Migrating to MadCap Flare

Adrian Morse describes how his company left the familiar waters of Adobe products for the open seas of Flare.

The date had arrived. After countless group discussions, numerous trials and an ongoing effort to separate the facts from the hype, we decided to replace Adobe's FrameMaker with MadCap's Flare. Actually, it was more like replacing 'FrameMaker plus Acrobat plus Mif2go plus a little bit of RoboHelp' with MadCap Flare. FrameMaker had long been our source. From there we had been kicking out PDF files via Acrobat and HTML Help files via Mif2go. It was as close to single-sourcing as you could get: a few clicks and we had the outputs, with no post-editing whatsoever. The help system for one of our products needed to be in WebHelp format; we already had some RoboHelp (for Word) licences so decided to just use it as a backend converter, taking Mif2go's HTML output and converting it to WebHelp format. It all seemed to work well…

If it ain't broke…

Our company had been through various mergers, each bringing its own styles, formats and functionality along for the ride. Our product suite had also expanded from client/server to browser-based applications. Individually, the documents and help systems were straightforward to create and they did the job. Globally, however, it was a mess; it looked as if we were still separate companies. Something had to be done.

Our first step in rectifying the situation was to decide on a common format for the help systems, and we chose WebHelp. Besides it being the only real contender for the browser-based products, our experience was that users felt more comfortable navigating such systems.

Next we had to decide on the tool. Having experienced occasional problems creating new projects with Mif2go, we decided to look at alternatives rather than use it together with RoboHelp (for HTML to WebHelp conversion).

At this stage it would have been natural to evaluate Adobe's Technical Communication Suite, but our technical communicators had had so many negative experiences with previous attempts to integrate FrameMaker and RoboHelp that (rightly or wrongly) the idea was quickly dismissed. By contrast, there was some zeal for MadCap Flare which eventually won us over. This article describes how we took the plunge and migrated to it.

Decisions, decisions …

We soon found out just how immense Flare is. Coffee-room discussions overflowed with terms like 'snippets' and 'togglers'. Team members involved with creating templates began to look rather haggard. Others were waiting for guidance. It became clear that the strength that was the product's versatility was also a weakness for those new to it.

So what if it's XML based?

MadCap is keen to tell us that Flare is 'XML based', so that must be a good thing, right? Well, yes, but…

As far as our final PDF and help files were concerned, it did seem that Flare could not really 'do' anything that our previous combination of tools could not; it just achieved the same end result by different means. For example, *applying a format to a paragraph* became *applying a tag to an XML element*, *opening the table catalog* became *opening the table style sheet.*

For XML or DITA outputs, there are likely advantages to authoring within XML itself but for us this did not seem to make a difference. Sure, we could look at the XML behind a topic but we only really had cause to do so when trying to distinguish a bug from 'by design' behaviour.

Perhaps one exception to this observation is the ability to *easily* apply DIV tags across multiple paragraphs in a topic. As well as being needed for dynamic HTML effects, DIV tags offer a way to control the look of sections rather than individual paragraphs or entire topics. For example, you can easily apply a coloured background to selected text (Figure 1).

What's up doc?

The first time we opened a Flare project we were hit by an 'okay, so now what do I do?' feeling. In fact, for the first few weeks, it happened pretty much every time we opened a project. However, the situation improved after we had assimilated the following:

1. We were accustomed to seeing a FrameMaker 'book', in which the main sections of the document or help system were clearly visible. It took a while to realise that the Flare TOC was now the first port of call. It is a little irritating that the TOC does not open by default but this can be mitigated to some extent by configuring Flare to reload open files when you open a project.
2. We were accustomed to working with a dozen or so filenames in a logical order. In contrast, our Flare projects typically had well over 100 files ordered alphabetically with no indication of where they slotted into

the document… welcome to topic-based authoring! The granularity of files was driven by the look and feel we wanted in the help system. MadCap's suggestion of 'make every topic self-contained' sounds like good advice and would certainly have reduced the file count, but it did not really work for us. Anyway, despite the paradigm shift in the way of working, a few months down the line and the technical communicators didn't really bat an eyelid anymore. They just made sure to link any new file to the TOC as soon as it was created.

Drag-and-flop

Getting to grips with Flare was made harder by minor annoyances in the interface. I use the word *minor* because the issues we hit had easy workarounds. Here are a handful of quirks to expect:

- Drag and drop can sometimes be a little tricky. If you select a whole line of text but do not include the carriage-return symbol you can drag the selection and drop it elsewhere, such as within another paragraph. However, the carriage-return symbol sometimes gets selected automatically right at the moment that you attempt to drag the selection. Flare then decides that you are trying to drop one XML element inside another, which is prohibited and so the 'drop' fails. Nothing that **Ctrl-X** and **Ctrl-C** won't sort out though.
- To insert a paragraph above a heading you put the cursor in the heading, press **Home** to move the cursor to the start of the line and then press **Enter**, right? Wrong! If you do this, the heading's tag changes to a **<p>** tag and you have to reapply the heading. This annoying behaviour only seems to happen with headings. For example, if you use this method to add a new paragraph above a bulleted item, the bullets are retained. Fortunately, there is an alternative: after pressing Home, you can press ↑ (which makes the cursor horizontal above the heading) before you press Enter.
- You can copy the contents of one table cell and paste them into another, but you cannot paste the contents of a single cell into multiple cells at once.

Spoilt for choice

Getting started with Flare can be quite daunting due to the proliferation of options at every stage. Here I describe some of the key choices we found ourselves making. They are presented in order of importance. This information by no means covers all functionality available in Flare, just the main areas where decisions are needed.

Source?

Our first big decision was whether to author entirely within Flare or continue with FrameMaker and just import into Flare in order to create the help output. Our decision was primarily influenced by two factors:

- We wanted to come as close to a 100% single-sourcing workflow as we could.
- We wanted to be able to take advantage of Flare's great features for adding dynamic HTML effects (such as drop-down text or popups).

There then followed the usual group discussions about what 100% single sourcing means. We concluded that it meant the ability to make changes in the source and create immediate outputs without any post-editing of the outputs being needed. This ruled out authoring within FrameMaker since dynamic HTML effects added in Flare after importing from FrameMaker would need to be recreated if the source was later changed and reimported.

PDF output from Flare… are you serious?

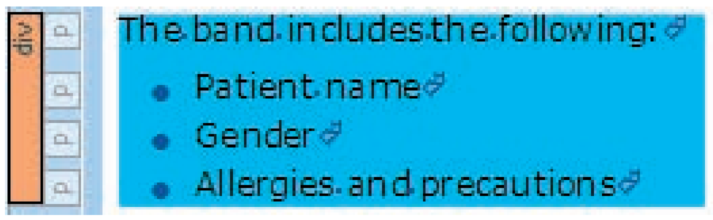
Flare v3 did not have support for directly creating PDF files, so we initially output to FrameMaker and then generated PDF files from there.

When support for PDF was added in version 4, we were very sceptical. MadCap were new players in the field and we could not see how they could compete with the very company that created PDF technology. We needn't have been concerned. PDF files from Flare looked great. A month down the line and we had completely recreated our templates, bypassing FrameMaker and Acrobat completely.

Lists

Out of the box, Flare lets you apply bullets and numbered lists to text using the drop-down control on the toolbar. In the background, **<ul>** and **<ol>** tags are applied. Our next decision was whether to use these innate list styles or create our own auto-numbered styles (just as in FrameMaker).

You can get a long way with the innate tags but they have their limitations. For example, they cannot show chapter, section or volume numbering and they cannot be used with custom bullets. You also need to restart new lists manually and there is no inherent 'control' over what type of bullets or numbers each technical author on a team uses. Autonumbered



The band includes the following:
- Patient name
- Gender
- Allergies and precautions

Figure 1. A DIV tag class applied to text. In the CSS, this class is configured to give a blue background.

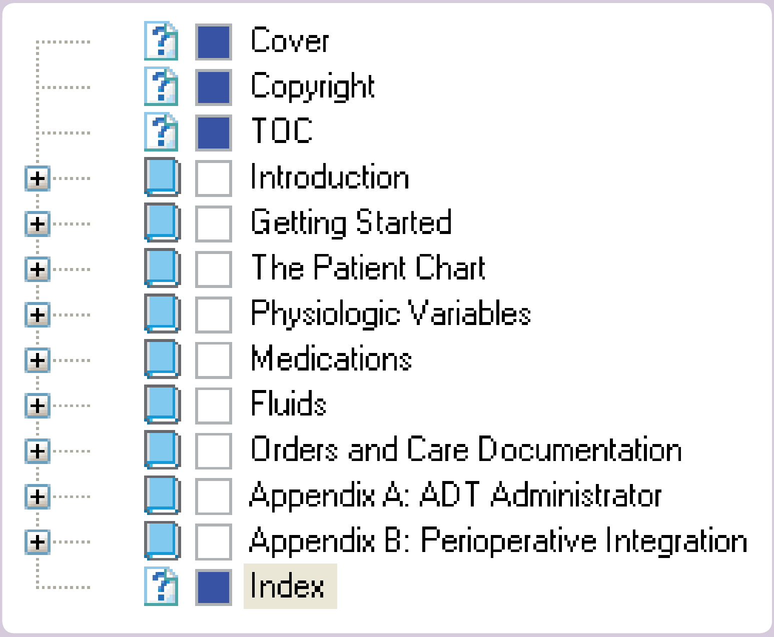

Cover
Copyright
TOC
Introduction
Getting Started
The Patient Chart
Physiologic Variables
Medications
Fluids
Orders and Care Documentation
Appendix A: ADT Administrator
Appendix B: Perioperative Integration
Index

Figure 2. A 'project TOC' with 'print only' conditions applied to the cover and copyright page, TOC and index

…the whole program is oriented around the concept of a single stylesheet

styles do not suffer from these deficiencies, so we decided to use them instead.

At this stage, we could have created the autonumbered styles as CSS classes for the ordered list tag. For example, **ol.numbers** (where *ol* is the tag and *numbers* is the class). Instead, we chose to create them as classes for the **<p>** tag (that is, **p.numbers**). Why? Well, some windows and controls in Flare only show classes that are pertinent to the selected style. For example, suppose you are editing a list item and want to change its style to a certain paragraph class. You are not able to select the paragraph class from a drop-down list of styles because the list will only show classes appropriate to list tags (such as **li.bullet**). So you first have to remove the list tag, adding extra steps to the operation. On the other hand, if you stick to paragraph classes for all styles, you will always have both list and non-list styles within easy reach. There is nothing about list classes that you cannot achieve through a suitably configured paragraph class.

The project TOC

It is useful to distinguish a 'project TOC' from a 'print TOC'. A project TOC determines the topics that will be included in the output, their order and hierarchy. For help outputs, the project TOC corresponds with the actual TOC seen in the resulting help system. By contrast, in print outputs it is the print TOC that appears rather than the project TOC.

A project can have one or more project TOCs. When you build an output you define the project TOC to use. For printed outputs, the project TOC needs an entry for a cover page, a print TOC 'proxy' (a placeholder for the print TOC) and an index as well as entries for topics (Figure 2).

Decision time again. Should we create the help and PDF outputs from two different project TOCs? Or should we stick with one project TOC but use 'conditions' to hide any print-related entries when creating a help output? (For instance, the condition 'print only' could be applied to the cover page entry to exclude it from the WebHelp.)

As far as the end product was concerned both strategies were the same. However, taking maintainability into account, we decided to use a single TOC with conditions. In our case, most content updates were likely to affect both print and help outputs. Having one TOC meant that we would only have to link the TOC to a new topic once. Using two TOCs would have meant linking each new topic twice and there was a risk of inconsistency between the TOCs.

The print TOC

There are two ways to create a print TOC:

- Replicate the structure seen in the associated project TOC
- Define the styles that you want to be TOC entries (for example, all H1s and H2s).

Basing the print TOC on the project TOC has the advantage that you can instantly see how your TOC will look. However, we decided to go for the option of using styles because it effectively encourages consistent formatting for the same heading level. For example, if you want a heading to appear at the usual H3 level, you must apply a H3 style to it.

Table styles

You can apply styles to tables either by using *table stylesheets* or by applying **<table>**, **<td>**, **<tr>** and **<th>** tags or classes from the usual project stylesheet (there can be issues if you try to mix the two).

We opted for table stylesheets for the following reasons:

- It's easier to create tables with them.
- You can tab to add a new row.
- It's easier to set up alternating row formats.
- They avoid a bug that occurs when 'undoing' a **<tr>** style application.

That said, table tags or classes can be needed for complex tables so perhaps a common-sense approach would be to set up table stylesheets initially and only create table classes if a particular situation calls for them.

Then there are indented tables that you might need for lists or other situations. Again, you have a choice to make: either create an indented table style or use an existing table style but edit the left margin after applying it. We decided to create specific styles because margin edits are not retained if a table style is reapplied or if an additional table style for print output is later applied to the table.

Localising WebHelp skins

The method for localising a WebHelp 'skin' depends on whether the English skin uses default text or has been customised. If it uses default text, you just need to substitute the skin provided for the appropriate language. However, if your English skin has custom text (for example, if you change the tooltip for a button on your WebHelp toolbar) and you want the tooltip to be localised too, another approach is needed. You have these options:

- Customise each language skin to match the English skin changes. One difficulty here is that language skins are stored under the 'Documents and Settings' folder for each user (that is, outside the Flare project folder). So you have to remember to include this file when sending the project for localisation and the localisation agency needs to make sure they place the file in the expected path for each user.
- Forget language skins and just use a a copied and renamed version of your English skin. If you do this, you must remember to change the skin link in the target for each localised project.

Stylesheet policy

Flare lets you use the same stylesheet for different outputs or specify a different stylesheet for each output. This one was a no-brainer: the whole program is oriented around the concept of a single stylesheet with 'print', 'non-print' and 'default' media sections. You link each output to the medium you want. (For example, you would link a PDF output to the 'print' medium.) When using a single stylesheet in this way, Flare enables you to see easily how a topic is going to appear for each output and the effort needed to maintain styles is reduced.

We found it best to avoid using the 'non-print' medium and instead define all styles in the 'default' medium section of the stylesheet first, with the 'print' media section just being used for any changes from the default. This ensures that all styles are available regardless of the media set for viewing and it also makes it easier to configure new styles.

We also found it to be a good idea not to set a default font size for the **<p>** tag but instead set a default font size for the **<body>** and **<td>** tags. It meant that we could set our **<p>** styles to drop down a font size automatically when used inside tables, avoiding us having to create a whole set of styles for use in tables. For example, our **p.Bullets** style normally appears with a font size of 10pt, but when used in a table it automatically becomes 9pt.

Policy for resizing screenshots

As well as Flare, we also purchased MadCap Capture for working with screenshots. Flare alone offers various ways to resize screenshots: throw Capture into the mix and you have even more options, which can get a little confusing. We needed a policy.

We first spent time testing the quality of images that we had reduced in size and noted that some resizing methods worked better than others. We looked at both PDF and help outputs. Our expectations for the quality of resized images in PDFs were high but we did not expect to see high-quality resized images in help output, since any reduction necessitates a loss of pixels. However, we were pleasantly surprised. With our previous workflow, we could only achieve acceptable results by making sure that any screenshots in the HTML output were recreated at their original size. They were crisp and clear on screen, but the larger screenshots could be overbearing. With Flare, this wasn't necessary. As long as the option **Generate copies of resized images** was selected we could reduce screenshots down to about 65% of their size and still find them acceptable in the help file. As Flare does not claim to be a graphic editing tool, we found this remarkable. It opened the door to using the same image dimensions for both PDF and help outputs for the majority of images. Doing this wasn't strictly necessary, as Flare allows you to specify separate dimensions for online and print output, but it certainly made things easier.

As for how to resize, the following methods are available in Flare:

- Dragging an image's borders
- Setting dimensions for an image
- Setting maximum and minimum image dimensions for all images in the project using the stylesheet.

We found that all methods produced good results. However, you cannot size an image by a scale factor or DPI in Flare; if you want to do that, you have to calculate and enter the dimensions yourself. Or you can use Capture. We observed that the **Print DPI** setting in Capture worked well but the **Print Size** setting (which one would assume should be reciprocally linked to the **Print DPI** setting) did not; it gave a low-quality image. In Flare, as I mentioned previously, we could set the same dimensions for online and print outputs in one place and get good quality outputs. In Capture, this did not work: the quality of the online image was as expected but images in the PDF output were of poor quality. Print settings always had to be specifically configured in order to get a good-quality image.

Because of these issues, we decided to do all resizing in Flare rather than Capture. We maintain a consistent scale factor by calculating the image dimensions we need manually. A handful of large screenshots do not look good in the PDF output when scaled by the factor needed for the online help. To cater for such situations we set a maximum print image width of six inches in the stylesheet.

I should add that MadCap has made ongoing improvements and changes in the handling of

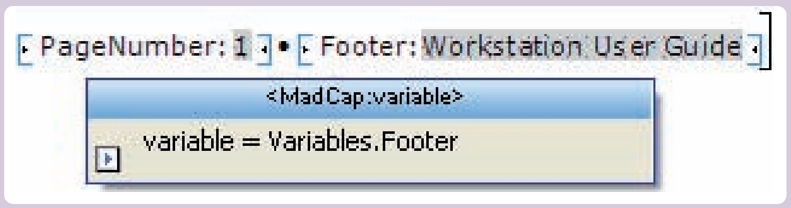

PageNumber: 1 • Footer: Workstation User Guide
<MadCap:variable>
variable = Variables.Footer

Figure 3. A variable used in the footer of a page layout.

images. After each new release, we run tests again and revise our policy as necessary.

Use of variables

The big decisions were behind us. Most choices ahead were typical of any tool, not just Flare, so I will just comment on a few of them.

First are variables (Figure 3). To make maintenance and localisation easier, we used variables for headers rather than direct text in the 'page layouts' (equivalent to master pages in FrameMaker). However, we decided against using variables for text on cover pages because you cannot manually control line breaking within a variable and this could have led to localisation issues in languages such as German, where the text length would be expected to increase significantly.

Compared with FrameMaker, maintaining styles in Flare projects is a walk in the park

Use of conditions

Conditions can be applied to individual letters, words, sentences or entire paragraphs. A policy was needed to ensure consistent usage. The localisation agency asked that we apply conditions only to entire sentences or paragraphs, not to individual words (with the exception of table headings). We get some repetition of source content this way, but it is easier to work with.

We also decided that conditioned paragraphs should include the paragraph mark at the end. (If it is included in some cases but not others, outputs might have sentences running into each other or empty paragraphs.) The important thing here was consistency; a policy of leaving out the paragraph mark would have been equally valid.

Adrian Morse is Documentation Manager at Picis, a US-based provider of information solutions for delivery of clinical, financial and operational results in hospitals. He is responsible for documentation and help files, and for the people who work on them. His background is scientific: he holds a PhD in Applied Physics and worked as an electrical engineer and later as a lecturer at Manchester University before becoming a technical communicator in 1997. He is an advanced user of FrameMaker and Flare, and has also been involved in numerous localisation projects (he is fluent in Spanish).
E: Adrian_Morse@picis.com
W: www.linkedin.com/in/adrianmorse

FrameMaker features we miss

Automatic markup

Start typing in FrameMaker and you can see a bar in the margin next to any text that has changed. Change bars are visible in the PDF output; this enables us to use an Acrobat shared review process, which we need for regulatory purposes and which reviewers seem to love.

Flare, on the other hand, offers the technical communicator a way to see the differences between a topic and an earlier backup of it but reviewers cannot see this view. As a workaround, we apply highlighting manually to mark text that has changed before generating the PDF file. This is laborious and error-prone.

Find/replace capabilities

Find and replace in Flare can be confusing as there are two similar windows for this purpose. Searching in one window creates a list of topics that contain the search term; searching in the other opens the next topic containing an occurrence of the search term.

Besides the comparative simplicity of FrameMaker's Find/Change window functionality there is also a noticeable difference in search options. Flare does not offer searches such as the following that can be found in FrameMaker:

- Character tags (spans in Flare)
- Paragraph tags (tags and classes in Flare)
- Character formats (such as italic and bold together)
- Any cross-reference
- Cross-reference of format
- Unresolved cross-reference
- Any table
- Table tag (a particular table style in Flare)
- Conditional text
- Anchored frame (just images in Flare).

You can search for some of these entities in Flare by searching in the source code for the tags you expect but this is undocumented and rather risky, especially for 'replace' operations. For some searches across topic files, we therefore find ourselves resorting to FrontPage.

It should be said that you can generate certain reports including the following:

- A list of topics that contain images
- For each image, a list of topics in which it is used
- A list of topics in which a condition is applied to text (but with no indication of what the text or the condition is — this could be a bug)
- For each variable, a list of topics (and templates) in which the variable is used.

However, these reports only list file names and do not provide the context in which the entity is found. That wouldn't be so bad if the entries were hyperlinked but they are not (for that you need the MadCap Analyzer add-in).

Ease of inserting rows and columns in tables

Inserting a single row or column in Flare is easy enough but you cannot insert multiple rows or columns from the right-click menu. To do this, you need to use the Table Properties window and you do not have control over where the rows or columns are placed.

Flare features we couldn't do without

Here I will discuss some features that did not exist in our previous Frame plus Mif2go setup.

Global Project Linking

Compared with FrameMaker, maintaining styles in Flare projects is a walk in the park. No risk of users selecting the wrong format import options. No obsolete styles hanging around afterwards. No need for FrameScript add-ins to help you out. Yes, definitely a walk in the park.

To make that walk even more pleasant, Flare offers a feature called Global Project Linking. This enables you to have templates propagated to all Flare projects in your workplace from a central location. You can also use Global Project Linking for sharing topic files between projects automatically… as soon as Mary updates the 'Basic Computing Skills' topic, Dave and Alan see it updated in their user guide projects! Cool.

Ease of adding dynamic HTML effects

It was possible to add dynamic HTML effects (such as expanding text and popups) using our previous workflow of FrameMaker plus Mif2go. However, it meant adding raw HTML code to markers inside FrameMaker, so wasn't the most user-friendly of methods. In contrast, Flare has great functionality for adding such features. We particularly like 'togglers', a feature that lets you show and hide entire sections within your topic (and they don't even have to be contiguous).

Integration with Capture

Whenever you create or open a graphic file in Capture it creates a properties (.props) file in the same folder. If you add callouts to an image they get added directly to the image file (for example, to the PNG file). However, the props file effectively contains the original graphic plus all the changes that have been made. So, when you open an image file in Capture any callout that you added is an editable layer rather than a merged and uneditable part of the graphic.

This is a clever design. It means that to edit callouts in a graphic that is part of your Flare project you just need to right-click the graphic in the Flare topic, select **Edit with MadCap Capture**, edit the callouts and save the file. In contrast, if you want to edit the callouts in another graphics editor, you have to maintain a separate 'master' version of the file for future editing and save any updated image file into the Flare project.

The support

A discussion of MadCap Flare would not be complete without reference to the fantastic support available for it. You can easily submit bug reports and enhancement requests through the MadCap website. This is not like sending information into a black hole; unlike some other vendors, MadCap goes to great lengths to listen to its customers and act on suggestions.

Besides MadCap's own support system there is a vibrant and extremely helpful user community at http://forums.madcapsoftware.com/index.php. Common words (such as CSS) are ignored in forum searches but we found that you can work around this by searching with Google (use www.google.com/advanced_search?hl=en and enter 'forums.madcapsoftware.com' in the **Search within a site or domain field**).

Conclusion

Although we have made a few sacrifices along the way (notably the loss of automatic markup), our migration has gone well. Setting up templates took longer than we expected due to the numerous and often interconnected decisions needed. The steepness of the learning curve was also a bit of a shock. However, the technical communicators in the group are now quite confident with Flare and can quickly turn out new projects. What's more, with regular product updates and MadCap's engagement with the customer, we really feel we have invested in a product with great potential. We look forward to what the future brings.

A parting word of advice to potential users? Flare is an ocean of possibilities… focus on replicating the familiar territory of your existing help system before sailing off into uncharted waters! C