Webmaster's Guide: When You Absolutely, Positively, Have Nothing Better to Do

Stephen Peters '92,'06

original 2006

Abstract

This is intended for MTG webmasters who are trying to figure out everything about how the Guild website works. It also tries to explain some of the reasoning behind the choices that were made here, as well as some of the little tricks I've used to make my life easier.

To make the most of being a webmaster, you have to know a little about a lot of things. Hopefully, this tome will go a ways towards giving you that leg up.

A formatting note: Commands that get typed in will usually appear '**like so**'. Other literals, like XML elements or pieces of stylesheets, will probably appear ⟨*thusly*⟩. Filenames will simply be in `typewriter` font. Anything that doesn't follow these conventions is probably because I fell asleep while typing this.

1 The Quick Summary

- Edit one or more of the `*.xml` files.
- Type '**gmake**'.

2 The Slightly Less Quick Summary

This site is implemented using XML to store the data, and a set of translating XSL stylesheets which turn the XML into pretty HTML pages. Plus, there's a Makefile which should do all the translating for you. The important bit is that you don't want to edit the HTML directly – you edit the XML or the stylesheets.

When you're making changes, it should be as easy as altering the XML (either by hand or through an XML editor of your choice). Then, just go to the directory with this file in it, and type '**gmake**'. This will rebuild all the necessary HTML files so that people can use the website.

3 Common Editing Tasks

There's a bunch of tasks you do over and over again as webmaster. Here's an example of edits that are commonly made. After you go through these steps, you should run '**gmake**' to make everything both hunky and dory.

3.1 Ending an old show/Starting a new show

When we start a new show, the site needs to be switched over to display the new, pretty show, and not the old, dead show that no one likes anymore. Steps for this are:

1. Move the top ⟨*showref*⟩ line from `upcomingshows.xml` to the first ⟨*archive*⟩ section of `archive.xml`.
2. If the new, current, show isn't already a ⟨*showref*⟩ line, create a new `[year]/[season]/show.xml` file (see existing `show.xml` files for the format), and copy the new show information into it. Turn the show into a ⟨*showref*⟩ line in the `upcomingshows.xml` file.
3. Make sure the `show.xml` file has everything useful, including dates, location, a description blurb, and an ⟨*attrib*⟩ with author credits.

3.2 Opening ticket reservations for a show

The website doesn't open tix reservations until it has a ⟨*tix*⟩ section with ticket prices. When you want to open reservations, make sure that the ⟨*dates*⟩ section has an individual ⟨*date*⟩ line for each performance, and make sure that the ⟨*tix*⟩ section has a ⟨*price*⟩ for each set of people.

3.3 Closing off reservations for a performance

Sometimes we sell out, sometimes it's just nice to update the site so that the ticket reservations window isn't listing performances that you can't see without going back through time. For this, the ⟨*date*⟩ elements can have a *reserved* attribute. *reserved="done"* is for performances that have already occurred; *reserved="full"* is for performances that are still coming up but don't have any more tickets.

3.4 Adding pictures to a show

When your archivist lovingly sends some pictures for the website, put them up for all to see by first throwing them into a directory, generating thumbnails, and then assembling them into a photo page within the ⟨*graphics*⟩ element.

To make all this easy as pie, use the '**tools/make-thumbnails**' utility, which will take a directory name and a photographer name, and create all the thumbnails as well as the XML that should go in the file. Just:

¿ **cd 2005/FAL** [or whatever the year/season is]
¿ **mkdir pics**
[copy a whole bunch of image files into the pics directory]
¿ **/mit/mtg/www/tools/make-thumbnails pics "Joe Photo-Taker"**

...and then cut and paste all the output and put it between the ⟨*graphics*⟩ and ⟨*/graphics*⟩ tags in show.xml. If you want to add captions, type them in for each photo just before the ⟨*/photo*⟩ tag.

3.5 After Guild Day

Ah, Guild Day. After lots of decisions get made, you should put them up. Update `officers.xml` with the new board members (and keep in touch so you know if there are new people for the

appointed positions). Then, add the chosen future shows into the `upcomingshows.xml` file (it's nice to list the backup show as well, at least until the show gets a producer).

3.6 Rehearsal Schedules

For some reason, they haven't been giving us rehearsal schedules to put on the web lately. This is because stage managers are silly, and don't recognize the value of having a neat little schedule that shows everything, in a format that has been tried and tested to display info in an easy-to-read and fairly easy-to-edit format. Like I said, silly.

But if they wake up and start asking for it again, here's what to do. In the `show.xml`, add a line that says something like ⟨*rehearsals file="schedule.xml"/*⟩. Then, you're going to create a `schedule.xml` file, which should look similar to other `schedule.xml` files elsewhere.

In the `schedule.xml`, there will be a ⟨*rehearsals*⟩ tag, filled with ⟨*day*⟩ tags that describe the rehearsals for each day. Each ⟨*day*⟩ will have an *date* element that defines the date.

Inside the ⟨*day*⟩ tags are ⟨*location*⟩ tags, which will list the rehearsals happening at a given location. The locations will use ⟨*place*⟩ elements to say where the rehearsal is, and a list of ⟨*rehearse*⟩ elements that say what's being done. You'll need *when*, *who*, and *what* elements to say what's going on.

You can also put ⟨*announce*⟩ inside the days, to put in major announcements like "off-book," or who is meant by the word "ensemble".

And then when you type '**gmake**', it usually only updates the schedule, for a nice, quick compile.

Sadly, there's no easy way to parse the dates and automatically get rid of old rehearsals. I usually comment out old info whenever doing a rehearsal update.

4 The Grungy Information

Here's where I go into an unbelievably pedantic description of the system details. Enjoy.

4.1 The XML Bits

The data is stored into XML files accordingly:

- `upcomingshows.xml` – contains information on the current show and future shows. Update this whenever we get information for upcoming shows, or when a show closes and we move it to the archive.
- `officers.xml` – The current list of officers and appointed positions for the Guild. This gets updated twice a year, after the officers are elected and they appoint suckers to fill positions.
- `archive.xml` – The list of archived shows. Those are the shows that already happened. You probably only need to update this after a show closes, when you move its information from `upcomingshows.xml` to this file.
- `[year]/[season]/show.xml` – When there's a lot of information for a show, (now defined as "hey, we added someone to the dir staff!") I move it from the `upcomingshows.xml`

file or `archive.xml` file into its own file. This makes it slightly easier to edit later, and keeps `archive.xml` from becoming many thousands of lines long.

- `[year]/[season]/schedule.xml` – When I'm dealing with a rehearsal schedule, I usually put it here. It's possible to put the rehearsals directly into the `show.xml`, but every time you update that file the makefiles will try to update the world. So, I put it here to define rehearsal schedules.

- `messages.xml` – Announcements for the home page that apply to the Guild as a whole, like big social events or membership meetings. If you have a show-related announcement (like auditions or interviews), it's a lot easier just to put it into the appropriate `show.xml` file.

- `prodstaff-desc.xml`, `ts-history.xml`, ... – these other XML files are really just HTML wrappers.

At first glance, XML looks a lot like HTML, but it's a bit more unforgiving of errors than HTML is. Every opening tag (like '⟨*show*⟩') *must* have a closing tag. If you want to have an empty tag (like '⟨*sep*⟩⟨*/sep*⟩'), you can abbreviate it by putting the slash before the closing bracket ('⟨*sep/*⟩'). I won't go majorly into XML here – you can do a search for 'XML' at Yahoo! or Google and get some good tutorials. If you know what a DTD is, the `mtg.dtd` file will describe the format pretty well – it's accurate for the most part.

The major tags we use are:

- ⟨*showref*⟩ – this pulls in information on a show from a different source (usually a show's `show.xml` file). The *href* attribute says where the show's XML data is living.

- ⟨*show*⟩ – this has a *title* attribute, a *time* attribute (which describes the season and year for the show), and a *dir* (which indicates the subdirectory that images and other show information inhabit – that's going to be in [year]/[season] format pretty much everywhere). Inside it can be most of the information below:

- ⟨*attrib*⟩ – contains information on the authors of the show. Multiple authors should be separated with a ⟨*sep/*⟩ element.

- ⟨*blurb*⟩ – contains a paragraph of information about the show. Usually I lift this from the rights publisher and then credit it with a link back to the rights site.

- ⟨*place*⟩ – contains two attributes, the *name* for the space where the show is being held, and a *loc* which will give a URL for the space location (usually a link to the MIT Campus Map). To make things easier, you don't need to provide a *loc* for some common locations. If *loc* is missing and *name* is 'klt', 'sala', or 'main', the system will automatically fill in information for Kresge Little Theatre, Sala de Puerto Rico, and Kresge Mainstage. You can also specify a *name* like '54-100' and the correct location will be substituted.

  The full list of locations is defined within `style/common.xsl`. But the current list is: 'sala', 'klt', 'main', 'rra', 'rrb', 'mezz', '20c', 'west', 'office', 'set-shop', 'lobdell', and 'tba' (for locations to be named later). 407 and 491 should be listed like 'W20-407'.

- ⟨*dates*⟩ – has a *desc* attribute, which gives a text description of the show dates. Can also contain a series of ⟨*date*⟩ elements, which use their own *desc* attribute to specify each individual show date and time (used to generate the ticket reservations page). Note that those ⟨*date*⟩ elements can also hold an optional *reserved* attribute, which can be set to "full" (for performances that are fully reserved), or "done" (for performances that have already occurred).
- ⟨*tix*⟩ – contains a set of price elements, each of which has a *value* attribute giving a ticket price and a *for* attribute describing the set of people who get charged that price.
- ⟨*graphics*⟩ – contains elements for logos, publicity, and photos related to the show. The elements are:
  - ⟨*logo*⟩ – can have a *large* and a *small* attribute specifying the filename for the large and small logos. The small logo is required, the large isn't. Small logos should be 130x130 in size.
  - ⟨*publicity*⟩ – has a *type* attribute for the kind of publicity item this is, and a *ref* attribute pointing to the filename holding the publicity.
  - ⟨*photos*⟩ – you can have a bunch of these, each with a *header* attribute which will allow you to partition the photos into sections. Most of the time, you're just using one ⟨*photos*⟩ element, though, and you neither need nor want a title.

    The ⟨*photos*⟩ element contains a list of ⟨*photo*⟩ elements, each of which has a *ref* attribute pointing to an image filename, and an optional *thumb* attribute pointing to a smaller version of the image. The ⟨*photo*⟩ element can have as its text a caption for the photo. If the *wide* attribute on the photo element is provided, it signifies that the thumbnail is wider than normal, and will be displayed in a wider format across the two photo columns. Finally, you can also have a *credit* attribute, so that you can properly credit the photographer.

    If there is an external page of photos somewhere, you can also supply a ⟨*page*⟩ element, with a *ref* attribute containing a URL to the page and a *org* attribute that names the organization maintaining the photos.

    A note on photo creation – often, once the photos from a production are uploaded, we create thumbnails of a standard size, so that they fit in a 200x200 rectangle. If ImageMagick's '**convert**' tool is installed, this can be handled through a command like:

    convert -scale 200x200 [filename].JPG [filename]-thumb.JPG

    The thumbnails can be easily created using the '**tools/make-thumbnails**' program. See the above section on installing archive photos for an example use.
- ⟨*notes*⟩ – information about the show. Can be information about interviews and auditions, or could be historical information like "Tech Show 2005". Note that this can have sub-sections:
  - ⟨*messages*⟩ – info for the front page. If you have announcements (usually for auditions or interviews), you can put them in a ⟨*messages*⟩ section inside the ⟨*notes*⟩.
  - ⟨*fullnotes*⟩ – special info for the current show. Usually this is a section on additional information for auditions, so that the users can get more information about the whens and hows of auditioning.

- ⟨*prodstaff*⟩, ⟨*cast*⟩, ⟨*orchestra*⟩ – Each of these contains a set of ⟨*job*⟩ elements describing a job and the people who worked on it. The ⟨*job*⟩ element has a *title* attribute for specifying the job name, and the contents of the job element specify the people. Multiple people can be separated with ⟨*sep/*⟩ tags.
- ⟨*review*⟩ – links to external reviews, with *ref* attribute containing a URL to the web page, an *org* attribute naming the organization reviewing the show, and a *title* attribute summarizing the review.
- ⟨*disclaimer*⟩ – If the show has objectionable content, the directors may want a notice to that effect.
- ⟨*html*⟩ – contains a *title* attribute with a page title, and encapsulates the body of an HTML document inside it. This body will get wrapped with the standard headers and upcoming show information. We usually only use this for those files under the "About MTG" link, which contain long-lasting information about our shows and organization.

5 Version Control

I setup CVS as version control on the entire contents of the website. I did this for two reasons. The first is for backup purposes. Although I have all the faith in the world that MIT will manage to not blow away the entire MTG locker in a fit of pique, if they did do that accidentally we'd be really, really sad. So, keeping a mirror of the entire locker somewhere is a good idea.

The second reason is for change tracking. If you make a mistake and need to go back to what the website looked like a couple of days ago, you can do that (provided you're keeping up on commits to CVS).

Right now, the CVS repository is stored on an external computer owned by me, Stephen Peters, long-time Guild member. This should probably be switched to something like `myfairlady.mit.edu`, but I actually trust that machine far less than I trust my own, so I've set up this situation.

For security, the only way to access that external CVS repository is through SSH public-key encryption. To setup your own public key, look for information on the '**ssh-keygen**' program. Once you've set up the public key, it will be in your `.ssh/id_dsa.pub` file or something similar; email it to `portnoy@alum.mit.edu` and he'll make sure that you can use it to get repository access. Also, look for info on '**ssh-add**' to find easy ways to add the SSH key to your session so you don't have to continually type in your password. Oh, and don't forget to set the `CVS_RSH` environment variable to "`ssh`" or it won't work.

Updates are pretty easy. You can find lots of CVS tutorials on the web, but the commands you'll probably care most about are '**cvs add**', for adding a new file or directory to the repository, and '**cvs commit**', for committing everything you've modified.

Before you ask, yes, I thought about using Subversion. The problem with Subversion is that it uses lots of disk space to stash the unedited versions of the files that have been checked out, and with Athena quota constraints that's just not acceptable. One interesting possibility would be SVK, but that has its own problems.

6 What the Heck Were You Thinking?

A socratic dialogue on why this stuff was created this way in the first place.

So, why all this XML stuff? HTML not good enough for you?

In a word, no. The web page design involves lots of information repeatedly shown on lots of different pages, so a straightforward HTML solution would have resulted in editing dozens of pages every time the show information changed. Some of this could have been fixed with clever applications of Server-Side Includes (which, by the way, wasn't implemented on MIT's website when I started all this), but there would still have been things that would be very difficult – like making sure that the show information got properly copied over to the ticket reservations form, for example. The more places you have to type in information, the more likely it is that you'll miss something. So, whenever possible, changes are put in one place, and then copied everywhere.

Well, we could go the other way. Build our own server, and store everything in databases that are easier to edit than these XML files.

Sure. And then someone has to maintain that server. Right now MIT does it for us, and that's actually a big win.

Grrr. I really hate XML files.

Which is too bad, because I think they kind of have a crush on you.

Really? Well, that's flattering. OK, I can make my peace with them. But what about the annoying behavior of the reservations and interest forms? I mean, really, the entire "type in a form, and it creates an email message?" Surely there's something we can do about that!

Stop calling me Shirley. But no, this is the way that MIT allows us to have access to user-submitted data. Anything more intricate, and we're talking the build-your-own-server thing again. If MIT ever gives us access to something better than cgiemail, we should look at it. But right now, nuh-uh.

How come 'make' doesn't work? 'gmake' is really clunky to type!

Well, make doesn't always work because Athena didn't standardize on GNU make. If you're on a Sun box (as often happens when logged in remotely), make is Sun's System V make, which doesn't have lots of useful stuff, and differs in some niggling details. But gmake is GNU make, which is the one you want. Trust me.

Hey, it doesn't work on this Athena workstation I'm using!

That's possible. I do most of my testing logged into the "dialup" servers, so I may have missed some details of the workstation environment. If you find something odd, I'll be glad to figure out how to make it work, but you might also try editing the Makefile yourself to load in whatever feature you need.

Whoa. I just looked at this LaTeX handling code. What the heck is going on here?

Well, it's kind of interesting. What I do is take the LaTeX file, and first turn it into a PDF using `pdflatex`. Then, I run it through `latex2html` to turn it into a `.htm` (HTML) file. That gets turned into XHTML (`.xml` extension) using `tidy`, which I then can plug into the website so that it looks nice and pretty.

But...but...that's insane!

Thank you.

It's not a compliment.

Neither are ugly pages that don't fit in with the look and feel of the rest of the site.