Active User Guide

Matthew Arnison
webcoders (http://cat.org.au/lists/webcoders/) (the active software development collective)

maffew@cat.org.au

Active User Guide

by Matthew Arnison

Active is a set of web scripts for helping activists share news, events, and group information. The news system handles multimedia, open publishing, and high traffic. This guide explains how to use the software at several different levels: for users of an active web site, for people organising the site, for people setting up and maintaining the software, and for people programming the software.

The current release version of active is 1-7-0cvs .

Table of Contents

Chapter 1. About Active

Active is a set of web software that does multimedia news, events and groups listings. In this chapter we go into detail about what active is designed to do, why it exists, and a little bit about this guide too.

This chapter is $Revision: 1.6 $. The last update was on $Date: 2001/05/19 12:47:24 $, at which time the current release version of active was 1-7-0cvs .

1.1. About the Active Software

The active software creates a set of web pages which allow web surfers to contribute to a shared calendar, groups listing, and multimedia news with discussion. It's designed to be easy to use, and easy to read the results. It is free software, and copyleft (http://www.gnu.org). It can be run on an entirely free software server.

active is under ongoing development as the backend for www.active.org.au (http://www.active.org.au), an activist web site in Sydney, Australia and indymedia.org (http://www.indymedia.org), a global network of activist news sites. It's primarily designed for busy activists, but by tailoring the text and graphics, you could probably use it in a bunch of different situations. See www.active.org.au/source (http://www.active.org.au/source/) for the latest version of this software.

1.2. Goals and Context

This is what I think the goals are.

We want to help activists organise and motivate and inform. We want to create a website that's a positive heartland for activists, a sense of place, a sense of achievement and reflection; and vibrant and open enough that it attracts people who are interested in actvism; and practical and focussed enough that all sorts of people in our audience dream up solutions to local and global problems and get out there and make them

happen. A nice side-effect is that we also become a source for stories for the wider media.

We want to help people develop the art of story telling and debate. We want to be a catalyst for those stories to reach into other media, parts of the city and the planet not touched by the web. We want to break down barriers and encourage the flow of information from people with both good and bad stories to tell, to the people we know are out there who want to hear them.

We are using the internet because it's cheap, it allows anyone connected to contribute, it's global as well as local, and it feeds into and out of other media. And it's fun too. It's free software (http://www.gnu.org) so that anyone can take it and reuse and adapt it, and contribute back in too if they want.

I wrote a rant about activist webcasting, and how it's important to have open publishing, frozen media nuggets, and free software (http://cat.org.au/cat/webcast.html). See also this quick and dirty explanation of what free software is (../freesoftware.txt).

How about some examples:

- Money from the rich countries is tied to privatisation for poor country social services - after a sell off the price of water in Bolivia rises higher than Washington DC USA, people living there can describe the experience direct to activists worldwide (dc.indymedia.org) (http://dc.indymedia.org) as part of the corporate globalisation protests on April 16 2000.
- A local community radio station is looking for volunteers, a dozen of the people who show up at the first training session say they found out about it on the www.active.org.au/sydney (http://active.org.au/sydney/) events calendar.
- November 30 1999, and the corporate media is doing the usual snow job on protests in Seattle. Giving the lie to patronising mainstream coverage, the Independent Media Centre (seattle.indymedia.org (http://seattle.indymedia.org)) brings to light the brutal violence of police, the passion and creativity of the protestors, and the ultimate success in shutting down the World Trade Organisation. Pictures, sound, video and hot text circle the globe independent of centralised corporate control.
- Looking for a local group campaigning on cycling, housing, indigenous justice?

Over a hundred such groups have contributed their own details to active-sydney (http://active.org.au/sydney/).

- Of course there's a lot of fantastic projects doing these kinds of things. The ones above are just some examples of ones tht are using the active software. Here's some other approaches:
  - news.tao.ca (http://news.tao.ca)
  - protest.net (http://protest.net)
  - damn.tao.ca (http://damn.tao.ca)
  - www.schnews.org.uk (http://www.schnews.org.uk)
  - www.activistsandiego.org (http://www.activistsandiego.org)
  - San Francisco Bay Area Progressive Calendar (http://www.emf.net/~cheetham/cal.html)

Toby from Schnews helped me (Matthew) draw up a very personal journey through the roots and shoots of the active software (../roots.pdf), no doubt I left out some huge influences that should be in there. People for Global Action (http://www.agp.org/) should definitely be in there. And Next 5 Minutes (http://www.n5m.org/), a series of three tactical media meetings in Amsterdam.

1.3. Features

Here's some specifics on what the active software does.

As of version 1.41 (April 2000):

- overall design
  - easy to contribute info using web page forms - if you can use hotmail, you can contribute to active!

  - open publishing: there is no editorial approval process - items submitted over the web show up immediately on the web pages, some people consider this a feature, others a bug!
  - information is clearly presented, designed to print out clearly
  - does not use fancy browser options like Java, Javascript or frames; so it works with older equipment and also with software for people with disabilities, see www.useit.com (http://www.useit.com) for writings about this sort of design philosophy
  - designed so it can be set up for independent use in different cities, even on the same server - but no ability yet to share info between those web sites
  - colours, headers, footers and graphics can be customised for a specific city installation without breakign the ability to upgrade to new verisons later
  - basic syndication - news and events details can be delivered in raw text for reuse and reformatting on other web sites
  - the package is free software, everything you need to run it on a web server is also free software
- news / webcast
  - users can add their own news articles
  - contributed stories can be text, images, sound or video; all published and managed using a simple web page form
  - basic ability for readers to add comments underneath stories
  - any new stories go immediately into the summary view of the news, which can be included in the front page of the website
  - if images are bigger than a given size, a thumbnail is generated for display in the summary view
  - related stories can be linked so that they are grouped together in the summary display. e.g. a picture with a text story; a modem quaity realaudio sound piece

with a high quality MP3 for reuse by radio stations; a colleciton of pictures of the same event; a video with a freeze frame image

  - management page for editing, linking and removing news
  - facilities to help manage rewriting web addresses for multimedia stored on different servers (mirrored), and optionally delay linking to stories while they are transferred behind the scenes
  - the summary pages are cached on the server to dramatically increase the number of viewers the web server can deal with if the site gets busy

- events calendar
  - calendar presented as either a listing with events about to happen shown first, or as a monthly calendar
  - users can add their own events
  - users get a login on the calendar so they can come back and edit or remove them
  - new events can be emailed to an announcements list, and a weekly email calendar can also be generated
  - editing users and events that you did not create is fiddly, involves hand editing the text databases

- groups listings
  - currently very basic display
  - readers can add info about groups they are involved in, some details in the form are Australian specific
  - admin page for editing and deleting groups

Please see also the wishlist (../wish).

1.3.1. How does active compare to slashdot's code?

I wrote up a comparison in this email to webcoders

(http://reflect.cat.org.au/lists/webcoders/msg01464.html).

1.4. Thankyou

Gabrielle Kuiper for the energy and the name behind the launch of active-sydney (http://active.org.au/sydney/) in 1998-99. The active collective for organising active.org.au, trusting our audience and exploring open publishing.

Cameron Shorter and Matthew Arnison for initial programming on the website calendar and groups listings, including generalising it just enough to be used by other cities as free software (open source). xchange (http://www.xchange.anarki.net) anarchist BBS in Melbourne for demonstrating so quickly how open source can autonomously spread.

Selena Sol for the calendar code we started from and for granting our request to relicense it under the GNU copyleft GPL. Richard Stallman for suggesting we chase Selena about that option.

The cat (http://www.cat.org.au/) crew for the J18 webcast (http://www.j18.cat.org.au/) (June 18, 1999) when we first designed and used the webcast software. cat is also where active is hosted.

Andrew Nicholson for programming the first version of the news and then the webcast software, and continued contributions to mutations thereof. Matthew Arnison for dodgy hacks on the server-side caching and multimedia support.

The late EMU technology cafe (Sydney) for lots of lovely vegan meals and positive vibe - a great place to have meetings and geek out. We miss you! The activist warehouse in St Peters (Sydney), a home base for cat and sydney.indymedia.org (http://sydney.indymedia.org/). Colene Woods for putting so much of her heart into the start up of indymedia in Sydney.

Inspirations, ideas, skills: community media in Australia, including the Community Broadccasting Association of Australia and their virtual conferences where we learnt lots about webcasting, Community Access TV (CAT’s ancestor), all the groups mentioned in the goals and context section above. Reclaim the Streets (Sydney and

globally) for great parties and webcasting from the footpath. The free software movement for providing so many powerful tools we could learn from and build on. slashdot (http://slashdot.org/) for making the free software movement visible and self-aware, and for trusting their audience.

The folks at freespeech.org (http://freespeech.org) (Colorado, USA) for meeting up with a stranger from Australia in October 1999. Peter McGregor for making the email link between cat and the indymedia centre in Seattle USA. Mansour Jacobi for trusting in our cobbled together software mess and linking it into the indymedia project, web design hacks and generally being a technical backbone.

All the amazing street activists, media activists and geek activists who made the Seattle World Trade Organisation protests (November 1999) such a turning point. The activists from the global South for still being there when finally us rich country folks woke up and were ready to hear the real story and help fix it. The people who posted their personal stories to the web, showing the power of open publishing. The organisers of the Seattle indymedia centre for linking so many media activists together in a powerful network.

The ongoing indymedia audience for being creative, diverse and inspiring. The contributing media activists for continuing to collaborate.

The indymedia volunteers all around the world who have stretched this dodgy software way beyond what we ever dreamed it would do, including translating it into a dozen languages.

Rabble Rouser for the front page features code. Bonnie in NYC USA for various helper scripts. John Kawakami for trying to get server side caching improved into something half reasonable. The SF Indybay geeks for adding in ratings, even if it never got to a state where it could be patched in. Idan Sofer for active in the first non-latin language: Hebrew.

All the people who have contributed ideas (see the wishlist) and code (see the cvs logs and the CHANGES file). If you've contributed and I've left you out, please let me know and I'll add you in!

1.4.1. Documentation helpers

christopher mitchell <cmitchell@macalester.edu> for the section on learning Linux.

1.5. About This Guide

It's about time we had a place to bring all the doumentation together for active.

The newest version of this guide can be found at http://active.org.au/doc (http://active.org.au/doc/).

> **Tip:** *Technical note*: throughout this guide the technical information assumes you are running Debian GNU/Linux (http://www.debian.org/) 2.2 on your webserver and possibly for other tasks such as documentation. It'll probably work fine on other versions of Unix (and probably not so well on Windows or Mac), it's just that I wanted to work out a complete reproducible path to setting things up, and Debian seems to provide a good starting point. It's also what I have easily available to test with. If you'd like to see it documented it under alternative systems, please write it up and send it in, and I'll include it.

1.6. sections of this guide we still need to write:

- how to use the site as a web surfing reader and writer and media producer
- how to maintain the site over the web and otherwise
- step by step installation and setup
- software design (or lack of it :)

- how to best contribute programming to the project: check for consensus on new features or make them optional
- how the webcoders collective is organised

Chapter 2. Editing and Managing an Active Website

This chapter is $Revision: 1.2 $. The last update was on $Date: 2001/05/19 12:47:24 $, at which time the current release version of active was 1-7-0cvs .

2.1. Overview: how active creates pages

The web pages are generated in several different ways.

1. the static web pages, including the about and howto sections, are generated using server side includes (.shtml files), to ensure a common look for all pages
2. the news and groups listings are programmed using PHP3 web scripting (.php3 files) with PostgreSQL storing the information in a database, and multimedia files received using http push from the user's browser and stored in the local filesystem
3. the events calendar is generated using a large Perl script with the database stored in text files. This is a heavily modified version of the WebCal script from www.extropia.com (http://www.extropia.com)
4. the front page is generated by combining static text with a dynamic headline view of the current events

The common graphics, headers and footers for the site are in the "local/include" and "local/images" folder. The configuration files are also all in the "local" folder.

Chapter 3. Setting up and maintaining active on a web server

This chapter is $Revision: 1.8 $. The last update was on $Date: 2001/06/11 04:57:53 $, at which time the current release version of active was 1-7-0cvs .

3.1. what software it depends on

Currently the active software depends on a handful of software installed on a web server (if there is a minimum required version for the software it is shown in brackets):

- a unix operating system has been assumed (e.g. linux)
- a web server (we use apache (http://www.apache.org)) - with access to server side includes (.shtml) and perl and PHP, plus local write access to certain folders for storing the calendar text databse, and multimedia news uploads
- perl (http://www.perl.org) scripting, including the Time::DaysInMonth module from CPAN (http://www.cpan.org)
- the ImageMagick program "convert" is used on the web server to create thumbnails of published images
- the unix program wget is used for certain page refreshing functions
- PHP (http://www.php.net) scripting (version 4.x), including PHP support for PostgreSQL
- PostgreSQL (http://www.postgresql.org) - an SQL database (version 6.5.x)
- access to a mailing list server, such as majordomo (http://www.greatcircle.com/majordomo/) or mailman (http://www.gnu.org/software/mailman/mailman.html) for distributing events and news by email - or you could use an online service like egroups (http://www.egroups.com)

- access to a unix shell for things like a scheduler, like unix cron, for weekly automatic emailing of event calendars, and for creating symbolic links within the web scripts folder
- access to a streaming media server, currently RealMedia (http://www.real.com) and MP3 are supported, and some sort of mirroring system such as "mirrordir" (active patched version (http://active.org.au/source/mirrordir/) recommended) or "rsync"
- code versioning system (CVS) if you want to keep up to date with the latest versions, or easily contribute your changes to the code back into the project

If you've not heard of some of this software, check freshmeat (http://freshmeat.net) for details.

3.2. Learning Linux

For many people, indymedia is their first experience with Linux. Here are some good places to start learning about the Linux way of doing things.

- A good primer for general linux knowledge appears to be this Linux Tutorial (http://www.linuxdoc.org/LDP/gs/node5.html). That primer is part of the Linux Documentation Project (http://www.linuxdoc.org/), a great place to look for FAQs, HOWTOs, and online books about Linux.

  For something on paper, check out the excellent Linux and Advanced Linux Pocketbooks (http://www.pocketbooks.net.au/).

- A crucial tool for remote administration is **ssh**. It allows you to execute typed commands on the server interactively. You might be more familiar with its ancestor **telnet**: ssh is simply an encrypted version on telnet, and therefore much better for server security. You can find more information about ssh including free software servers and clients for various operating systems at the Open SSH (http://www.openssh.org/) website.
- You might know ftp, but unfortunately it's not very secure because it sends your

password in plain text over the net. Check out the secure alternative in this introduction to sftp and scp (http://www.linuxgazette.com/issue64/dellomodarme.html).

- For tips on general Linux server security, check out this quick reference card (http://www.linuxdoc.org/LDP/ls_quickref/QuickRefCard-A4.pdf).
- If you're setting up a server from scratch you'll need to choose an operating system (OS) and a distribution of that OS. I (Matthew) have had a lot of good experiences with Debian Linux (http://www.debian.org/). Debian is an awe-inspiring collective effort of hundreds of volunteers who package up Linux and associated free software. It works particularly well for remotely-maintained web servers, and it's used at cat (active's home) and indymedia on the US (stallman) and London (durutti) webservers.

3.3. Handy shell environment tricks

It's always a bit confusing to me how best to configure default shell aliases and environment variables. Unfortunately debian doesn't seem to be well documented in this regard. But by playing with debian 2.2, I think I figured out how it works.

For normal text shell logins, either at the machine or over the net with ssh, you need to add aliases and environment variables to /etc/profile. Except for some annoying reason this file is not read when you login under X, for that you need to add stuff to /etc/bash.bashrc (which is not read by text mode shells at startup!). There must be a better way to do this!

So it's useful to add aliases for: proc, cvsa, cvsp.

```
function proc ()  { \
        com-
mand echo "USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME CO
MAND"; \
        command ps auxwwwwwww | grep $* | grep -v grep; }

export CVSA=:pserver:anonymous@cvs.cat.org.au:/usr/local/cvsroot
```

```
export CVSP=:pserver:${USER}@cvs.cat.org.au:/usr/local/cvsroot
export CVSROOT=:ext:cvs.cat.org.au:/usr/local/cvsroot

alias cvsa="cvs -d $CVSA"
alias cvsp="cvs -d $CVSP"
```

- **proc** is a shortcut for searching through the list of running processes by name. You can use it to quickly find matching process names or users. E.g. **proc apache** to find how many apache processes are running.
- **cvsa** and **cvsp** are shortcuts to **cvs** (Code Versioning System). **cvsa** is for read-only anonymous access to the active CVS server, and **cvsp** is for read-write access where you have a login and password on cvs.cat.org.au (assuming your cat username is the same as your current login). Before using either you will need to login (the password for anonymous access is blank) e.g. **cvsa login**.

3.4. Setting up postgresql for communication with PHP

PostgreSQL (http://www.postgresql.org/) is a free software SQL database server. When we began programming on active in 1999, it was GPL (copyleft) but MySQL was not. Also PostgreSQL has a richer feature set. It also has better bugs, increased quirks, and decreased speed, although things seem to be improving.

One of the challenges with PostgreSQL is setting it up to communicate with PHP so that web scripts can access information in the database tables. So this is how to setup PostgreSQL, atleast on a Debian GNU/Linux (http://www.debian.org/) system.

1. Install these debian packages and their dependencies: postgresql, postgresql-client (you want atleast postgresql 6.5.x which is in debian 2.2 stable - postgresql 7.x is worth getting if you think your server might get really busy), makepasswd (needed

to manually make encrypted passwords), php4-pgsql (now available in debian 2.2 stable).

2. Change to the postgres user, e.g. with **su - postgres** or by putting **sudo -u postgres** in front of your command. I'm going to use **sudo** because it's cool.

   Make yourself a postgres database admin user: **createuser your-username**

   ```
   $ sudo -u postgres createuser your-username
   Enter user's postgres ID -> xxxx
   Is user "your-username" allowed to create databases (y/n) y
   Is user "your-username" a superuser? (y/n) y
   WARNING: Any user who can add users can also modify the sys-
   tem catalog
   createuser: your-username was successfully added
   ```

   where **xxxx** is your Unix userid (you can find that out by running **id** when you're logged in).

3. Now that you are a postgresql superuser, you can setup another database user (not a Unix user) for communication between PostgreSQL and PHP: **createuser php**

   ```
   $ createuser php
   Enter user's postgres ID -> xxx
   Is user "php" allowed to create databases (y/n) n
   Is user "php" a superuser? (y/n) n
   createuser: php was successfully added
   Shall I create a database for "php" (y/n) n
   don't forget to create a database for php
   ```

   where xxx is some userid that isn't going to conflict with a human username id, I used 100 because on Debian the userid's for humans start at 1000.

   Now setup a password for this user:

   - Pick a password for the php user to use when connecting to postgresql, for this example I'm gonna pick **secret**. Except you should pick one that's not an english word because it's more secure.

- [optional] Encrypt the password. This is good security - it means a cracker cannot easily find out the password even with system level access - which is good because those passwords might be used in other places a cracker hasn't got access to yet. However, it's maybe more hassle than it's worth because sometimes as a system admin you might need to find out the password without wanting to reset it, and it is only useful for connecting to the database from the local machine (unless you open up your database to other machines, which is not recommended for beginners). Also, the active code contains the password in plain text, so a cracker can easily find it there if they have a login on the server.

  Anyway, the command is **makepasswd**, see below for how to use it:

  ```
  $ echo "secret" > /tmp/phppass; makepasswd --crypt --
  clearfrom /tmp/phppass; rm /tmp/phppass
  secret     DpR4hs.6l19SI
  ```

  the bit on the left is the crypted password. Use that instead of `secret` below.

- Connect to the default database `template1` and set the password for the php user:

  ```
  $ psql template1
  Welcome to the POSTGRESQL interactive sql monitor:
    Please read the file COPYRIGHT for copy-
  right terms of POSTGRESQL
  [PostgreSQL 6.5.3 on i686-pc-linux-
  gnu, compiled by gcc 2.95.2]

     type \? for help on slash commands
     type \q to quit
     type \g or terminate with semicolon to execute query
   You are currently connected to the database: template1

  template1=> up-
  date pg_shadow set passwd='secret' where usename='php';
  UPDATE 1
  template1=> \q
  $
  ```

- Edit the PostgreSQL security filter to tighten up security so that a password or identity is checked before allowing access, and so that access is only offered to people on the same machine as the database: `/etc/postgresql/pg_hba.conf`. The default debian setup gives users on the local machine access as whatever username they want (if they know the right psql options) and denies connections from other machines.

  Warning

  I think a good security balance would be to allow local users to connect using their own username without a password *or* with a different username and a password. However, I haven't figured out how to actually do this; my attempts so far at what looks logical seem to fail.

- Edit the PostgreSQL startup options so that it listens for TCP/IP connections (even though we are only connecting from the localhost, PHP only knows how to connect over TCP/IP):

  sudo vi /etc/postgresql/postmaster.init

  Uncomment the `PGALLOWTCPIP` option and set it to yes, so that the whole line reads: **PGALLOWTCPIP=yes**

  Then restart postgresql: **sudo /etc/init.d/postgresql restart**

4. Setup a database for your active city website: **createdb active_cityname**
5. Run the database definition scripts for the parts of active you want to use, e.g: **psql -e active_cityname < /www/active/cityname/webcast/webcast.def**. This creates the tables, indexes and sets access permissions for PHP.

   The SQL database definition files in active assume you are using the username `php` to connect PostgreSQL with PHP. If you are using a different username, you'll need to edit the **GRANT** lines at the bottom.

6. Test it works from the command line using **psql -u active_cityname** (postgresql version 6.5.x) or **psql -U php active_cityname** (postgresql version 7.x).

3.5. Active installation notes

We designed this software to be easy for people to use on the web, and also to be easy for us to code. I think we may have achieved these things, but ended up with something a little harder to install.

The top folder is designed to be the top of the active web site. So index.shtml is the front page. Then there are various scripts (inline via PHP3 and CGI via Perl). These depend on text include files, text static images (see the local/include and local/images folders), databases, and SQL databases (PostgreSQL).

More detail on what you will need:

- a server machine (we use Linux www.linux.org on a Pentium, you could probably use any Unix) with web server software (we use apache www.apache.org)
- add index.shtml, index.cgi and index.php3 to the line in your web server config specifying the default web page for a folder when the user only gives the filename (e.g. active.org.au/news/)
- server side includes turned on for the active folder
- CGI scripts turned on for .cgi files in the "active" scripts folder
- the right permissions, symbolic links, and folders (see the Makefiles)
- the PostgreSQL database with access via TCP/IP enabled for localhost connections (i.e. an internet link from your web server direct back to your web server). the permissions for this can be tricky. we set up a special PostgreSQL user, then GRANTed it permissions to read and modify the database. you also need to tell PostgreSQL seperately that locahost internet access is OK and set a password, other wise it’s off by default. look in the news/news.def, groups/groups.def and webcast/webcast.def files for table definitions within the "active" database

- the PHP software installed as a loadable apache module. you might be able to use it via a CGI script (we haven't tested this)
- the PHP loadable module that lets you talk to the PostgreSQL database server

3.5.1. What's in which folder

[Need to check if this is still up to date.]

See the file layout diagrams (http://process.indymedia.org/tech/active-files/) for more clues.

- top level folder "active"
  - the front page wrapper scripts index.cgi and index.shtml (index.shtml should have precedence - check your web server configs if it doesn't)
- calendar
  - the main calendar perl script index.cgi, uses forms to let people add their own events
  - also daystogo.pl and hourstogo.pl which use some of the calendar libraries to provide a plain text countdown to an event, which can then be included in a front splash page or something, updated using cron
- calendar/Documentation
  - the docs that came with WebCal before we heavily hacked it
- calendar/Library
  - some libraries that the calendar script depends on
- groups
  - code for the groups listings, uses forms to let people add info about their group

- news
  - code for the news
- webcast
  - multimedia version of the news, specialised for a webcast of a live event
- shared
  - shared code which both the news and webcast code uses, at the moment just the code which handles SQL queries
- local
  - all the configurations or local info for a given city or site using the active code should be in here. To get a new active site up and going, you shouldn't need to edit anything outside of the local folder and its subfolders. Unless of course you want to do some heavy duty customising!
- local/db-setup.php3
  - the configuration stuff on how to access the PostgreSQL server over TCP/IP
- local/about
  - the about page, currently static with header and footer includes
- local/calendar/calendar.setup
  - the main configuration file for the calendar
- local/calendar/Calendar_session_files
  - temporary session files used to keep track of individual visitor settings
- local/calendar/Databases
  - the flat text event databases: events, users, and a unique ID counter
- local/calendar/Databases/dummy

    - I think you can have subfolders like this to manage multiple calendars, not sure because we don’t use it

- local/groups
    - configuration files for the groups listing
- local/howto
    - the howto page, listing useful resources, currently static with header and footer includes
- local/images
    - the logo that goes in the top right hand corner, and some other images we use
- local/include
    - front page bits including a feature banner (feature.inc) and a blurb at the bottom of the left hand column (front.inc), and the headers and footers used for all pages (header is formed by sandwiching the logo in between top1.inc and top2.inc -- footer just uses bottom.inc)
- local/news
    - news setup
- local/news/cache
    - as the front page news only needs to be changed when a story is added, edited or deleted, we keep a copy here to save time in serving the front page. To force a refresh of this file (say, if your database crashed and you recovered it - i find PostgreSQL’s command "vacuum" very handy in this situation by the way) access the script news/refresh_front.php3 from your web browser
- local/news/uploads
    - for the multimedia news, this is where we store images, sounds and videos that people upload

- local/webcast
  - webcast setup
- local/webcast/cache
  - again because the index pages involve quite complex SQL queries, we cache a copy here between story changes
- local/webcast/uploads
  - for the multimedia items, this is where we store images, sounds and videos that people upload
- local/log
  - any logs, currently just sql_log which lists all SQL queries made by the news and webcast code, together with a datestamp

Some tech. details on the *webcast* folder:

```
web browser start points:
-------------------------

active.org.au/dev/maffew/active/webcast/display.php3
  - display front page sum-
maries, does database query and HTML construction on
  the fly, also displays articles (al-
ways done on the fly at the moment) using
  ?article_id=N where N is the database article ID

active.org.au/dev/maffew/active/webcast/refresh.php3
  - calls the above code in a special mode so that in-
stead of outputting the
  page to the browser, it dumps them in files called
  ../local/webcast/cache/frontN.html where N is the page number
  this saves the web server thrash-
ing when we get lots of visitors
```

```
active.org.au/dev/maffew/active/webcast/front.php3?page=N
  - wraps the ../lo-
cal/webcast/cache/frontN.html files with the header and
  footer

active.org.au/dev/maffew/active/webcast/search.php3
  - searching, currently very specific to seattle, needs head-
ers and footers
  generalised to point at ../local and date specifica-
tion made more general

active.org.au/dev/maffew/active/webcast/publish.php3
  -
enter new stories, or click on link to link/edit/delete existing
          ones
  - i suggest adding a few dummy sto-
ries to get a feel for how this works

active.org.au/dev/maffew/active/webcast/display.php3?led=y
  - story edit mode
  - can link related items here, e.g. a still im-
age to a text story about
  the same event
  - can delete items
  - needs a few added features: ability to un-
delete, show a page of stories at
  a time instead of all of them

script files layout:
--------------------

[slightly out of date]

active/webcast:

auto_add.php3           not sure
```

```
cast_class.inc          display routines for front page sum-
maries or articles
debug                   if this is present, the de-
bug database tables are used,
                     so we can play without stuff show-
ing on the live site
display.php3            main display routine -
front page summaries, article
                        display, linking, editing, delet-
ing (led)
edit_webcast.php3       edit a story
entry_munger.php3       i think this takes entryform.inc and in-
serts data
                        into the fields so a story can be edited
entryform.inc           the plain form for entering and edit-
ing stories
led-process.php3        processes actions from the LinkEdit-
Delete form
make_anim               perl script to make ani-
mated GIFs, not used
make_thumb              perl script to make thumbnails if up-
loaded image is
                        bigger than, say, 160x120
new_data-
process.php3   processes data after people press publish
publish.php3            publish form
refresh.php3            hacked version of dis-
play.php3 which creates html files
                        instead of writing front page to stan-
dard out
search.php3           search for articles
uploads                 symlink -
where uploaded images / vids / sounds get put
                        also i think dis-
play.php3 looks for static.html here,
                        which is a header that goes on ev-
ery page
```

```
webcast-debug.def       postgresql table definition
webcast.def             postgresql table definition
webcastd.pl             daemon which monitors mirroring of au-
dio / video files
                    used for J18 but not for N30

active/shared:

db_class.php3           all functions for accessing sql ta-
ble (all sql queries
                        get logged to ../local/log/sql_log)

active/local/:

appear.inc                      sets colours for all active apps
db-setup.php3                   sql db config
webcast/webcast-setup.php3      config variables for webcast
webcast/webcast.inc             header and footer defini-
tions for webcast
```

3.5.2. how to setup a new active city

> **Warning**
>
> These notes assume you already have active running atleast one city on your server. They're also out of date. Sorry about that.

Permissions needed to do it:

- a login to cat with membership of the *active* , and *geeks* groups (setup with `usermod -G active,geeks <username>`)
- access to the postgresql database as a user on cat with permissions to create databases(`createuser <username>`)

- super user on cat to setup mailing lists for the city (use `makelist` and `makelistarchive`)

If the city will be hosted on cat.org.au also known as active.org.au, login to cat, and then run the commands:

```
cd /www/active
./new_city cityname
```

The *new_city* script checks out copies of the code, puts them in the new cityname folder (/www/active/cityname), also it creates a new copy within CVS of the local folder for the new city (active/cityname/local, copied from the active/template city) and checks it out into /www/active/cityname/local. Certain folders and files are created and marked with open permissions so the web server can write to them (e.g. the calendar text databases, and the webcast uploads folder). It then creates a new SQL database active_cityname.

There is also a need to setup the mirroring of multimedia content. This is not yet automated by the new_city script. We do this by making each cities' uploads folder a link into a folder named after the city in /www/active/uploads, e.g. for a new city

```
cd /www/active/uploads
mkdir cityname
chmod 777 cityname
cd /www/active/cityname/local/webcast
rm -r uploads
ln -s /www/active/uploads/cityname uploads
```

The new city will be then available at the web address active.org.au/cityname

3.5.2.1. Things to customise for a new active site:

- local/appear.inc defines the colour scheme for the site
- the local/include folder has the front page, and the headers and footers for all pages. Specifically, edit *top1.inc* and *top2.inc* to change the HTML that appears at the top

of the generated file (e.g. a banner at the top of every page, or a left hand sidebar). And edit *bottom.inc* to change what appears at the bottom of the HTML (e.g. a right hand side bar or a footer). You can edit local/webcast/webcast.inc to change the way these include files are used, or add new ones - see the top() and bottom() functions, which get called anywhere a page is being generated in the webcast code. E.g.

index.php3 -symlink to-> front.php3 -> which then calls:

- top($title) [function defined in local/webcast/webcast.inc] which writes the HTML header (HTML / BODY tags) then includes:
  - local/include/top1.inc
  - the page heading $title as passed to the top() function
  - local/include top2.inc
- local/webcast/cache/frontN.html which is page N of a dynamically generated list of summaries of the stories (you can refresh these by calling webcast/refresh.php3 from your web browser - don't edit them manually as your changes will get overwritten)
- bottom() [function defined in local/webcast/webcast.inc] which includes
  - local/include/bottom.inc

  and then writes the HTML footer (/BODY and /HTML tags)

- the local/images folder has the logo that goes in the top right hand corner, and some other images we use
- you will want to set the passwords in local/password.php3
- local/calendar/calendar.setup will definitely need attention -- make sure you change the announce email address!
- local/webcast/uploads is where uploaded multimedia stories end up (including text stories longer than 6K). You may need to make sure this folder gets mirrored onto your multimedia server (e.g. a realmedia server).

- please put anything specific to the local city in the local folder somehow, and only put code that would be applicable to any installation of active into the other folders such as webcast. this makes it *much* easier to share enhancements to the code, and upgrade to new versions of the software. e.g. if you are adding a new page such as a list of allies, provide a warpper script to call it in webcast, but provide the specific details in the local/include folder, and include it from the wrapper script.

There is some pretty reasonable documentation of the calendar script in the calendar/Documentation folder, unfortunately, we haven't updated it to reflect any changes we made to it from the Extropia version.

3.5.3. how to setup a new indymedia city

> **Warning**
>
> These notes assume you already have active running atleast one city on your server. They're also out of date. Sorry about that.

Permissions needed to do it:

- superuser access on fs.freespeech.org to setup the domain name *cityname.indymedia.org*
- superuser access on emma / turtle to setup the apache virtual web server for *cityname.indymedia.org*
- a login to the web server to be used:
  - turtle with membership of the *web* group (setup with `usermod -G web <username>`) *OR*
  - login to emma with membership of the *imc* group (setup with `usermod -G imc <username>`)

- access to the postgresql database as a user on emma / turtle with permissions to create databases (setup with `createuser <username>` from a user that has postgres super user access, e.g. the user *postgres*)
- the loudeye.com login and password (one way to get it is given below, but only works if you have superuser on emma or turtle)

On cat each city has its config stored in CVS separately from the main code, so that it is easy to setup test verisons of a city's configuration on other machines or in different folders on the same machine.

To setup a new indymedia city, you would do this: *on stallman:*

```
cd /www/active-cvs
toplevel/new_city_imc cityname /www/active-cvs
```

[this pulls across the shared code in /www/active-cvs/cityname, and new city config using the template_imc indymedia style, via anonymous CVS from cat, and sets up a symlink folder /www/cityname which will be the entry point from the web, plus a few other things]

Then edit the DNS and the apache web server configuration to add the virtual server. The virtual server should point to /www/cityname.

Finally you will need to manually ftp into centerstage.loudeye.com (see /usr/local/sbin/pushtoeye for the password, but you need root access to read /usr/local/sbin/pushtoeye - because it has a password in it!). Anyway, you will need to ftp into there and create the cityname folder and a metafiles/cityname folder, which is needed for MP3 and image hosting.

3.5.3.1. How to setup the calendar within an indymedia city

If you have a default indymedia-style install of active, you can easily activate the web calendar:

```
$ cd /www/cityname
$ ln -s ../calendar .
```

Test it by running:

```
$ cd calendar
$ ./index.cgi
```

if it fails with:

```
$ ./index.cgi
Can’t locate Time/DaysInMonth.pm in @INC (@INC contains: ...
...
```

you need to install the CPAN Time::DaysInMonth module:

```
# perl -MCPAN -e shell;
cpan> install Time::DaysInMonth
cpan> quit
```

then test it again:

```
$ ./index.cgi
Content-type: text/html


    <HTML>
    <HEAD>
...
```

and when it prints a web page it means it’s working.

For an example layout, see the template indymedia calendar (http://template.indymedia.org/calendar/?display=front&days=8) (template.indymedia.org is a copy of what a default new indymedia city looks like). The newswire is on the left, and the events for the next week are listed on the right. Obviously we need to tweak the defaults. :) For a good looking active calendar site, see active sydney (http://active.org.au/sydney/).

You can control a lot of things in local/calendar/calendar.setup, the rest are in local/appear.inc, e.g. the colour settings are shared between the webcast newswire and the calendar. See below for details on customisation.

One important feature you might want to use is the *weekly email calendar*. You'll need a mailing list to send the calendar out to, which active doesn't do, so get one from an activist server or Yahoo groups or something. You can usually add a subscription box to your headers or sidebar on your site, to make it easy for people to sign up. Then choose someone to receive the calendar every week and pass it on to the list. Let's say their email address is: **someone@activist.org**.

Then you'll need to use **cron**, the Unix system scheduler, to generate the calendar at regular intervals. Get a shell prompt on the server that's running the active calendar, then type in:

```
$ export EDITOR=pico
$ crontab -e
```

(use another editor apart from pico if you like of course!). Then paste in 3 lines like the following:

```
0 18 * * 0 (cd /www/cityname/calendar; /usr/bin/perl \
/www/cityname/calendar/index.cgi dis-
play=mail_text days=15 | mail \
-s"Cityname Events" someone@activist.org)
```

Note that the sloshes "\" must be the very last character on the first 2 lines.

The above example will send out a calendar every Sunday (day zero - 5th number 1st line) at 6pm (18:00 - 1st two numbers, 1st line), using the active calendar installed in `/www/cityname/calendar`. The calendar will have events for the next two weeks (15 days), and will be sent to **someone@activist.org**. The subject line on the email will be "Cityname Events".

Note that you can also get events emailed to someone as announcements when they are entered into the website. That email address is set in `local/calendar/calendar.setup`.

3.5.3.2. Things to customise for a new indymedia site:

All file paths are relative to */www/cityname*.

- local/appear.inc defines the colour scheme for the site
- local/include has a bunch of files that get used to make up the pages and give them a consistent look:
  - local/include/bottom.inc - included at the bottom of every page
  - local/include/cities.inc - used in the sidebar to list other indymedia sites, usually is a symlink pointing to /www/cities.inc
  - local/include/feature.inc - not used in the indymedia style
  - local/include/front.inc - not used in the indymedia style
  - local/include/howto-publish.html - the text above the form on the publish page
  - local/include/imcfront-header.inc - the text at the top of the front page
  - local/include/imcfront.inc - the text in the middle column of the front page, usually used to list featured stories
  - local/include/sidebar.inc - the sidebar that appears on every page (including the front page)
  - local/include/top1.inc - not used in the default indymedia style, but if you put something in here, it will show up *before* the page numbers in the main body of every page
  - local/include/top2.inc - not used in the default indymedia style, but if you put something in here, it will show up *after* the page numbers in the main body of every page
- the local/images folder has the logo that goes in the top left hand corner (local/images/imcfront.gif), and the banner across the top of the front page (local/images/imcfront-banner.gif)
- you will want to set the story administration password in local/password.php3

- to change the window title and bookmark label (i.e. the <title> tag) edit local/webcast/webcast.inc
- local/webcast/uploads is where uploaded multimedia stories end up (including text stories longer than 6K, images and image thumbnails). You may need to make sure this folder gets mirrored onto your multimedia server (e.g. a realmedia server).
- major known problem with *timezones* : the times shown on the website are currently set by which web server is being used. emma is on US east coast time, and turtle is on US mountain time.
  - *Advanced users* : You can edit local/webcast/webcast.inc to change the way these include files are used, or add new ones - see the top() and bottom() functions, which get called anywhere a page is being generated in the webcast code. E.g.

    index.php3 -symlink to-> front.php3 -> which then calls:

    - top($title) [function defined in local/webcast/webcast.inc] which writes the HTML header (HTML / BODY tags) then includes:
      - local/include/top1.inc
      - the page heading $title as passed to the top() function
      - local/include top2.inc
    - local/webcast/cache/frontN.html which is page N of a dynamically generated list of summaries of the stories (you can refresh these by calling webcast/refresh.php3 from your web browser - don't edit them manually as your changes will get overwritten)
    - bottom() [function defined in local/webcast/webcast.inc] which includes
      - local/include/bottom.inc

      and then writes the HTML footer (/BODY and /HTML tags)

  - please put anything specific to the local city in the local folder somehow, and only put code that would be applicable to any installation of active into the other folders such as webcast. this makes it *much* easier to share enhancements to the

code, and upgrade to new versions of the software. e.g. if you are adding a new page such as a list of allies, provide a wrapper script to call it in /www/cityname, but provide the specific details in the local/include folder, and include it from the wrapper script.

Chapter 4. Active Software Development

Active is a free software project under the GNU Public License (http://www.gnu.org/) (also known as copyleft). See www.active.org.au/source (http://www.active.org.au/source/) for the latest version of this software.

You are invited to help develop new features and fix problems. Hopefully this chapter will help explain how best to do that.

This chapter is $Revision: 1.7 $. The last update was on $Date: 2001/05/19 12:47:24 $, at which time the current release version of active was 1-7-0cvs .

4.1. Contributing to the active software development

There is a separate list for discussing and organising the programming of the active software, called *webcoders*. To get on it, send a blank email to geton.webcoders@cat.org.au (mailto:geton.webcoders@cat.org.au). Or visit the archive (http://cat.org.au/lists/webcoders/).

There is a lot of more up to date info at tech.indymedia.org (http://tech.indymedia.org/).

We have a to-do list using the bugs tracking system at sourceforge (http://sourceforge.net/project/?group_id=819). If you're really keen, you can join the webcoders-cvs list (geton.webcoders-cvs@active.org.au (mailto:geton.webcoders-cvs@cat.org.au)) where you will be barraged with automated logging emails everytime someone is hacking on the code. There is a digest version on the indymedia servers, also called webcoders-cvs (http://lists.indymedia.org/mailman/listinfo/webcoders-cvs).

4.2. Key design ideas

- any story can be linked to any other. e.g.
  - a picture can be linked to a report about the same thing
  - a still image from a video can be linked to the video
  - a high quality MP3 audio file can be linked to a lower quality realaudio version
  - a series of pictures from the same event can be linked

  this is the best way we could think of letting people collect related webcast stories together, yet still keep the flow of new stories coming. The logic is, if people have to wait until they've got all the related stories together, before posting them in a bunch, or figure out what keywords to use, it just slows people dowm, and when you slow people down who are publishing for a live webcast, that means you end up simply missing out on a lot of stories.

  There is probably a lot more that could be done here. But this seemed to work quite well at both J18 and N30.

- cache pages that are heavy duty database queries - the frontN.html pages require quite a nasty SQL query because of the completely general (and probably vastly improvable) way we have implemented story linking
- the scripts in the actual webcast folder are kept as general possible; anything specific to a particular city or eevent should go in the ../local/ folder. This helps in two ways (1) people wondering what to customise first can go to the local folder, (2) it helps us developers manage changes to the software.
- use relative links always - in both the URLs and within scripts for including files - in fact there are no absolute paths ideally - this gets a bit tangly, but makes it heaps easier to setup new sites for new cities or events, and also new sites for testing, or for different developers to play with, because you can put the active folder wherever you want to in the web tree and it still works.

If you want the webcast folder to be top-level, you can do it with symlinks, but i think are nicer ways to do it, like having the front page do a static include of active/local/webcast/cache/front0.html, although you do need to munge the relative links in that case, which is what we have done for active.org.au/sydney and friends.

- config files with passwords in them end in .php3 so they can't be snooped over the web
- all the uploaded files go in the one messy folder. That includes text files for stories longer than 8192 bytes (a PostgreSQL hard limit on transaction length - *warning* postgresql can trash your database if you throw long queries at it. Use VACUUM to recover it.) It also of course includes images, their automatically generated thumbnails, realaudio, mp3's, realvideo, etc. The new_data-process.php3 script which processes the publish.php3 form, makes sure any new files get a unique name. We got several hundred megs during N30, including some very big movie files (encoded at multiple quality levels in RealMedia).
- pushing files (realmedia, images) to different servers has to be handled outside these scripts, because it varies a lot depending on what setup, security etc. the mirror servers have. rsync is I think the absolute bees knees for mirroring files, because it handles a lot of niggles automatically, but you can use FTP to push files as well. I'd still like to see some sort of wrapper for both rsync and ftp that writes a lock file, so that if you have a short wait time between mirroring (e.g. 10 minutes) a second copy doesn't start up during a long transfer. You could also use a daemon, but I had trouble with one daemon I wrote in perl (webcastd.pl) mysteriously dying. cron seems much more reliable.

4.3. Code Versioning System

Many people from around the world are contributing to the active software. This takes some co-ordination! We use the Code Versioning System (CVS) software to help us manage multiple versions of the code, and to make it easier for more than one person to work on the code at the same time.

CVS takes a little getting used to. However, it is very powerful, and allows us to be much more effective as a group.

I want to use this section to write up a lot of specific details about how to use CVS for programming on active. Meanwhile, I just ran across an excellent description of CVS (http://lists.debian.org/debian-devel-0102/msg00824.html) from a Debian developer. Scroll down 3 pages to get to the part of the email that talks about CVS. It looks like that project uses a very similar approach to CVS. I'll be using his description as a starting point for writing this section.

You can browse the active code CVS tree online at cvs.cat.org.au (http://www.cvs.cat.org.au/).

4.4. Releasing a new version of active

The latest version of the code is available to anyone at any time through CVS. However, it is useful to mark particular versions with a release number, so we can refer to them more easily, and to help document changes in the code.

It can be tricky to decide when a new release is needed. The general free software approach is "release early, release often." So if in doubt, it probably doesn't hurt to release, although it does take a few moments to do. In fact, that's even more motivation for more frequent releases: the longer we leave it, the longer it takes to write up the release notes!

Specifically, I think a new release is needed when:

- a lot of changes have been made to the code, without introducing signifigant new bugs or surprises to people upgrading (atleast not undocumented surprises) - however, if you're not sure whether the version works or not, a release is still often a good idea, because if you wait then new features may get added with new bugs before the existing bugs are fixed
- there is an important new feature or bugfix
- some time has elapsed since the last release

- the database schema has changed (this should be indicated by increasing the second number in the version number, e.g. 1.7.3 to 1.8.0)

Because we are always guessing when is a good time to make a release, some releases might be bad. That's part of open source - hopefully our documentation can highlight which versions are good, which are bad, and which introduce features or bugfixes.

These are the steps I go through when releasing a new version of the code:

1. ChangeLog: Ideally you could just generate a ChangeLog straight away. However, as of April 2001 there are corrupt headers in some of the files in the active CVS repository.

   So the workaround is to lookup the archives of the webcoders-cvs list to see which folders in the code have changed since the last release (ignore changes to city specific local folders, e.g. active/sydney/local, but do include changes to template or template_imc as these are the defaults for new active and indymedia cities). Then checkout a temporary workarea that includes all these folders. E.g. for release 1.7.0 the folders active, active/shared, active/webcast, active/template_imc, active/doc, active/source have all had changes since the last release, so in my home folder I ran:

   ```
   cvsa co -l active
   cd active
   cvsa co -d shared active/shared
   cvsa co -d webcast active/webcast
   cvsa co -d template_imc active/template_imc
   cvsa co -d doc active/doc
   cvsa co -d source active/source
   ```

   You can then generate a ChangeLog using cvs2cl.pl (http://www.red-bean.com/cvs2cl):

   cvs2cl.pl --usermap usermap --gmt

   cvs2cl.pl uses the file `usermap` to convert cat login names (used for write permission to CVS) into email addresses. Have a look in the ChangeLog to see if any of the latest changes are by new people not listed in `usermap`, if so, add them in, and re-run cvs2cl.pl as above.

Since the ChangeLog shows the newest changes first, you can select the first part of the file up until the previous release, which will be marked with a CVS comment like this:

```
* shared/release-version.txt: release version 1-62 of active
```

The cvs2cl website has some good notes on how to write good CVS commit messages (http://www.red-bean.com/cvs2cl/changelogs.html) with the ChnageLog in mind.

2. CHANGES: You can use the changelog to write a summary of the changes for people who don't have time to read through it all. It's important to highlight: new features, bugfixes, known bugs, database schema changes, and special steps needed to upgrade.

   Paste the summary and the ChangeLog since the last release into active/source/CHANGES, then commit that change to CVS.

3. Login into the cat CVS server (cvs.cat.org.au) and run the actual release script:

   ```
   cd /www/active
   cvs update source/CHANGES
   ./release x-y-z
   ```

   where x-y-z is the version number to release, e.g. 1-7-0 (we can't use 1.7.0 here, because CVS release tags can't have dots in them). The release script does a few things:

   - Opens source/CHANGES in **vi** for you to edit (in these instructions we've already done that, so you can just quit vi using **:q[enter]**)
   - Writes the release version into `shared/release-version.txt`, then commits the changed release-version.txt and source/CHANGES to CVS.
   - Tags the current version of all source and template files in CVS with the tag: relx-y-z
   - Exports a copy of the code and makes a tar file, copies it into the source folder so people can download it. Writes the current.tar.gz symbolic link to point at the

new version.

- Reminds you to announce the new release on webcoders and freshmeat.

You can repeat the release script with the same verison number if you got something small wrong that you need to fixup. Just quit straight out when CHANGES comes up in an editor, and answer "y" when asked if you want to overwrite the tar file. It will generate a couple of extra CVS commit messages as it switches the release-version.txt version number back and forth, but that’s OK.

4. Do as the script says: write an email to webcoders@cat.org.au (blind CC to the big users of the software at imc-tech@indymedia.org and webkids@cat.org.au). At some point we will probably want to setup a list just for announcements of new releases. (A list for general users might be good too.) And announce on freshmeat.net (ask maffew for the freshmeat login which controls the active project lsiting on there).

   You will need an even more summarised list of changes for freshmeat, approximately 1-2 paragraphs. Include this at the top of the webcoders email too.

Appendix A. Editing This Guide

This guide is written in Standard Generalized Markup Language (SGML) DocBook. This is a bit like HTML, but it has better features for longer documents. It can be used to generate contents, cross-references and indexes, multipage HTML manuals, PDF or postscript, and a bunch of other things. For a rough and ready quickstart read Get Going With DocBook (http://nis-www.lanl.gov/~rosalia/mydocs/docbook-intro/docbook-intro.html).

Unfortunately, SGML is less forgiving than HTML. So there are a bunch of tools that make it easier. See the LDP Author Guide (http://www.linuxdoc.org/LDP/LDP-Author-Guide/)for details. If you're running Debian Linux, you can get a lot of useful tools by installing the packages: sgmltools-2, docbook, docbook-stylesheets, and cygnus-stylesheets.

For editing SGML, I've found using emacs in psgml mode is the most convenient, because it helps you insert valid tags in the right context, and will verify your tags on the spot. It even promises to have colour syntax highlighting, but I haven't figured out how to get that to work yet. I had a little trouble getting xemacs to work with psgml, until I upgraded to a newer version of emacs21-basesupport - version 2000.10.23-1 (http://packages.debian.org/testing/editors/xemacs21-basesupport.html) available in Debian testing.

Useful emacs tricks for SGML editing include:

- the first time you work on the document, load active.sgml and use the menu **DTD -> Save Parsed DTD** to save `active.ced` which makes work much quicker and allows the subfiles to be edited in SGML mode
- **Ctrl-C Ctrl-E** inserts valid tag pairs in a given context and **Ctrl-C Ctrl-R** wraps selected text with a tag, and both have **Tab** completion for the SGML tags, which is lovely; also handy is **Ctrl-C /** which finishes an open tag.
- use **Tab** to indent tagged text to show the nesting level, and **Meta-Q** to fill or word-wrap paragraphs (the **Meta** key took me a while to figure out: **Esc** always works, but sometimes you can use **Alt** or the **left-hand Windows key**). Turn on auto

word wrapping using **Meta-X auto-fill-mode**.

- explore the menus, I've found the following especially useful: **SGML -> Validate** (you will be prompted for the file to validate - you want to validate from the top file: `active.sgml`), **Markup -> Tag region** tags an existing area of text

To compile SGML into a set of HTML files, you can use:

```
$ db2html -d $PWD/active.dsl active.sgml
```

Note that you can edit `active.dsl` to change how the document is displayed or printed, e.g. how many levels to show in the table of contents, what background colour to use, etc.

One common task in dealing with SGML is converting an existing document from HTML into SGML. More specifically, we need to go from HTML to SGML DocBook. I've found a couple of different ways to do this:

- Using the **html2sgml** perl script you can go from HTML to SGML LinuxDoc. To get from LinuxDoc to DocBook (both SGML variants with different Document Type Definitions (DTDs)) you can use **sgmltools -b ld2db**.
- I just found a much newer package called html2db (http://freshmeat.net/projects/html2db/) that promises to convert straight from HTML to SGML DocBook. This uses openjade, which is not in Debian 2.2, but it is in Debian testing & unstable. I found I could get something half-decent to work using jade: **jade -t sgml -d ~/html2db/html2db.dsl < filename.html > filename.sgml** where `html2db.dsl` is from html2db.

  However, this leaves the > tag dangling over the end of the lines. I used this perl script to clean up the output from the above hack to make it easier to read and edit: **perl -pi.bak -e '$/ = undef; s/\n>/>\n/g' filename.sgml**

For more than you ever wanted to know about DocBook, see DocBook: The Definitive Guide (http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html).

This appendix is $Revision: 1.5 $. The last update was on $Date: 2001/06/11 04:57:53 $, at which time the current release version of active was 1-7-0cvs .