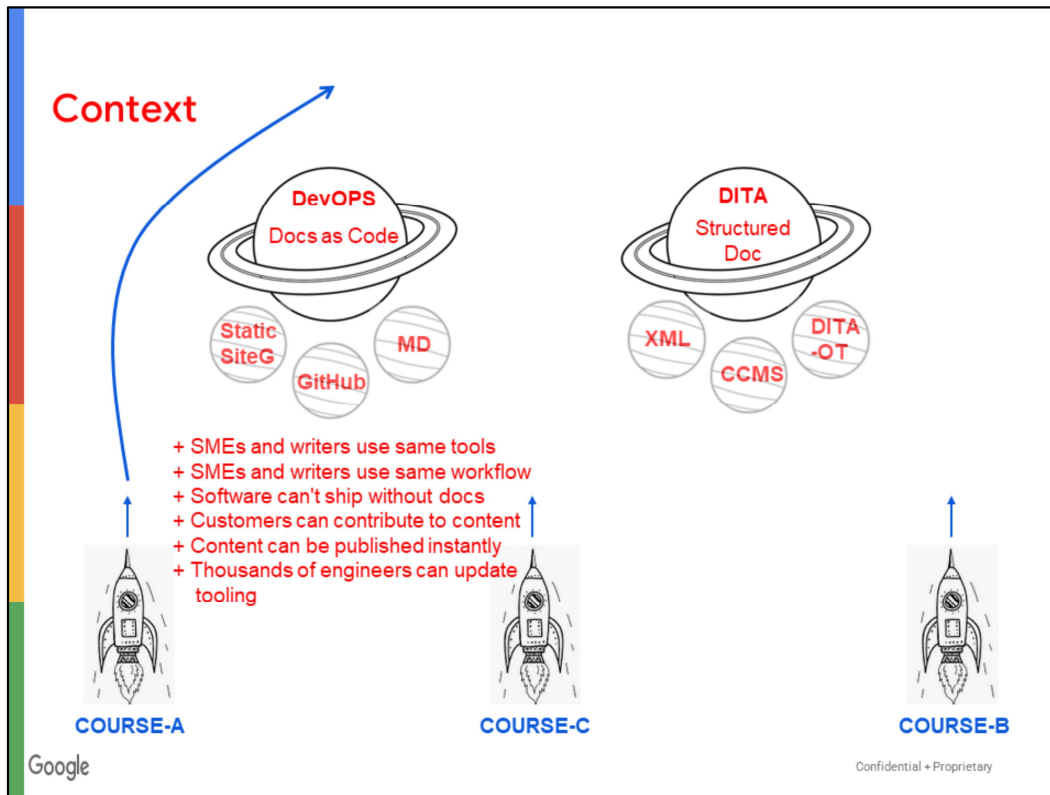
1. **OASIS DITA**: The industry standard for topic-based, structured (XML) documentation.
2. **Docs-as-code**: The DevOPS-driven framework for authoring and publishing content using the same tools and processes as software engineers.



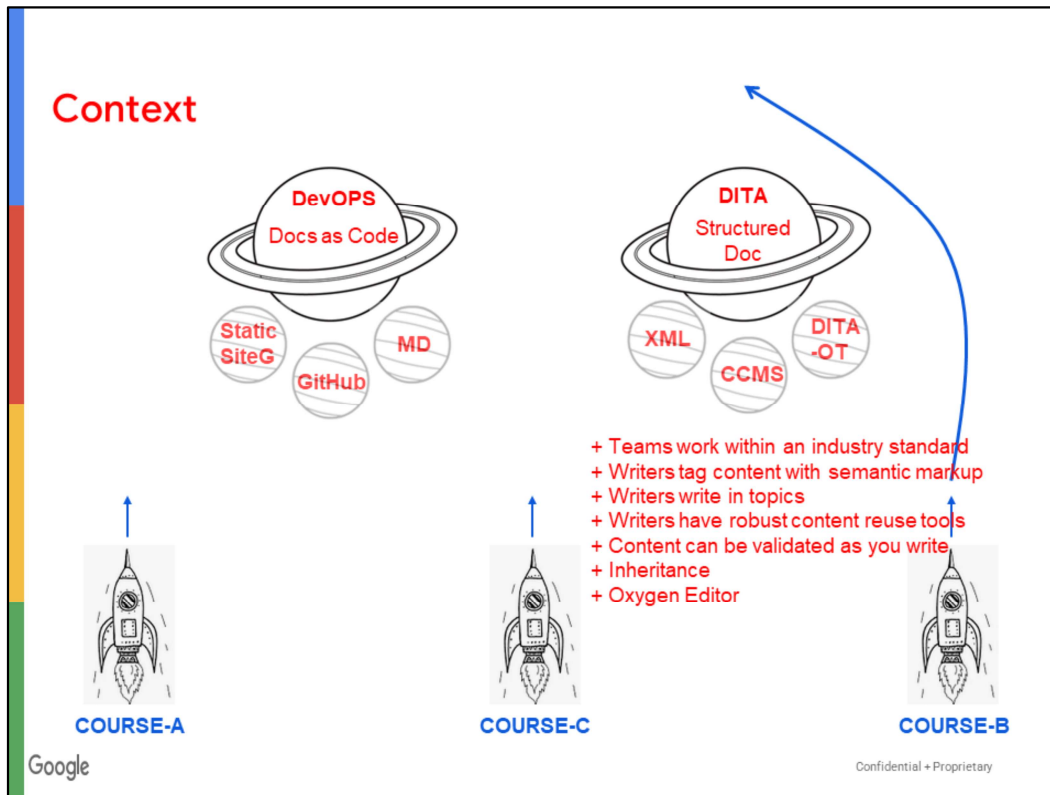
Within the gravitational sphere of influence for each of these frameworks, we see satellite technologies. Docs-as-code relies on static site generators. DITA relies on the DITA Open Toolkit. To understand how one of these frameworks operates, you'll eventually need to learn about satellite technologies. If you chase a satellite, you'll be pulled into its parent's gravity.



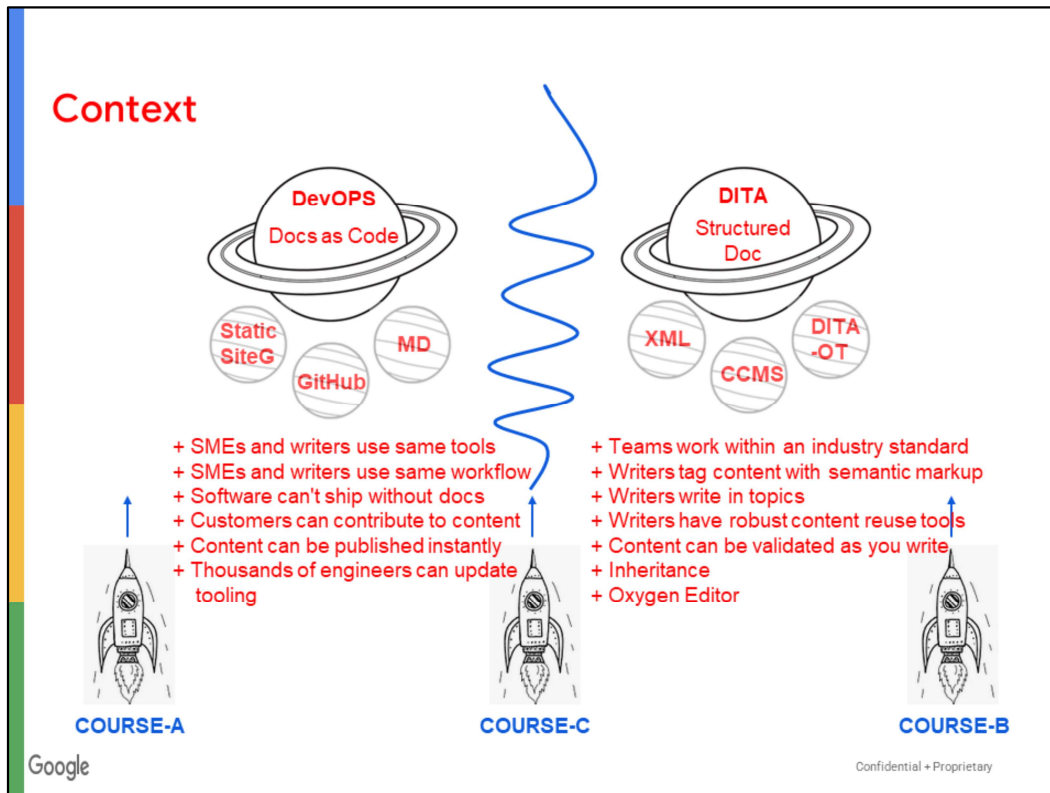
New content development teams need to set a course around or between these frameworks. Established teams in one framework may be asked to “play nicely” with another team, partner, or subsidiary committed to the other framework. If you are doing extensive work in software, needing to consider an alternative frameworks is mater of *when* and not *if*.



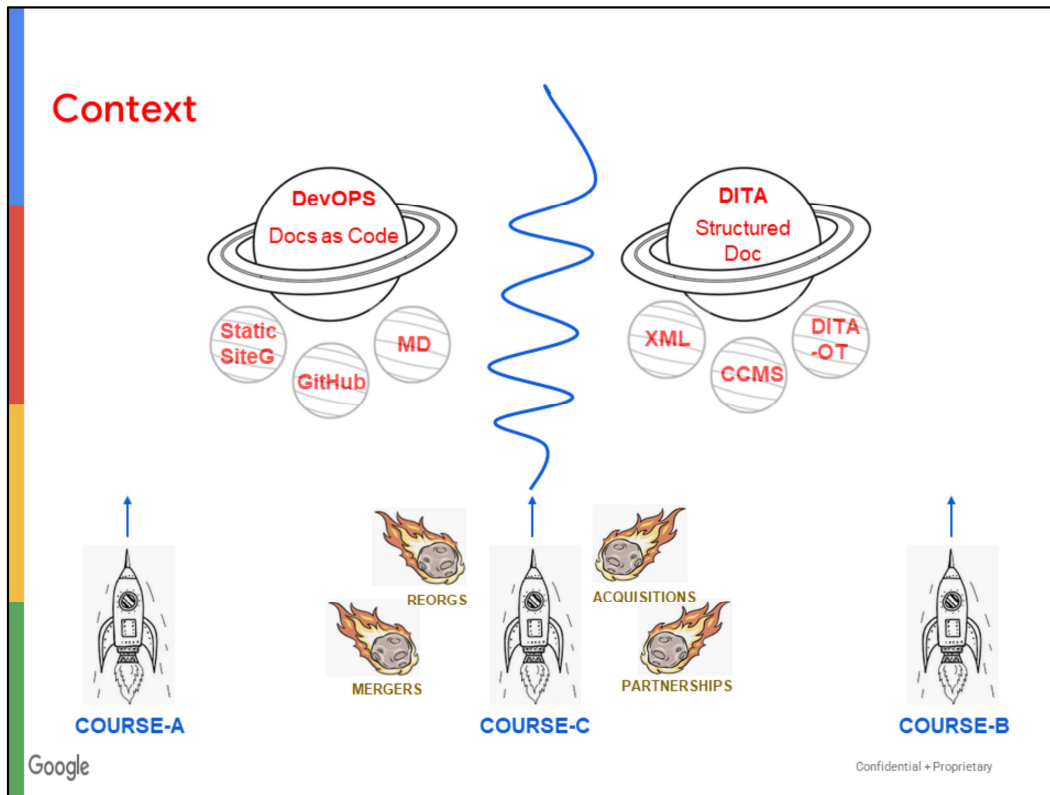
If your organization develops software or solutions for developers, the prevailing framework for content development is DevOPS docs-as-code. The center of gravity for DevOPS is tool and process integration. If engineering tools and processes are good enough to produce software deliverables, they should be adequate for technical writers to deliver content in support of that software.



If your organization works in a regulated industry, develops consumer products, or requires an ultra-smart content architecture, DITA and structured markup should be close to your flight path. The center of gravity here is writer efficiency and intelligent content.



If your organization has a mix of existing DITA and existing docs-as-code implementations, you be navigating between two strong gravitational pulls. You're in for a bumpy journey – or what Mark Twain said of *Pilgrim's Progress* – "interestin' but tough". If you can style the HTML generated from DITA and docs-as-code to look the same of your doc portal, these parallel solutions can co-exist for a long time. If you need to converge your authoring and processing infrastructure, then you need to make *very* clear to your stakeholders what they are obtaining and losing by choosing on framework over the other. The onus is on *you*, the publication specialist, to lay out the costs, benefits, and use cases *for each option*. You need to do some homework, sometimes on very short notice.



And – if you are currently doing a COURSE-B in DITA and do not believe that you'll ever have to deal with DevOPS docs-as-code, consider that business initiatives outside your control such as reorgs, mergers, acquisitions, or partnerships may force the conversation. Consider asking one of your talented junior writers to become your team expert of docs-as-code. A little insurance never hurts.

Conversions

From DITA to Markdown

DITA Open Toolkit See [Markdown](#) in DITA-OT docs.

You can convert a single topic or an entire map.

```
dita --input=my-topic.dita --format=markdown -output=output_markdown
```

```
dita --input=my-map.ditamap --format=markdown -output=output_markdown
```

You can choose flavors of Markdown.

- `format=markdown` (generic markdown conventions)
- `format=markdown_github` (GFM)
- `format=markdown_gitbook` (GFM + `summary.md` TOC file)

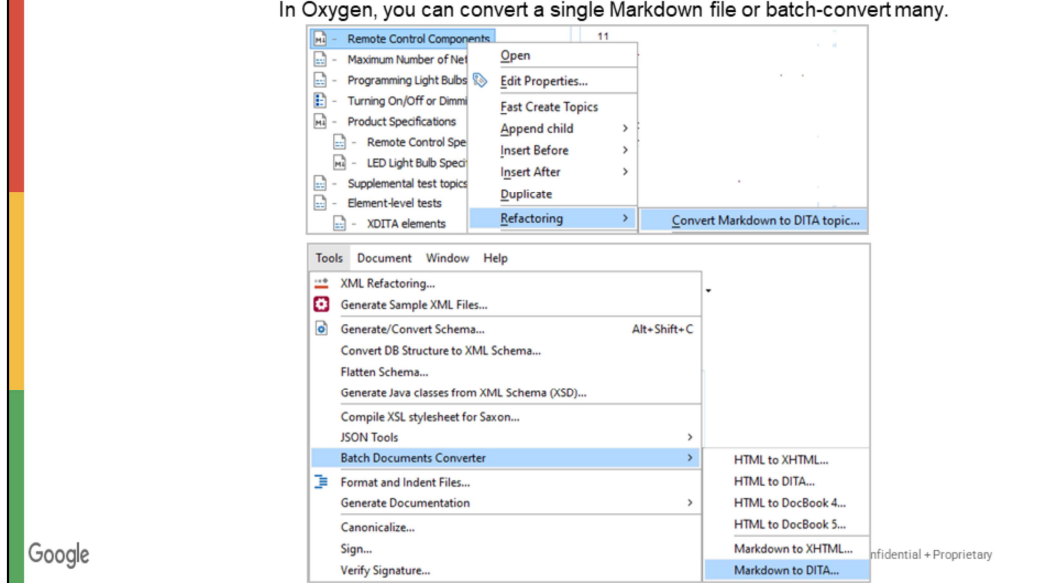
If you need to move some or all of your DITA content into Markdown, the DITA-OT provides robust support. NOT EVERY feature that you can do in DITA will have a Markdown equivalent, but the basic stuff moves over easily.

Conversions

From Markdown to DITA

Oxygen Editor

See [Working with Markdown Documents in DITA](#) in Oxygen docs.
In Oxygen, you can convert a single Markdown file or batch-convert many.



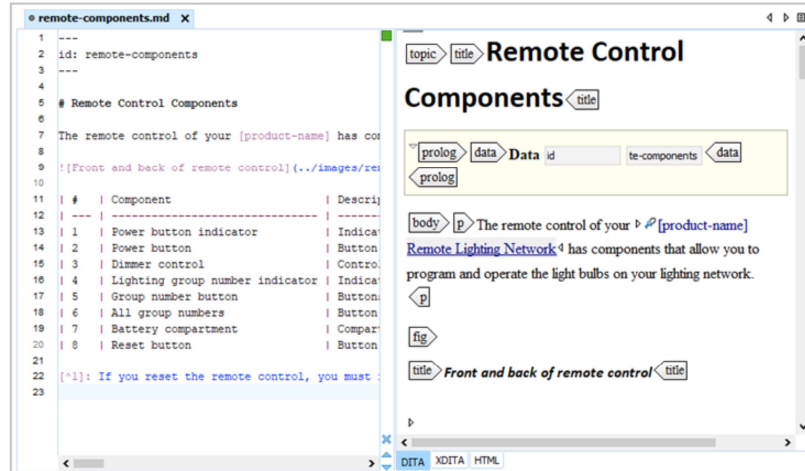
Oxygen Editor does a brilliant job converting basic Markdown documents into DITA topics.

Conversions

From Markdown to DITA

Oxygen Editor

See [Working with Markdown Documents in DITA](#) in Oxygen docs.
You can preview the conversion in the Editor.



Google

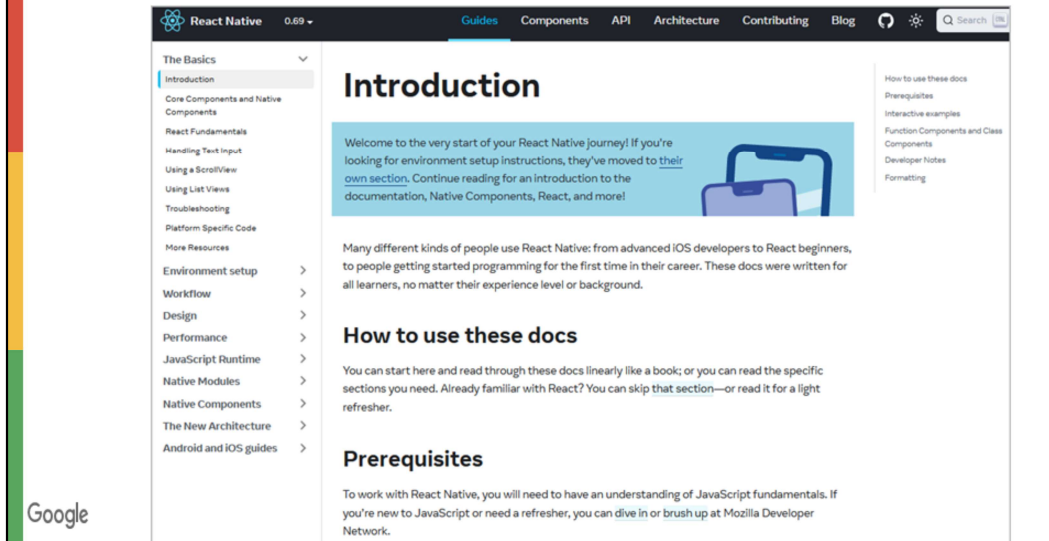
Confidential + Proprietary

When you open a Markdown document in Oxygen, it displays both the raw Markdown markup and the equivalent DITA markup that it has silently converted. This is also a great environment to debug Markdown markup that is not behaving correctly when converted to DITA.

Publishing

Markdown only

Static Site Generators See [Jekyll](#), [Hugo](#), [Sphinx](#), [GitBook](#), and [Docusaurus](#).
Sample: React Native - <https://reactnative.dev/docs/getting-started>.



Publishing. I recommend experimenting with Markdown-only static site generators such as Jekyll or Hugo. The screen shot here from Docusaurus 2.0 illustrates that the out-of-the-box HTML5 generated from static site generators is quite serviceable. Expect to have an "is this is good enough?" conversation with people in your organization who want to move away from XML and its "separate content fro formatting" axioms. If you have experience using a static site generator to build demo web sites, you will be in a better position to manage the conversations.

Publishing

DITA maps + DITA topics + (wicked simple) Markdown files

DITA maps

See [Working with Markdown Documents in DITA](#).

```
map  topicref
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
3  <map id="rlnmap">
4  <topicmeta>
5  <navtitle>Remote Lighting Network</navtitle>
6  </topicmeta>
7
8  <topicref href="xdita/intro-product.dita" format="dita" />
9  <topicref href="mdita/remote-components.md" format="markdown" />
10 <topicref href="xdita/max-number-bulbs.dita" format="dita" />
11 <topicref href="mdita/product-specs.md" format="markdown">
12 <topicmeta>
13 <metadata>
14 <audience type="administrator"/>
15 <prodinfo>
16 <prodname>Console</prodname>
17 </prodinfo>
18 </metadata>
19 </topicmeta>
20 </topicref>
```

Google

Confidential + Proprietary

The DITA-OT and XML IDEs such as Oxygen Editor can publish content that is authored in XML, Markdown, or a mix of the two. Your standard DITA 1.3 map has been capable of referencing Markdown documents for many years now. In addition, DITA maps can augment the information in a basic Markdown document with `<topicref>` metadata. You can keep these Markdown documents pretty dumb if you can compensate with wicked smart maps.

Publishing

DITA maps + Lightweight DITA topics + robust MDITA (Markdown) files

See [Lightweight DITA Testing](#) (GitHub repo).

```
map
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE map PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Map//EN" "lw-map.dtd">
3 <map id="rlnmap">
4   <topicmeta>
5     <navtitle>Remote Lighting Network</navtitle>
6   </topicmeta>
7
8   <keydef keys="product-name">
9     <topicmeta>
10      <linktext>Remote Lighting Network</linktext>
11    </topicmeta>
12  </keydef>
13  <topicref href="xdita/intro-product.dita" format="dita" />
14  <topicref href="mdita/remote-components.md" format="mdita" />
15
16  <topicref href="xdita/max-number-bulbs.dita" format="dita" />
17  <topicref href="xdita/program-bulbs-to-groups.dita" format="dita" />
18  <topicref href="dita/turn-on-off-dim-lights.dita" format="dita"/>
19  <topicref href="mdita/product-specs.md" format="mdita">
20    <topicref href="xdita/remote-specs.dita" format="dita"/>
21    <topicref href="mdita/led-specs.md" format="mdita"/>
22  </topicref>
```

Google

Confidential + Proprietary

The yet-to-be-released Lightweight DITA standard introduces specialized DTDs that allow your Markdown documents (MDITA) to inherit key values and to transclude content living in XML libraries. Worth checking out.

Single sourcing

XML DITA resources for static site generators – no XML parser



Google

Confidential + Proprietary

In plotting your course between the DITA framework and the docs-as-code framework, will you be able to develop content in XML and have it processed without modification in a static site generator? No. Static site generators currently do not have an integrated XML parser, so they cannot make much sense of DITA topics.

Single sourcing

Simple Markdown files between static site generators and DITA map publishing



Google

Confidential + Proprietary

As you've seen in the previous section, you can single-source basic Markdown topics so they can be published without modification through a static site generator or the DITA-OT. In some sense, that's makes DITA the more attractive framework for collaboration in the short run.

Single sourcing

"Optimized" Markdown files between static site generators and DITA map publishing

Bumps, potholes, and broken pavement



Google

Confidential + Proprietary

Generic Markdown was never designed to support the sorts of sophisticated things that we do as technical writers – global reuse, keyword reuse, filtering, conditions, topic-based authoring, metadata inheritance, and so on. So – if I am working in docs-as-code framework and I need or want to do these sophisticated things, I need to reach outside generic Markdown to add instructions from what is called a templating language such as Liquid or Django. Once I introduce these instructions, compatibility with the DITA-OT pipeline erodes quickly. These optimizations make what you can do in Markdown significantly more powerful – at the cost of interoperability. You cannot even single-source a Markdown file optimized with Liquid with one optimized by Django. It gets to be a bumpy road quickly.

Single sourcing

Docs-as-Code teams supplement basic Markdown code with metadata from YAML and processing instructions from an HTML template languages such as [Liquid](#), [Jinja](#), or [Django](#).

```
YAML headers      ---
                  title: What is Azure NetApp Files | Microsoft Docs
                  description: Learn about Azure NetApp Files, an enterprise-
                  class, high-performance, metered file storage service that
                  supports any workload type and is highly available.
                  services: azure-netapp-files
                  documentationcenter: ''
                  author: b-hchen
                  manager: ''
                  editor: ''

                  ms.assetid:
                  ms.service: azure-netapp-files
                  ms.workload: storage
                  ms.tgt_pltfrm: na
                  ms.topic: overview
                  ms.date: 10/04/2021
                  ms.author: anfdocs
                  ---
```

Google

Confidential + Proprietary

Many of the things we would put in our DITA <prolog>s, appear in YAML headers in optimized Markdown. This public sample from Microsoft is certainly readable, but not portable.

Single sourcing

Docs-as-Code teams supplement basic Markdown code with processing instructions from an HTML template language such as [Liquid](#), [Jinja](#), or [Django](#).

```
Liquid instructions    {% if product.title == "Awesome Shoes" %}
                        These shoes are awesome!
                        {% endif %}

                        {% assign foo = "bar" %}
                        {{ foo }}

                        {% capture snippet_content %}
                          {% render 'your-snippet-name' %}
                        {% endcapture %}
                        {% if snippet_content contains "Could not find asset" %}
                          {% comment %} do nothing {% endcomment %}
                        {% else %}
                          {% render 'your-snippet-name' %}
                        {% endif %}
```

The Liquid templating language is a sophisticated collection of document modeling and document processing resources. When a static site generator reads a Markdown document with any of these Liquid instructions, it *first* processes the Liquid command and then performs formatting. Note that Liquid instructions are derived from software programming standards.

Single sourcing

Docs-as-Code teams supplement basic Markdown code with processing instructions from an HTML template language such as [Liquid](#), [Jinja](#), or [Django](#).

IBM [Markedit](#)

Extensions

```
{:step: data-tutorial-type='step'}
{:shortdesc: .shortdesc}
{:new_window: target="_blank"}
{:codeblock: .codeblock}
{:screen: .screen}
{:tip: .tip}
{:pre: .pre}

# Apply end to end security to a cloud application
{: #cloud-e2e-security}
{: toc-content-type="tutorial"}
{: toc-services="containers, cloud-object-storage,
activity-tracker, Registry, secrets-manager, appid,
Cloudant, key-protect, log-analysis"}
{: toc-completion-time="2h"}
```

Google

Confidential + Proprietary

Occasionally you'll see Markdown and templating markup that reflect an XML DITA ancestry. I recommend looking at [Markedit](#).

Single sourcing

Theoretically – you *could* write optimized Markdown for static site generators and then dumb it all down for DITA-OT.



Google

Confidential + Proprietary

Would it possible to have a repo full of optimized Markdown documents that you filtered for Jekyll on Mondays and then filtered for DITA-OT on Tuesdays? Theoretically – yes, Practically – no.

Questions?



Stan@Modularwriting.com
StanDoherty@google.com