Creating Free Software Magazine

A long path that takes us to the very beginning of this project

Tony Mobily

This magazine was inspired by a conversation I had with a great friend of mine called Massimo. I said to Massimo "I think it would be great to start a magazine. It's my ideal job, and I think I know what the world needs right now. It's a pity there's no money in publishing, and I'm not willing to run a magazine that doesn't pay it's contributors *well*...". His answer was very simple: "Tony, there's money everywhere, as long as you do something good and promote it well". Well, seeing that he has a successful business, I thought I would listen. And it's thanks to him that you are reading this article right now.
A few months ago, just before my conversation with Massimo, I realised that the world needed a magazine on free software. My ideal magazine would be aimed at IT professionals (the *techs*) as well as managers.

Massimo said "Tony, there's money everywhere, as long as you do something good and promote it well" Well, seeing that he has a successful business, I thought I would listen

The magazine would contain technical articles, but they would be focused on describing the possibilities of free software, rather than the technical details of how to configure a specific server. It would also publish technical articles on software patents, copyright laws, and how the world is changing thanks to free software. Above all, it wouldn't be yet another technical Linux magazine (there are plenty of them at the moment) and it wouldn't compete with Linux magazines. It would also break the common rules for magazines of this kind, and contain a fiction section - short stories about the new technological era.
This magazine would pay its authors well, and would release all the articles under the GNU Free Documentation License.
A few months ago, after talking to Massimo, I decided that I would do it - and I have.

Starting up

When starting a magazine from scratch, the first concern is creating an appropriate structure to support the project (an office, the staff for the magazine's composition, the managing editor, the web designer, etc).
I have been working on magazines my whole life. I am aware of all the processes involved, and I know that if I had followed the "normal" path, I would have needed a lot of capital to start the project, and a lot of advertisers and subscribers to keep it going. Seeing as Free Software Magazine would attract a restricted audience (we are not Cosmopolitan or Playboy...), such a structure would have been far too expensive.
There is also the technological side of the story. I wasn't willing to accept that Free Software Magazine (I will call it "FSM" from now on) would need a compositor to actually make up the magazine *by hand* for every issue. The manual composition of a magazine and the subsequent quality checks take phenomenal amounts of time and money (for a while I was the man who checked that the all the captions were correct, all the borders aligned, and so on for another magazine). Also, I wasn't willing to accept that in order to create FSM I would need proprietary software (I discovered later that I'd have to give in on this one, but only marginally and temporarily).
A magazine that talks about free software and solutions had to be set up in such a way that composition simply wouldn't be needed: the magazine's PDF and HTML versions would have to be generated *automatically*, providing the articles as input.

Well, I can now say (and not without immense satisfaction, and a sigh of relief): we did it.

The system we created can now take an article written using OpenOffice or Microsoft Word (using the right styles), and generate the XML version - as well as an extremely professional looking PDF file.

It has been an up-hill struggle. I have designed the whole system, coordinated the amazing people who wrote various components, and coded a great deal of it myself. It has been hard. I have sent and received more than two thousand emails for this project. But in the end, we did it. The system is here, and we are now using it for the real "Issue 0" that I am writing right now.

The initial technical planning: XML

A magazine is a collection of articles. Deciding the format for the articles was, in my opinion, one of the most crucial steps of the project - everything else would depend on it. Getting it wrong could have compromised the project's success.

The choice went automatically to XML: it's a language that allows you to define your own file format; it has existing, powerful tools (such as XSLT and XPATH) for converting data into HTML and other formats; it is supported by every platform on this planet; and (which is quite important as well) I had worked on it before, even though it had been a while.

But it's not enough to say, "I'll use XML files", there's still all the design work that needs to be done.

XML lets you decide what tags (or, more correctly, entities) you will use in your mark-up file. You normally do this by writing your DTDs (or more modern schemas). But this decision is an important one, as it's very easy to design an XML structure that simply doesn't work properly. What's worse, you may discover the problem further down the track, when changing a detail in the XML file generates a chain reaction that explodes into many, many hours of work.

This is why the first thing I had to decide was: do I want to use an existing, established DTD, or shall I define my own?

I did my research, which was crippled by my limited knowledge of XML; I looked into other systems that dealt with similar issues, but they all looked too complex or boring to me.

I wanted a simple, lightweight XML structure that would contain exactly what I needed - after all, if XML was a way to store information intelligently, who could decide what information to store better than me? I had worked in the industry long enough to know what information I needed for each article. So, I designed it.

I did my research, which was crippled by my limited knowledge of XML; I looked into other systems that dealt with similar issues, but they all looked too complex or boring to me.

Well, I did it with the help of Michael Eastwood, who has three fantastic qualities: he's a genius, he knows XML very well, and he's a graphic designer. Michael did a lot more than help me with XML: he designed the initial web site, and wrote the XSLT transformations to translate articles into HTML (Michael said from the beginning that it would be up to me to set the XML structure).

Here is what a very basic article looked like:

```
<?xml version="1.0" encoding="UTF-8"?>
<page>

  <title>
    <main>The title</main>
    <sub> The subtitle</sub>
  </title>

  <article_info>
     <publication_date
        day="31"
        month="12"
        year="2004" />

     <authors>
       <author>First Author</author>
       <author>Second Author</author>
     </authors>

     <article_type>Article type</article_type>
     <biography>First Author is this.
                Second Author is that.
     </biography>
   </article_info>

   <contents>

     <p>This is a sample article</p>
     <heading>Today's heading:</heading>
     <p>This is a simple paragraph
        under an heading</p>
```

```
   </contents>
</page>
```

The structure was very simple: everything is enclosed into a <page> tag. An article had a title (divided into main and sub), a publication date, an author, a type, an author's biography, and the text (which was made up of paragraphs).
This is what the resulting HTML looked like after the transformation (please remember that this is a *very* stripped down version of it):

```
<html
<head>

  <title>FSM - The title</title>
</head>

<body>

<p class="article_type">Article type</p>
<br>
<p class="date"> Date: 31/12/2004</p>

<h1>The title</h1>
<h2> The subtitle</h2>

<p class="author">By First Author, Second Author</p>

<p>This is a sample article</p>
<h3>Today's heading:</h3>
<p>This is a simple paragraph under an heading</p>
<h3> About the author </h3>

<p class="biography">First Author is this.
                     Second Author is that.</p>
</body>
</html>
```

I created a directory structure that would contain everything; I placed the HTML documents into `htdocs`, and decided that each directory would only contain one XML file called `index.xml` I also created the directory `bin`, where I placed the command `ov_make_html`. This command basically runs:

```
xmllint --xinclude $OV_PATH/xslt/page_html.xsl \
| xsltproc - index.xml  > index.html
```

The directory tree you can see in the downloads section of this article is a little more complicated (and so are the scripts), but it's easy to find your way once you understand how the whole system works.

Once all this work was done, I candidly asked Michael: "Why don't we use this fantastic system to create the *whole* web site?" Michael was sceptical. He said "you can if you want, but then you are limited: you won't be able to put anything fancy on the web site, only what's allowed in articles".
There is one thing I hate about Michael: he is always right. However, in that particular instance I was right too: using the same template for articles and web pages would simplify the web site's management to a great degree.
What was the difference between a web page and an article? Well, all (and *only*) the information within <article_info> was inappropriate for a web page (author, publication date, article type, and biography). Therefore, a web page would share the same structure as an article, but without the entity <article_info>. This is why both an article and a web page start with <page> (rather than something like <article> or <web_page>).
This was only the beginning. The hard part had yet to come.

Overcoming the limitations: exec filters

In the previous section, I mentioned that using the same converter for articles and for our web site made the latter look really boring.
One of the main technical decisions I made was that even if the site's HTML was going to be generated automatically, the web site itself had to be static. There are several reasons for such a (possibly limiting) choice. A static web site has the following characteristics:

- it can be hosted anywhere;
- it takes less processing power to serve a static pages;
- it is much, much less prone to DOS attacks, SQL injections, and so on;
- it is easier to maintain;
- it will never return an error message to the clients;
- it is much easier to have it mirrored without going insane with mod_rewrite.

So, FSM was going to be an automatically generated, static site.
This created a number of flexibility problems. For example, I wanted every web page to show its "path", so that people visiting it would never get lost while

browsing. I truly dislike web sites that have an unclear and illogical structure (and yes, this does mean that I dislike most web sites on this planet...).

But I wasn't happy to put the path information on the XML file itself: what if the file changed its position in the file system? If the PATH information were in the XML file, I would have to manually change the page's XML as well - unfortunately, I dislike duplication of information as much as illogically structured web sites...

I needed a way of embedding some pseudo-dynamic material in a web page. I also needed to prove Michael wrong, and show him that yes; it was possible to make the web site look great. The basic idea was that the result of a program would be embedded into the resulting HTML page; this would give me a great deal of flexibility.

I didn't find any information on how to do this using "standard" XML. Besides, I needed to be able to:

- run a program which would return XML code, which would then in turn be processed by the XSLT processor (a "pre-processor");
- run a program which would return HTML code (a "post-processor").

This is why I wrote the scripts `ov_pre_exec_filter` and `ov_post_exec_filter` (which are really both the same script, but it changes its behaviour according to its name), which I placed in the `bin` directory of the project.

I wanted to keep the web site "alive": FSM is a magazine, and not a news site; as a result, the only news is when a new issue comes out, and updating the web site only once a month can make it a little "static"

The script `ov_pre_exec_filter` is very simple: it looks for the XML entity `<exec-pre exec="PROGRAM"/>`; it then executes "PROGRAM" and substitutes the program's output with the entity itself. The `ov_post_exec_filter` does exactly the same thing, but it looks for the entity `<exec-post>` instead.

In practice, this would mean for example, that fixing the PATH problem (see above) simply required inserting this entity in the XML file: `<exec-post exec=''ov_exec_path''/>`.

The script `ov_exec_path` would take, as input, the path of the page it refers to, and would print out a complete clickable version of the path.

The exec filters system became much more crucial than I expected. It gave me an incredible amount of freedom, and allowed me to make the web site far less boring.

One fine example is the organisation of the "daily pills". I wanted to keep the web site "alive": FSM is a magazine, and not a news site; as a result, the only news is when a new issue comes out, and updating the web site only once a month can make it a little "static". For this reason, I wanted to have a system where I could publish "daily pills": short articles about absolutely *anything* (even fiction) that could possibly interest the magazine's audience.

This is what the "daily pills" page looks like:

```
<contents>
    <p> Every day (well, nearly every day)
    FSM publishes a <i>pill</i>,
    a short article on Open Source's
    "current affairs".  </p>

    <heading>Index by the year</heading>
     <exec-pre exec="ov_exec_pill_year"/>
</contents>
```

That's all! The script `ov_exec_pill_year` will scan the directory it's in, and give a list of years to choose from. The same applies to `ov_exec_pill_month`, which gives you a list of months, and `ov_exec_pill_day`, which lets you choose what pill you'd like to read.

The exec filters also solved the great problem of the copyright notice, which changes depending on the position of the article (articles in `current_issues` are not yet released as GFDL, GNU Free Documentation License).

So, in the XSLT transformation file `page_html.xsl`, you can see `<exec-post exec=''ov_exec_copyright''/>`, which is expanded into the right copyright notice for that particular article. The script is very smart: if it finds the "LICENSE" file in the directory where the article is placed, then the copyright notice is taken from the "LICENSE" file. This means that I can decide the default copyright (the GFDL if it's in `free_issues`, copy reserved if it's in `current_issues`), or assign

specific copyright for a specific article. This is specifically important for articles in the "fiction" section of the magazine, which are normally not released under the GFDL.

Converting OpenOffice and Word document into XML

After managing a magazine for a few weeks, you discover that it's pretty hard to get an author to follow the writing guidelines.

After a few years, you realise that you were wrong It's not hard... it's *impossible*.

You also realise that very few people want to write their articles using XML tags. This is the reason for authors writing their articles using OpenOffice, StarOffice or Word. The point was: how could we let people write their articles, save them in RTF, and then convert the RTF file into our XML?

There were several answers to this question. The main ones were to:

1. open the RTF file in OpenOffice (in batch mode?), save as OpenOffice's native format (which is XML). Then, apply a XSLT transformation to convert OpenOffice's XML into our own;
2. open the RTF file in OpenOffice or Word, and then use a Basic macro to create the XML file by hand.

The first solution would have been better from a technological point of view, and it would have been a lot faster. The big problem was time. Gian Maria, a Microsoft VBA guru, was available and offered to write the converter - but it had to be a Microsoft Word script.

Writing it wasn't easy at all: we had to decide on a style that would let the authors create a simple, readable RTF document, and convert it into a valid XML file. The solution was (obviously) the use of styles, but we couldn't be too invasive towards the authors, there had to be as few styles as possible, and the RTF document had to be easy to write.

We decided on an optimal file format (which of course was changed a million times in the process) and wrote a finite state automaton that scanned the RTF based on those styles into a valid XML documents.

It would take a long list of articles to explain all the problems we had to solve. In this article, I will only list a few of them:

- API incompatibility between Word for OS X and Winword. I would constantly get back to Gian Maria, saying "you can't really use this function because it's not supported". I don't know how he put up with me...
- UNICODE support. We had to allow UNICODE characters, and the only way to do that was to decode them ourselves (and therefore create the right XML entity). The problem? Think of PCs and Macs... big endian... little endian...
- Speed issues. VBA is very, very slow. The first versions of the converter would take up to 3 minutes to convert a document.
- Text boxes within the articles. The problem with text boxes was that you couldn't use a style for them, because they could contain anything else (headings, bullet lists, etc.).
- Word kept on crashing (especially on Mac).

Gian Maria and I exchanged a ludicrous number of emails. The first document generated by the converter looked OK, and yet it returned about 10 pages worth of problems when checked against the page's DTD. To make things more fun, I would sometimes change the XML format under Gian Maria's nose; some other times, I would report consistency problems in the XML files generated by his macro.

In the end, the converter was rock solid, reasonably fast, reliable on the XML, and above all it *worked*.

I don't think I'll ever be able to thank Gian Maria enough for what he has done He is a busy consultant, and gave up a lot of his free time to write this piece of software.When he offered to write an OpenOffice version of it I simply ran out of ways to say "thank you"! This time, there is no hurry and the XML format is not going to change. That will be the next step and by that point FSM will be entirely based on free software.

The PDF generation

There isn't much I can say about the PDF converter which is responsible for converting a bunch of XML files into a fantastic magazine. Gianluca Pignalberi did it all, and I don't know LaTeX in the slightest!

What I do know, is that he wrote a class called "openpaper.cls", and that his PDF files were simply fantastic. The PDF files created by Gianluca's class don't look like the "typical" LaTeX journals you see around. The PDFs look like a "proper" magazine, with coloured

text, textboxes, and everything else. His LaTeX class creates a fantastic table of contents and a great looking editorial board box.

His work has been invaluable. Without Gianluca, nobody would have ever been able to see a paper version of FSM. In the future, I am sure Gianluca will honour us with a paper on how he created the LaTeX class for FSM.

Without Gianluca, nobody would have ever been able to see a paper version of FSM. In the future, I am sure Gianluca will honour us with a paper on how he created the LaTeX class for FSM.

One thing I can say: I wrote the converter from XML into LaTeX, and realised how hard it is to do it! The problem is that LaTeX wants you to escape particular characters. For example, `$` has to turn into `\$` It would be possible to do so using regular expressions. But you can't! The first reason is that characters like `\` have to turn into `$\backslash$` (which contains itself a `$` symbol...). The second reason is that these rules don't apply in a CDATA section - and I had used CDATA sections for example in listings... This time, it was me who had to write a finite state automaton, in order to deal with all the escaping required by LaTeX. The script is called `ov_latex_preprocess`, and it applies to XML files before they are turned into LaTeX files.

The current problems

So, you may ask: what's missing?

Well, first we have to thoroughly audit our code. In general, the scripts tend to handle problems pretty well, but I am pretty sure I put one or two `die()` statements here and there. I would like to be sure that each script handles critical problems the same way.

Secondly, the system needs to get rid of the need for Microsoft Office to convert RTF documents into XML. This is possible, and it is an absolute priority. Now that we have a working system, and we are in no hurry anymore, we can take the time to "do it right".

We could for example port the macro to OpenOffice and see if we manage to get OpenOffice to process the file in batch mode; or we could write another XSLT transformation to convert OpenOffice's XML into our own.

Thirdly, the system needs a graphical interface. Right now you can only use it by the command line, but the system's inner structure is so consistent that writing a graphical interface would be extremely simple. The possibilities are amazing: you would be able to give your authors a login and a password to access their "slice" of their issue; they could then upload the `index.rtf` file, and see what it will look like once it's online or even printed.

To conclude...

Even though the system has a long way to go before it becomes a fully automated publishing system (if it ever will), FSM *works*. There are still a few bugs to iron out, but in general everything functions.

FSM's system is a little bit like Unix: it's complex as a whole, but each building block is simple and straightforward.

Creating Free Software Magazine gave us an opportunity to work together, and get to know and respect each other more. It is with enthusiasm that we now work on this project, and it will thrive mainly thanks to the hard work we have put in up till now.

Copyright information

About the author

Tony Mobily is the Editor in Chief of Free Software Magazine