



# FM/WW TO DITA/FLARE

A Case Study presented by Mysti Berry, [salesforce.com](https://salesforce.com)

# Surviving Conversion FM to XML



Environment & Skills

Why XML/DITA and Flare

How We Did It

What We Learned



# ENVIRONMENT & SKILLS

# Environment & Skills

- ❖ SaAS Company
  - ❖ Six senior technical writers, including manager
  - ❖ One writer 50 -75% on build issues – not a programmer, but very close
  - ❖ No dedicated development resources
  - ❖ Already converted online help, so writers familiar with tagging, topic-based writing.
  - ❖ Everyone winced when “conversion” mentioned.
- ❖ Lots of re-use at word, phrase, sentence, paragraph, and topic level.
  - ❖ Translate help into 13+ languages
  - ❖ Fast moving company
  - ❖ Agile Development (see above)
  - ❖ Serious quality procedures in place for documentation
  - ❖ No CMS investment, just Dita, OpenToolkit, perforce for source control.



Search:

Go!

[Search tips](#)[Contents](#) | [Index](#) | [FAQs](#) | [Glossary](#) | [Tips & User Guides](#) | [Printable User Guide](#)**Getting Started**[Signing Up](#)[Logging In](#)[Using the Application](#)[Entering Data](#)[Accessing Data](#)[Navigating](#)[Using Help & Training](#)[What's New](#)**Setup**[Personal Setup](#)[Administration Setup](#)[Security and Sharing](#)[Data Management](#)[Customize](#)[Create](#)[Develop](#)

## Using the Help & Training Window

All information in the online help applies to **All Editions**, unless otherwise noted.

The Salesforce Help & Training window offers the resources you need to be successful.

### Find Answers to Your Questions

Click **Help & Training** at the top of any page. Enter your keywords in the Search box and click **Go!**. The search returns online help topics, knowledge base solutions, and recommended training classes that match the keywords you enter.

#### Tips for searching within the Help & Training window:

Consider these tips when searching:

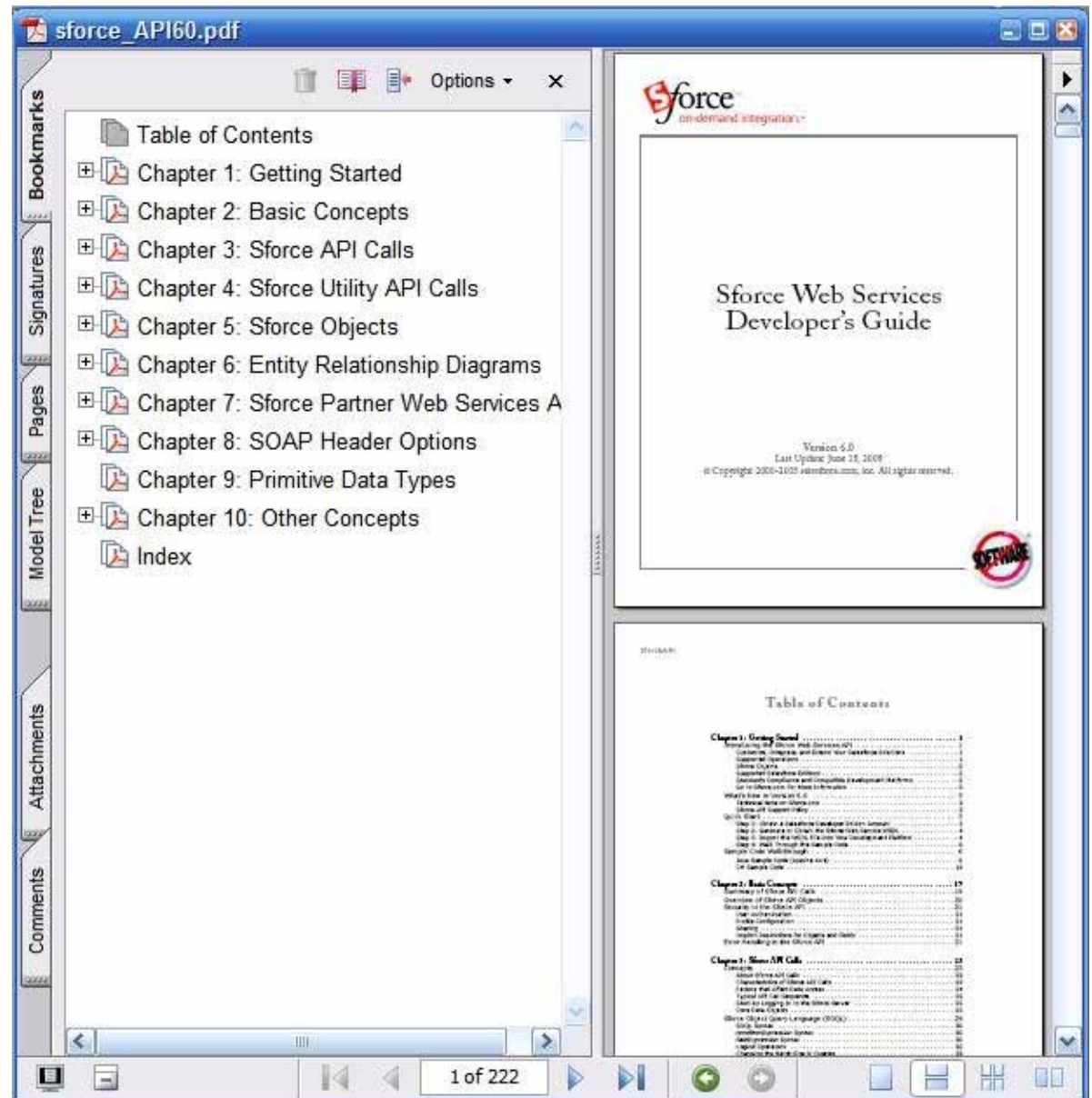
- Search returns online help topics, knowledge base solutions, and recommended training classes that include all or any of your keywords. For example, searching for *synchronize Outlook* may return results with just *synchronize*, just *Outlook*, or both keywords in any order. Items that include more of your keywords are higher in results, and items that contain your keywords as a phrase or near phrase are also high in results.

# The Conversion Project

Take FrameMaker files for a two-hundred page Developer's Guide with two conditional text flavors, convert to:

-XML source

- PDF, WebHelp output that lives on a standalone web server (and CHM, but more about that later!)





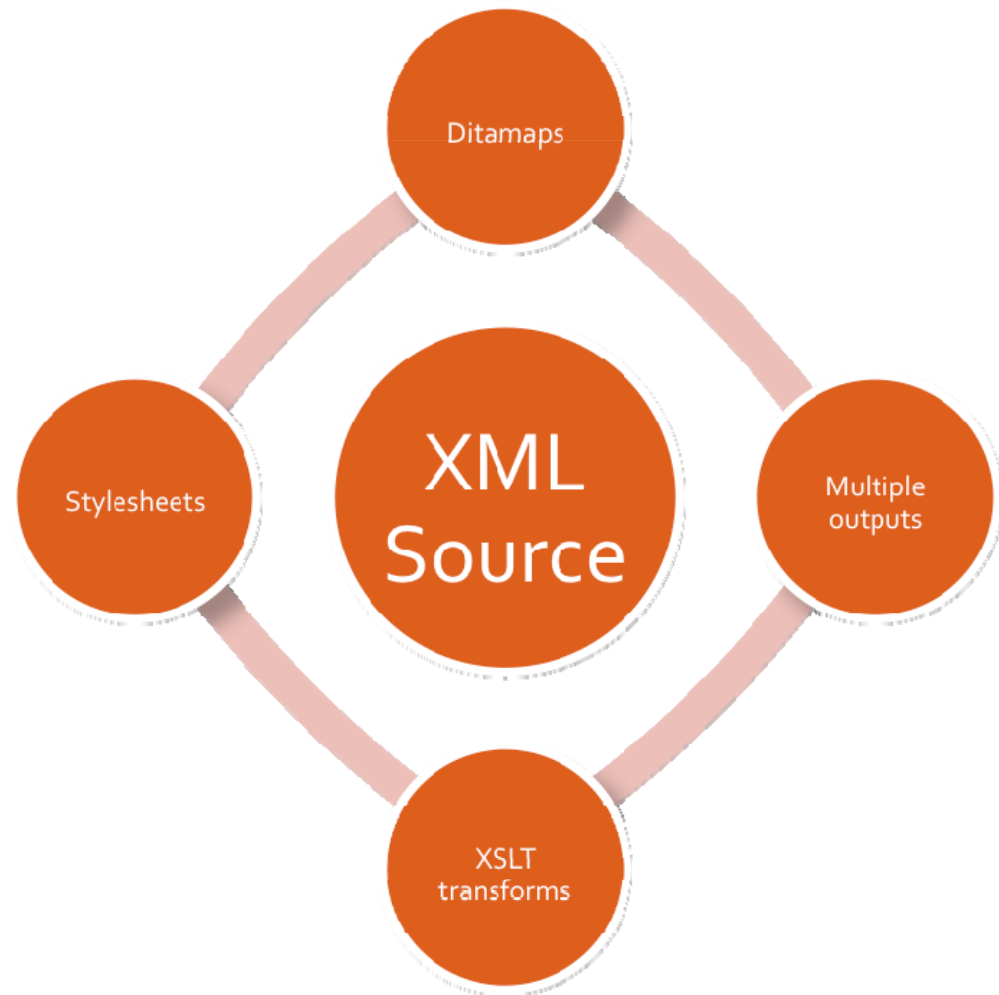
# WHY XML DITA AND FLARE?

# Why XML & DITA?

Company had invested a lot of time and energy into DITA/Open Toolkit with locally-created and maintained customizations.

Needed to have smart links between help and developer guides (mostly for re-use of titles, like hyperlinks in FrameMaker).

FrameMaker 8 didn't exist yet, but even if it did, bias in favor of open standards.



# XML Source

Help in  
App,  
WebHelp

Context-  
Sensitive  
Help

Translation

Re-use

# Why Flare?

We KNEW we didn't want RoboHelp or WebWorks one second longer.

We didn't want to reinvent the wheel regarding Index, Table of Contents, Search.

We researched, and narrowed it down to Eclipse help, Flare, and a third party none of us can remember two years later.

- Eclipse help required Tomcat at that time, and we had no control over what ran on our web servers.
- Yay, Flare! We created a template project file, and our build would grab it, add output HTML files: Presto!

# More details....

A map file lists the contents of a chapter:

```
<topicref
href="sforce_api_objects
_list.xml">
<topicref
href="sforce_api_objects
_account.xml">
</topicref>
<topicref
href="sforce_api_objects
_accountcontactrole.xml"
>
</topicref>
```

An XML source file holds the content, and we use `ant` on build targets to create the output

The build takes parameter info from the map file and fills it in for PDF and for certain values in the Flare template project, things like Title.

We vary styles depending on type of book—reference, online help, printed books.

We share content across all.



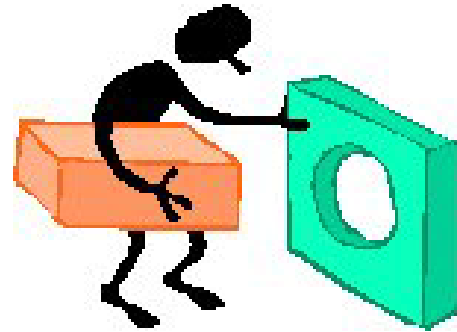
HOW WE DID IT...

# Brainiac and Ms. Gump

Our resident brainiac, Steven Anderson, figured out how to write XSLT transformations to turn MIF into XML that conformed to our set of XML tags.

Sort of...

A grunt (me, new to the job) had to go through the transformed xml, looking for systematic errors. These errors could be fixed by revising the XSLT transformation and rerunning ...except...



# Conditional Text is HARD!

MIF files don't always write their own condition tags in a systematic manner, believe it or not. So unless the writer applied conditional text tags in FrameMaker in just the right way, the converted files would have overlapping info that FrameMaker could interpret, but Steve's XSLT transformation scripts couldn't. He says the culprit was overlapping closing tags (close caret!)

We ran a lot of revisions, but at a certain point, we had to just go in and fix things by hand.

It took me nearly two months.



# || I made a LOT of mistakes 😊

1. We edited in the WYSIWIG tool, Epic, instead of using Oxygen more often to clean files quickly.
2. We didn't test enough on a small doc with only a few topics.
3. We should have done more conversion of narrative-style frame files into atomic single-topic files. Even though you can nest topic in topic, it makes those files hard to change around later because of links....
4. I started adding new content to some sections of the book before we had “sanctified” the conversion, making it hard for others to do sanity checks for hunks of missing stuff (I don't THINK we lost anything).
5. Iterative changes meant sometimes unexpected things broke—instead of insisting on nightly builds, I let resentment simmer.

# ■ We did some smart things, too.

Created profile definitions, so the conditional text displayed in different colors in Epic.

Wrote smaller and smaller topics and pulled files apart – lots of links to re-do, but made doc easier to revise.

Flare was, and is, the easiest part (that's the part I did!). We haven't touched the template project in over a year. A few bugs, but that was an early version.

Abandoned CHM output. Awful problems with either Flare, MS, or perhaps the combination. Unacceptable quality output.

(post-conversion, sad to say)  
Wrote smaller and smaller topics and pulled files apart – lots of links to re-do, but made doc easier to revise.

<http://www.salesforce.com/apidoc>

# How Did It Turn Out?

Force.com Web Services API Developer's Guide

http://www.salesforce.com/us/developer/docs/api/index

Google

Getting Started Latest Headlines

Contents

salesforce.com  
Success On Demand

- Getting Started
  - Introducing the Force.com API**
    - Customize, Integrate, and Extend Your Sales
    - Supported Salesforce Editions
    - Standards Compliance
    - Development Platforms
    - API Support Policy
    - What's New in Version 12.0
    - Related Resources
  - Quick Start
  - Standard and Custom Object Basics
  - Call Basics
  - Error Handling
  - Security
  - Using the Partner WSDL
- Reference
- Using the API with Salesforce Features

Force.com Web Services API Developer's Guide

Version 12.0

**SOFTWARE**

© Copyright 2000-2008 salesforce.com, Inc. All rights reserved.  
Various trademarks held by their respective owners.

Done



# WHAT WE LEARNED

# Prepare in FrameMaker First

Get your files in parallel structure, if not totally topic-based. That way you can consistently break up a big FM file into little topic files in XML.

Scrub your FM files for junk (extra spaces, one-off formatting changes, etc).

It'll take 4 times as long as you think.

Choose naming conventions for xml files, link IDs. We didn't convert numerical FM IDs, now we all wish we had...

Identify commonly used terms (brand names, all that stuff you've got in variables in FrameMaker now), and put them all in a single "reusable.xml" file. Then pull references in from that file.

You won't believe the time it saves when they change the product name just before release!

# Test Test Test

Test for systematic errors on a sample set of files first.

XSLT lends itself to unpredictable changes, so test all targets before checking in build changes.

Chek with your translation team – variables in XML can be hard to translate. See if you can do some transformations for them before they tackle translation.

It's well worth it.

Little shops can try with DocBooks instead of Dita/Open Toolkit. It takes a lot of time to build and maintain a Dita system.