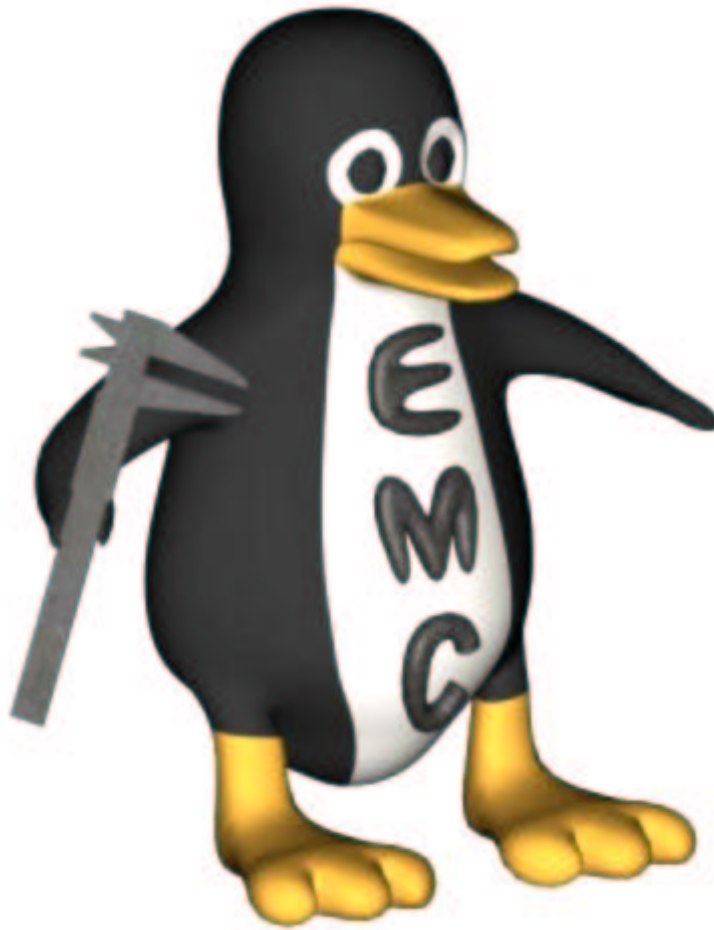


The Enhanced Machine Control Developer Handbook



The EMC Team¹

20th July 2003

¹Ray Henry <rehenry@up.net> serves as the general editor. Authors include Will Shackleford, Fred Proctor, Jon Elson, Don McLain.

Herein follows a collection of notes and comments on the EMC source code. Much of it is taken directly from the multitude of C sources, and some cribbed from various web sites and posts. This document should be considered "work in progress" and may contain errors.

Copyright (c) 2000 - 03 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and one Back-Cover Text: "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

Contents

1	Becoming a Developer	9
1.1	My Sourceforge account.	9
1.2	Becoming an EMC developer.	9
1.3	Becoming a SSH client.	9
1.3.1	Downloading ssh.	10
1.3.2	Setting up ssh	10
1.4	Sourceforge checkout.	10
1.5	Updating your copy of the repository	11
1.5.1	Update	12
1.5.2	Update codes	12
1.5.3	Making a change to the sourceforge repository.	14
1.5.4	Adding files to a directory	15
1.5.5	Checkout	16
1.5.6	Remove	16
1.6	CVS Usage	17
2	EMC Source Code Introduction	21
2.1	src/drivers	21
2.2	src/emcnml	22
2.2.1	inivar	22
2.2.2	emcnmlsize	22
2.2.3	canon.hh	22
2.2.4	emc.cc	22
2.2.5	emc.gen	22
2.2.6	emc.hh	22
2.2.7	emcargs.cc	23
2.2.8	emccfg.h emcglb.c	23
2.2.9	emcglb.h	23
2.2.10	emcnmlsize.cc	23
2.2.11	emcops.cc	23
2.2.12	extintf.h	23
2.2.13	extpvt.c	23
2.2.14	getinput.c getinput.h	23

2.2.15	iniaux.cc iniaux.hh iniaxis.cc iniaxis.hh inicool.cc inicool.hh inilube.cc inilube.hh inispin.cc inispin.hh initool.cc initool.hh initraj.cc initraj.hh	24
2.2.16	inivar.c	24
2.3	src/emcmot	24
2.3.1	usrmot	24
2.3.2	stgmod.o	24
2.3.3	stg8mod.o	24
2.3.4	stg2mod.o	24
2.3.5	stg_v2_8axis_mod.o	25
2.3.6	steppermod.o	25
2.3.7	freqmod.o	25
2.3.8	smdromod.o	25
2.3.9	simmod.o	25
2.3.10	minitetra.o	25
2.3.11	cubic.c cubic.h	26
2.3.12	dro.c dro.h	26
2.3.13	kinematics.h	26
2.3.14	emcmot.c emcmot.h	26
2.3.15	emcmotcfg.h	27
2.3.16	emcmotglb.c emcmotglb.h	27
2.3.17	emcmotlog.c emcmotlog.h	27
2.3.18	emcsegmot.c	27
2.3.19	emcstepmot.c	28
2.3.20	extsmdromot.c	28
2.3.21	extsmmot.c	28
2.3.22	extstgmot.c	28
2.3.23	genhexkins.c genhexkins.h	28
2.3.24	lsutil.c lsutil.h	28
2.3.25	mmxavg.c mmxavg.h	29
2.3.26	pid.c pid.h	29
2.3.27	puma560kins.c puma560kins.h	29
2.3.28	segmentqueue.c segmentqueue.h	29
2.3.29	stg.h stg.c stg2.c stg2.h stg_v2_axis8.c stgdiag.c	29
2.3.30	tc.c tc.h	30
2.3.31	testlsutil.c	30
2.3.32	testtp.c tp.c tp.h	30
2.3.33	trivkins.c	30
2.3.34	usrmot.c	30
2.3.35	usrmotintf.c usrmotintf.h	30
2.4	src/emcsim	30
2.4.1	amplifier.c amplifier.h	30
2.4.2	demotor.c demotor.h	31
2.4.3	encoder.c encoder.h	31

2.4.4	extsimaio.c extsimdio.c extsimmot.c	31
2.4.5	inisim.cc	31
2.4.6	sim.h simaio.c simdio.c simmot.c	31
2.4.7	simmot_n.cc simmot_n.gen simmot_n.h	31
2.5	src/emcio	32
2.5.1	emcio.hh	32
2.5.2	bridgeportaux.cc bridgeportcool.cc bridgeportlube.cc bridge- portspin.cc bridgeporttool.cc	32
2.5.3	minimillaux.cc minimilltool.cc	32
2.5.4	extbridgeportio.c extminimillio.c	32
2.5.5	iosh.cc	32
2.5.6	tkio.tcl	32
2.6	src/rs274ngc	33
2.6.1	canon_stand_alone.cc	33
2.6.2	driver.cc	33
2.6.3	rs274ngc.cc rs274ngc.hh	33
2.6.4	rs274ngc.var	33
2.7	src/emctask	33
2.7.1	emcpanel (built in emc/src/emctask)	33
2.7.2	inimot (built in emc/src/emctask)	33
2.7.3	bridgeporttaskintf.cc minimilltaskintf.cc	34
2.7.4	debug.cc	34
2.7.5	tkemc.tcl	34
2.7.6	genedit.tcl	34
2.7.7	tkbackplot.tcl	34
2.7.8	emccalib.tcl emclog.tcl emctesting.tcl emctuning.tcl . .	34
2.7.9	emccanon.cc	35
2.7.10	emcdummy.cc	35
2.7.11	emcpanel.cc	35
2.7.12	emcsh.cc	35
2.7.13	emcsvr.cc	35
2.7.14	emctask.cc	36
2.7.15	emctaskmain.cc	36
2.7.16	inimot.cc	36
2.7.17	keystick.cc	36
2.7.18	xemc.cc yemc.cc	36
3	Hardware Drivers	37
3.1	Wrapper Functions	37
3.2	Hardware Functions	39
4	Makefiles	43

5	Customising the software	45
5.1	Re-compiling EMC along with various flags	45
5.2	Feed Per Rev Example	45
5.3	Customising NML	46
6	EMCSH	47
6.1	Adding Commands	47
6.2	Working with emcsh	47
6.3	Tcl command extensions for EMC	47
6.4	EMC commands:	48
6.5	Additional Comments	52
7	IOSH	53
7.1	Adding Extra Inputs and Outputs.	53
7.2	Working with iosh	54
7.3	EMC IO commands:	55
	7.3.1 IO status commands, set associated field in the NML status structure	55
	7.3.2 IO commands:	56
8	TKIO	57
8.1	Introduction	57
8.2	Start Tickle Using	58
8.3	IniLoad	58
8.4	Parport I/O	59
	8.4.1 parportSetBit <bit> <val>	59
	8.4.2 parportCheckBit <bit>	59
	8.4.3 parportGetBit <bit>	59
8.5	doit	60
	8.5.1 Global Variables	60
	8.5.2 NML	60
	8.5.3 Matching the command string	61
	8.5.4 Running the controllers	61
	8.5.5 Housekeeping	62
8.6	Group Controllers	62
8.7	Group Processes	63
8.8	Widgets	66
	8.8.1 Create the main window top frame	66
	8.8.2 Build the top menu bar	66
	8.8.3 Create frames to hold label and value pairs and create labels	66
	8.8.4 Use to pack geometry manager to place widgets in top frame	67
	8.8.5 Adding widgets	67
8.9	Process List from tkio in the order written	68

<i>CONTENTS</i>	7
8.9.1 parport processes	68
8.9.2 Tool processes	68
8.9.3 Spindle processes	68
8.9.4 Coolant processes	68
8.9.5 Estop processes	69
8.9.6 Lube processes	69
8.9.7 Misc processes	69
A README	71
A.1 Organization of this directory	71
A.2 Master L ^A T _E X Files	71
A.3 Chapter Files	71
A.4 General Files	72
A.5 L ^A T _E X Versions	72
A.6 L ^A T _E X Notes	72
A.6.1 Path Names	72
A.6.2 Thoughts for contributors	72
A.6.3 Image conversion	72
A.6.4 Fonts	73
A.6.5 White space	73
A Legal Section	75
A.1 GNU Free Documentation License Version 1.1, March 2000 . .	75
A.1.1 GNU Free Documentation License Version 1.1, March 2000	75
B Index	81

Chapter 1

Becoming a Developer

1.1 My Sourceforge account.

The development version of EMC exists on sourceforge.net so I needed to get a free user account there. Soon after I asked for an account, I received an email with a url for my final sourceforge setup. I had to enable cookies on my browser while I do this and must allow them anytime while I'm using the sourceforge service.

1.2 Becoming an EMC developer.

As soon as I got that far, I emailed Will my login name so that he could set me up as a developer. (Now you can email any one of the people listed as administrators and they can add you.) Will did that and after a few minutes (hours) I was listed. Then the first thing I did was trash my developer status by a wrong click. (Sorry for the second request Will) After a bit I was a developer again. So the next thing I tried was a secure login and there I was.

The secure login got me to my personal page. From there I searched for EMC and selected the correct one. I added messages and news and thought I was all set. I went back to my personal page and saw those added things. But when I went to the cvs repository for EMC, I was immediately lost in deep stuff. Searching around sourceforge for information was very little help!

1.3 Becoming a SSH client.

The first question I had was how do you send the commands listed on the sourceforge cvs page. My thought was that perhaps, with the secure login, sourceforge had somehow taken over my computer so I started a virtual terminal and just typed in the stuff listed there. I got back a complaint so I read that documentation again.

For all developer (read/write) access, you will be using SSH. SSH (1.x) client must be available to your local machine. The local environment variable CVS_RSH must also be set to the path to ssh. This is done on most linux (bash) systems by typing:

```
export CVS_RSH=ssh
```

Anonymous CVS access uses CVS pserver and does not require SSH. If you get 'permission denied' errors with no prompt for a password, you do not have this environment variable set properly or SSH is not available to your system. Fix this before suspecting a password problem.

Since the complaint was something about ssh, I started reading and found out that I needed something called SSH. A quick check of my system revealed that I had no commands called ssh, no files named ssh, and no package around that said anything about ssh.

My fall back plan is an altavista search where I came up with this link. <http://www.ssh.org/> Reading there confused me because it says very little about USING the thing that they call a secure terminal. I did learn that ssh is some sort of secure terminal that can be used to identify me and allow me to do work on a remote computer as a "trusted" user (little do they know).

A little more altavista turned up a site that provides very good links to the whole world of ssh. <http://www.heimhardt.de/htdocs/ssh.html>. Blessings on this fellow because he has provided a bunch of links and a bunch of info (much of it I did not understand, and I do not understand it yet).

1.3.1 Downloading ssh.

From the site above, I found a binary download for Linux 2.2.* and untarred it at the root directory. It installed several files in the /usr/local/bin directory. I considered that progress, because now when I entered the command ssh, I got a pile of options on the screen. Still a bit lost I went back to the web site above.

1.3.2 Setting up ssh

From the heimhardt.de site, I found this site: <http://www.tac.nyc.ny.us/~kim/ssh/> which provides a very good introduction to what setting ssh up on your system is about. So I followed the instructions given for setting up a public key and copied it into the file that allows remote access to my machine. This is a very mysterious process but it did provide the files that they said it should.

1.4 Sourceforge checkout.

I know from what little concurrent version system work that I'd done around home that in order to participate fully with it I would need a current version of the repository on my computer. I returned to the cryptic

documentation on sourceforge. I started this process by a secure login using my netscape browser because it doesn't have a problem with ssl. Then from my personal page at sourceforge I clicked on the EMC project near the bottom of the page.

Hint. If you answer their questionnaire, it will be reduced to a small block and the entire page will fit on a single screen.

The sourceforge document says:

```
How to check out source via SSH
Type the following, making necessary obvious substitutions
for your username and project.
cvs -z3 -d:ext:loginname@cvs.yourproject.sourceforge.net:/cvsroot/yourproject
co directoryname
```

This did not turn out quite like I expected because what's obvious to them is still muddy to me. I got the `-d:ext:loginname` part okay. My name at sourceforge is rayhenry so that gets substituted right after the `(-d:ext:)`. I got some further clues from the cvs page itself so I started my virtual terminal again (konsole) and made sure I was using bash by just typing in bash. Then I entered the line:

```
export CVS_RSH=ssh
```

and I was back to my prompt. Then I entered the next line but the obvious substitution of `directoryname` took a couple of tries because it is lowercase.

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co emc
```

after a bit, sourceforge asked for my password. "Wow! Now I'm getting somewhere."

This checked out the entire structure and files of the emc part of the NIST system onto my machine. So I did it again for the rcslib and documents directory and after a half hour my hard drive held a copy of the sourceforge repository.

1.5 Updating your copy of the repository

There are three commands that help you keep your repository in line with the sourceforge set of files. They are update, checkout, and remove.

1. *Update* works to compare each of the files in your local repository with the same file in the sourceforge repository. Any changes that have been made there will be updated on your local copy.
2. *Checkout* is required for any new files or directories that have been added to the sourceforge set. Occasionally you will need to use checkout to get the latest new files.
3. *Remove* is used to delete files from the repository. If you have files on your system that no longer exist in the sourceforge set you will get a warning message when you update.

1.5.1 Update

Once you have a working copy of the repository you can log into sourceforge, start a terminal, issue the command *export*

```
CVS_RSH=ssh
```

and update all of a module by using the same command indicated above to check out the module. You could replace the *co* in that command with *update*. You can also *cd* into the directory that you wish to update and issue the command

```
cvs update
```

This means that you can move anyplace in your copy and update just that directory.

I like to update my copy of the whole repository at one time. On my workstation I use a main directory named *emcdevelop* so I am in the habit of issuing the following set of commands after I have my login to sourceforge confirmed.

```
export CVS_RSH=ssh
```

```
cd emcdevelop
```

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co emc
```

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co emc
```

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co emc
```

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co emc
```

Once a local copy of the repository has been established, *cd* works the same as *update*.

You could also update the entire repository by a procedure like this.

```
export CVS_RSH=ssh
```

```
cd emcdevelop/emc
```

```
cvs update
```

```
cd ../emc-HAL
```

```
cvs update
```

```
cd ../rcslib
```

```
cvs update
```

```
cd ../documents
```

```
cvs update
```

If your development efforts are primarily in the *documents* module, and you are working with *Lyx* you might only update that directory using a command like

```
export CVS_RSH=ssh
```

```
cd emcdevelop/documents/lyx
```

```
cvs update
```

1.5.2 Update codes

After you've run *checkout* to create your private copy of source from the common repository, other developers will continue changing the central

source. From time to time, when it is convenient in your development process, you can use the 'update' command from within your working directory to reconcile your work with any revisions applied to the source repository since your last checkout or update. While update is working, it will give you a list of the files that it looks at and will report any changes, conflicts, etc. The following info page describes that report.

update output

The commands *update* and *checkout* keep you informed of their progress by printing a line for each file, preceded by one character indicating the status of the file:

'U FILE' The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven't changed but are not the most recent versions available in the repository.

'P FILE' Like 'U', but the CVS server sends a patch instead of an entire file. These two things accomplish the same thing.

'A FILE' The file has been added to your private copy of the sources, and will be added to the source repository when you run 'commit' on the file. This is a reminder to you that the file needs to be committed.

'R FILE' The file has been removed from your private copy of the sources, and will be removed from the source repository when you run 'commit' on the file. This is a reminder to you that the file needs to be committed.

'M FILE' The file is modified in your working directory.

'M' can indicate one of two states for a file you're working on: either there were no modifications to the same file in the repository, so that your file remains as you last saw it; or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory.

CVS will print some messages if it merges your work, and a backup copy of your working file (as it looked before you ran 'update') will be made. The exact name of that file is printed while 'update' runs.

'C FILE' A conflict was detected while trying to merge your changes to FILE with changes from the source repository. FILE (the copy in your working directory) is now the result of attempting to merge the two revisions; an unmodified copy of your file is also in your working directory, with the name `.'.#FILE.REVISION'` where REVISION is the revision that your modified file started from. Resolve the conflict as described in Conflicts example. (Note that some systems automatically purge files that begin with `.'.#'` if they have not been accessed for a few days. If you intend to keep a copy of your original file, it is a very good idea to rename it.) Under VMS, the file name starts with `'_'` rather than `.'.#'`.

'? FILE' FILE is in your working directory, but does not correspond to anything in the source repository, and is not in the list of files for CVS to ignore (see the description of the '-I' option, and `cvsignore.`).

1.5.3 Making a change to the sourceforge repository.

So there it is. The connection has been made. The download is complete. I am a developer. Now I need to learn a little bit about using CVS lest I trash the entire repository and look to all the world like the idiot that my close friends already know me to be.

You may work on those files with your favorite text editor and save changes. One thing that I found out is that sometimes my editor leaves backup and invisible files laying around after I close it. Another thing that I've noticed is some cases where a comparison between versions of a file will show differences between Microsoft end-of-line versus unix end-of-line characters. Or just putting a space in a line will cause diff to think that the line has been edited. So I'll need to be a little careful with these kinds of editing problems so that others can see the real differences that I've made to a file. Okay now that I've made changes that will add a feature that all of us want, I need to update the repository at sourceforge with the revised file and any new files and directories.

The sourceforge documentations says:

```

    After initial checkout, you can change into this directory and
    execute cvs commands without the -d tag. For example:
    cvs update
    cvs commit -m "comments for this commit"
    or
    cvs add myfile.c
  
```

I am going to do this by following the commands that Will sent me that are intended to commit some of my Tcl/Tk work to the repository and make these changes a part of the next release of EMC. First I'll copy my email to Will that describes what I've done.

```

> I've been working with running scripts from a variable menu under
> tkemc. For this I used a sub directory called emc/scripts. I see that
> you already have a scripts directory in the release that contains a lot of
> the emc setup files.
>
> I'd like to see these tcl scripts implemented as a part of the release.
> What do you recommend that we do for a directory name for these kinds
of
> things.
  
```

Will responded

```

>Currently only developers see the scripts directory. When we build a
>distribution the script files for the appropriate platform are
>moved to the top level directory before the distribution is built.
>But end-users will probably not want the tcl scripts in the top
>level directory.
>Perhaps instead of using emc/scripts we could put it in a directory
>emc/tcl or emc/tclscripts.
  
```

I suggested that we expand on this just a bit with:

```

emc/tcl/lib
emc/tcl/scripts
  
```

since this would allow us to expand the tcl based stuff without adding much clutter to the main emc directory. Then all I had to do to my genedit

and tkemc files was change the directory reference in the scripts menu setup code.

Will responded with the following set of commands to use after the ssl log in and the bash shell in konsole.

```
export CVS_RSH=ssh
cd emc
mkdir tcl
cvs add tcl
cd tcl
mkdir lib scripts
cvs add lib scripts
cd scripts
cvs add * ( Copy all your scripts here. )
cd ../../scripts/<PLAT> (edit the emc.inc file to add emc/tcl/lib &
scripts)
cd ../../
cvs commit
```

He also suggested that I edit the file emc/scripts/emc.inc and add the new directories because, "That way the files would automatically be added the next time we build a distribution." (Will)

I got hung up somewhere in the cvs commit command. Sourceforge kept insisting that my files had been added but I couldn't see them when I looked in the emc/tcl/scripts directory. After a bit of reading of the cvs documentation I found this helpful page.

1.5.4 Adding files to a directory

To add a new file to a directory, follow these steps.

- You must have a working copy of the directory. Getting the source.
- Create the new file inside your working copy of the directory.
- Use 'cvs add FILENAME' to tell CVS that you want to version control the file. If the file contains binary data, specify '-kb' (Binary files.).
- Use 'cvs commit FILENAME' to actually check in the file into the repository. Other developers cannot see the file until you perform this step.

You can also use the 'add' command to add a new directory.

Unlike most other commands, the 'add' command is not recursive. You cannot even type 'cvs add foo/bar'! Instead, you have to

```
$ cd foo
$ cvs add bar
- Command: cvs add ['-k' kflag] ['-m' message] files ...
```

Schedule FILES to be added to the repository. The files or directories specified with 'add' must already exist in the current directory.

The added files are not placed in the source repository until you use 'commit' to make the change permanent...

The '-m' option specifies a description for the file. This description appears in the history log (if it is enabled). It will also be saved in the version history inside the repository when the file is committed.

For example, the following commands add the file 'backend.c' to the repository:

```
$ cvs add backend.c
$ cvs commit -m "Early version. Not yet compilable." backend.c "
```

I had successfully added the files to the repository but I had not successfully committed them. What I did was commit each file with the following set of commands. After I submitted each command, it asked me for my sourceforge password. (makes me wish I hadn't made every other letter a capital)

```
export CVS_RSH=ssh
cd emc/tcl/scripts
cvs commit -m "tkemc parport show script" IO_Show.tcl
cvs commit -m "tkemc script" MDI_Input.tcl
cvs commit -m "Experimental tkemc script" Subroutine.tcl
cvs commit -m "genedit gcode script" feedrate.ncw
cvs commit -m "genedit gcode script" gohome.ncw
cd ../../src/emctask
cvs commit -m "added script menu" genedit.tcl
cvs commit -m "tkemc popup" tkbackplot.tcl
```

But I forgot that I edited and added a copy of emc/scripts/emc.inc so I will have to commit that one as well or it will not be visible.

1.5.5 Checkout

When you have a working repository on your computer, checkout will compare your files with the sourceforge files and will download any new or revised files. You can run the command - cvs checkout emc - and it will update all of the emc files. I had to use the longer version to checkout all of the rcslib files.

```
cvs -z3 -d:ext:rayhenry@cvs.EMC.sourceforge.net:/cvsroot/EMC co rcslib
```

The same command with emc or documents will check out those modules as well.

1.5.6 Remove

The remove command is used to take files out of the repository. Normally I will not use this command at all because most of the files are not mine. But occasionally a file gets removed from the repository and I want to remove my local copy. Fortunately I can do this by just removing them from my local set and they will no longer be considered when updating. Since they have been removed from sourceforge they will not show up on the next checkout either.

If you want to remove some of your files from the sourceforge set, the following page tells how to do that.

cvs add Directories change. New files are added, and old files disappear. Still, you want to be able to retrieve an exact copy of old releases.

Here is what you can do to remove a file, but remain able to retrieve old revisions:

* Make sure that you have not made any uncommitted modifications to the file. Viewing differences, for one way to do that. You can also use the 'status' or 'update' command. If you remove the file without committing your changes, you will of course not be able to retrieve the file as it was immediately before you deleted it.

* Remove the file from your working copy of the directory. You can for instance use 'rm'.

* Use 'cvs remove FILENAME' to tell CVS that you really want to delete the file.

* Use 'cvs commit FILENAME' to actually perform the removal of the file from the repository.

When you commit the removal of the file, CVS records the fact that the file no longer exists. It is possible for a file to exist on only some branches and not on others, or to re-add another file with the same name later. CVS will correctly create or not create the file, based on the '-r' and '-D' options specified to 'checkout' or 'update'.

- Command: cvs remove [options] files ...

Schedule file(s) to be removed from the repository (files which have not already been removed from the working directory are not processed). This command does not actually remove the file from the repository until you commit the removal. For a full list of options, see Invoking CVS.

Here is an example of removing several files:

```
$ cd test
$ rm *.c
$ cvs remove
cvs remove: Removing .
cvs remove: scheduling a.c for removal
cvs remove: scheduling b.c for removal
cvs remove: use 'cvs commit' to remove these files permanently
$ cvs ci -m "Removed unneeded files"
cvs commit: Examining .
cvs commit: Committing .
```

1.6 CVS Usage

Much of the following has been extracted from <http://cvsbook.red-bean.com/cvsbook.html>

Tip: add the line

```
export CVS_RSH=ssh
```

to the .bashrc file in your home directory - Saves having to remember it each time.

CVS commands follow this form:

```
cvs [GLOBAL_OPTIONS] COMMAND [OPTIONS] [FILES]
```

The second set of options is sometimes called command options. Because there are so many of them, though, I'll just call them "options" in most

places to save space.

Many commands are meant to be run within a working copy and, therefore, may be invoked without file arguments. These commands default to all of the files in the current directory and below. So when I refer to the "file" or "files" in the text, Depending on how you invoked CVS, these files may or may not have been explicitly mentioned on the command line.

Global Options

Here are a few of the global options to CVS.

`-dãREPOSITORY`

This specifies the repository, which might be an absolute pathname or a more complex expression involving a connection method, username and host, and path. If it is an expression specifying a connection method, the general syntax is:

`:METHOD:USER@HOSTNAME:PATH_TO_REPOSITORY`

Here are examples using each of the connection methods:

`:ext:jblogs@cvs.sourceforge.ne:/cvsroot/emc` - Connects using rsh, ssh, or some other external connection program. If the `$CVS_RSH` environment variable is unset, this defaults to rsh; otherwise, it uses the value of that variable.

`:server:jblogs@cvs.sourceforge.ne:/cvsroot/emc` - Like `:ext:`, but uses CVS's internal implementation of rsh. (This may not be available on all platforms.)

`:pserver:jblogs@cvs.sourceforge.ne:/cvsroot/emc` - Connects using the password authenticating server (see The Password-Authenticating Server in Repository Administration; see also the `login` command.)

`:local:jblogs@localhost:/var/cvs/emc` - Accesses a local repository directly, as though only the absolute path to the repository had been given.

`-helpã[COMMAND]ãorã-Hã[COMMAND]`

These two options are synonymous. If no `COMMAND` is specified, a basic usage message is printed to the standard output. If `COMMAND` is specified, a usage message for that command is printed.

`-help-options`

Prints out a list of all global options to CVS, with brief explanations.

`-help-synonyms`

Prints out a list of CVS commands and their short forms ("up" for "update", and so on).

-n

Doesn't change any files in the working copy or in the repository. In other words, the command is executed as a "dry run" - CVS goes through most of the steps of the command but stops short of actually running it.

This is useful when you want to see what the command would have done had you actually run it. One common scenario is when you want to see what files in your working directory have been modified, but not do a full update (which would bring down changes from the repository). By running `cvsã-nãupdate`, you can see a summary of what's been done locally, without changing your working copy.

-q

This tells CVS to be moderately quiet by suppressing the printing of unimportant informational messages. What is considered "important" depends on the command. For example, in updates, the messages that CVS normally prints on entering each subdirectory of the working copy are suppressed, but the one-line status messages for modified or updated files are still printed.

-Q

This tells CVS to be very quiet, by suppressing all output except what is absolutely necessary to complete the command. Commands whose sole purpose is to produce some output (such as `diff` or `annotate`), of course, still give that output. However, commands that could have an effect independent of any messages that they may print (such as `update` or `commit`) print nothing.

-vãorã-version

Causes CVS to print out its version and copyright information and then exit with no error.

-zãGZIPLEVEL

Sets the compression level on communications with the server. The argument `GZIPLEVEL` must be a number from 1 to 9. Level 1 is minimal compression (very fast, but doesn't compress much); Level 9 is highest compression (uses a lot of CPU time, but sure does squeeze the data). Level 9 is only useful on very slow network connections. Most people find levels between 3 and 5 to be most beneficial.

A space between `-z` and its argument is optional.

Example Commands

Checkout

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc checkout emc
```

checkout is used to acquire a local working copy of the emc directory. Alternative names are co and get.

This can also be used with a -D option to get date specific versions of the code. For example:

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc -D"yesterday" emc
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc -D"2003-01-12"
emc
```

In the first example, all files that were modified prior to yesterday will be checked out, in the second, all changes before 12th Jan 2003.

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc -r test_5_1_03
emc
```

This checks out all the files that have been tagged (or in the branch) test_5_1_03.

Tags and dates are sticky and any further check outs will use the same tag unless

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc -A emc
```

is used.

Commit

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc commit -m"foo
bar edit"
```

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc ci -m"foo bar edit"
```

Commits changes from the local working directory to the central repository. This only works with checked out branches or the main trunk. If the sources have been checked out with a tag or date, then commits will fail (unless the tag is for a branch).

-m"message" is compulsory.

Add

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc add new_file
```

```
cvs -z3 -d:ext:jblogs@cvs.sourceforge.net:/cvsroot/emc new new_file
```

Adds a new file or directory to the repository. The new addition will not be visible until a cvs commit has also taken place - Note the comments about sticky tags also apply.

Chapter 2

EMC Source Code Introduction

This chapter was largely written by Fred Proctor and posted to the emc list. It describes the purposes of many of the source files that make up a typical EMC compile.

Most of the emc source code resides in the emc/src directories within a typical EMC installation. Source files are divided roughly according to the part of a running system that they build. Sub directories include

- drivers
- emcio
- emcnml
- emcplot3d
- emctask
- java
- rs274ngc
- rs274ngc_new
- simctrl
- stripchart

The purpose of most of these files is briefly described below.

2.1 src/drivers

There is some material regarding these in chapter 3 (need to include these)

2.2 src/emcnml

2.2.1 inivar

A program that prints to stdout the .ini file result of a variable-in-section search, useful for scripts that want to pick things out of INI files, e.g.,

```
you> plat/linux_2_2_14/bin/inivar -var CYCLE_TIME -sec TASK -ini youremc.ini
0.010
```

2.2.2 emcnmlsize

A short program that prints the size of the EMC NML status structures. Running this gets you something like this:

```
... sizeof(EMC_STAT) = 6552
```

which shows that the size of the top-level EMC status structure is 6552 bytes.

2.2.3 canon.hh

Declarations for the canonical interface functions called by the NC code interpreter. This includes functions like STRAIGHT_FEED(), which the interpreter calls when it sees a G1. The implementation of these functions is in emc/src/emctask/emccanon.cc. In that file, the functions are implemented to append NML messages onto the interp_list, which are dequeued in emc/src/emctaskmain.cc and send to either the motion or IO controller.

2.2.4 emc.cc

Definitions for the NML message classes declared in emc.hh. This file is automatically generated from emc.hh by the CodeGen Java program, and shouldn't be edited. The class methods in this file call the base class constructors, etc.

2.2.5 emc.gen

An ad-hoc script that tells the CodeGen Java code generator what to do when generating emc.cc from emc.hh.

2.2.6 emc.hh

The declarations for all NML command and status classes. This is the crux of the EMC messaging. NML messages are sent from the GUI to the EMC task level, from the task level to the motion and IO controllers, and are put onto the interp_list by the canonical interface in emc/src/emctask/emccanon.cc. All the EMC status is declared in the NML status structures.

2.2.7 emcargs.cc

A short file containing the definition of the `emcGetArgs()` function, which gets the common command line arguments like `-ini` and `-nml` and sets the EMC global variables `EMC_INIFILE`, `EMC_NMLFILE` which are declared in `emcglb.h` and defined in `emcglb.h`.

2.2.8 emccfg.h emcglb.c

Declarations and definitions for the default value of global variables, e.g., `DEFAULT_EMC_INIFILE`, `DEFAULT_EMC_NMLFILE`, `DEFAULT_AXIS_MAX_VELOCITY`.

2.2.9 emcglb.h

Declarations of the EMC global variables, defined in `emcglb.c`, e.g., `EMC_INIFILE`, `EMC_NMLFILE`, `AXIS_MAX_VELOCITY`.

2.2.10 emcnmlsize.cc

A short program that prints the size of the EMC NML status structures. Running this gets you something like this:

```
... sizeof(EMC_STAT) = 6552
```

which shows that the size of the top-level EMC status structure is 6552 bytes.

2.2.11 emcops.cc

Forced initializations of the EMC NML classes. This complements the auto-generated `emc.cc` and lets you force initializations to be what you want.

2.2.12 extintf.h

Declarations for all the "External Interface" functions that are called by the EMC motion and IO controllers, e.g., `extEncoderRead()`, `extDacWrite()`, `extHomeSwitchRead()`. Implementations of these functions dispatch the calls to the actual hardware, like the Servo To Go board or the parallel port. These dispatchers are files such as `emc/src/emcmot/{extstgmot.c,extsmmot.c}`, for the Servo To Go and parallel port stepper motion functions, respectively.

2.2.13 extppt.c

Dispatcher of the external interface functions for general-purpose IO declared in `emc/src/emcnml/extintf.h`, for the parallel port.

2.2.14 getinput.c getinput.h

Utility function for getting terminal input in a portable way. Used by the terminal-based diagnostics programs `emcpanel` and `usrmot`.

2.2.15 **iniaux.cc iniaux.hh iniaxis.cc iniaxis.hh inicool.cc inicool.hh inilube.cc inilube.hh inispin.cc inispin.hh initool.cc initool.hh initraj.cc initraj.hh**

Declarations and definitions for the functions called by the EMC task level to set parameters corresponding to .ini file parameters. These functions are called by `emc/src/emctask/{bridgeporttaskintf.cc,minimilltaskintf.cc}`, and do such things as compose and send NML messages to the motion and IO systems. The letters after the "ini" signify: aux – parameters from the [EMCIO] section related to estop axis – parameters from the [AXIS_#] section cool – parameters from the [EMCIO] section related to coolant lube – parameters from the [EMCIO] section related to lube spin – parameters from the [EMCIO] section related to the spindle tool – parameters from the [EMCIO] section related to tools traj – parameters from the [TRAJ] section

2.2.16 **inivar.c**

Definition of the 'inivar' program, which prints to stdout the .ini file result of a variable-in-section search, useful for scripts that want to pick things out of INI files.

`interp.cc interp.hh` Declarations and definitions for the NML_INTERP_LIST class, for which there is one instance, 'interp_list'. The `interp_list` is appended to by the NC interpreter via the canonical interface functions in `emc/src/emctask/emccanon.cc`, and dequeued by the task controller in `emc/src/emctask/emctaskmain.cc` using the methods such as `get()`, `clear()`, and `len()`.

`parport.c parport.h` Declarations and definitions for the parallel port IO functions, e.g., `pptDioByteRead()`, `pptDioShortWrite()`, `pptDioWordCheck()`.

2.3 **src/emcmot**

2.3.1 **usrmot**

A terminal-based diagnostics program that connects to the EMC motion controller, not the top-level EMC, and lets you type in motion commands and view status. It only requires that the motion controller be running.

2.3.2 **stgmod.o**

RT Linux motion controller based on the 4-axis Servo To Go Model I board.

2.3.3 **stg8mod.o**

RT Linux motion controller based on the 8-axis Servo To Go Model I board.

2.3.4 **stg2mod.o**

RT Linux motion controller based on the 4-axis Servo To Go Model II board.

2.3.5 stg_v2_8axis_mod.o

RT Linux motion controller based on the 8-axis Servo To Go Model II board.

2.3.6 steppermod.o

RT Linux motion controller for parallel port stepping, using the "binning" algorithm for allocating pulses that causes resonance in some systems. This links in the stepper algorithm from emcstepmot.c, compiled with the -DSTEPPER_MOTORS flag, which generates emcstepmot.o.

2.3.7 freqmod.o

RT Linux motion controller for parallel port stepping, using the "servo" algorithm for allocating pulses that improves the resonance problem from steppermod.o. This uses PID "servoing" to feed a pulse generator. You need to set at least the P gain to run this. This links in the stepper algorithm from emcmot.c, compiled with the -DSTEPPER_MOTORS flag, which generates emcfreqmot.o.

2.3.8 smdromod.o

A variation of freqmod.o, that uses feedback from Dan Mauch's DRO board to get real position measurements rather than simply accumulating output pulses.

2.3.9 simmod.o

A simulated system using physics-based modeling of DC servomotors. It's effectively a servo system with a loopback of DAC output through motor simulators and back through encoder simulators. This will exhibit overshoot, etc. if you don't have the PIDs tuned right. It uses the .ini file parameters

```
[AXIS_#] TORQUE_UNITS = OZ_IN ARMATURE_RESISTANCE = 1.10 AR-  
MATURE_INDUCTANCE = 0.0120 BACK_EMF_CONSTANT = 0.0254 RO-  
TOR_INERTIA = 0.0104 DAMPING_FRICTION_COEFFICIENT = 0.083 SHAFT_OFFSET  
= 0 REVS_PER_UNIT = 10 AMPLIFIER_GAIN = 1 MAX_OUTPUT_CURRENT  
= 10 LOAD_RESISTANCE = 1 COUNTS_PER_REV = 4096
```

This is good for running the EMC if you don't have a real machine tool or Servo To Go board, but do not have RT Linux up and running.

2.3.10 minitetra.o

A controller for the mini-tetra cable-based Stewart Platform (hexapod) machine. This replaces the trivial kinematics for machine tools (trivins.o) with the hexapod kinematics (genhexkins.o), and uses frequency stepping (emcmot.c, compiled with -DSTEPPER_MOTORS, which generates emcfreqmot.o).

2.3.11 cubic.c cubic.h

Declarations and definitions for the cubic interpolator, which interpolates trajectory points from the slower trajectory rate (.ini file parameter [TRAJ] CYCLE_TIME) to the faster axis rate (.ini file parameter [AXIS_#] CYCLE_TIME). Typically this is 10:1. Cubic interpolation matches the end points of each section, and the slopes as well, connecting them with a 3rd-order polynomial. Linear interpolation matches just the end points, connecting them with a line (a first-order polynomial). You could write a quintic interpolator, which would match the end points, slopes, and curvature with a fifth-order polynomial. This would eliminate any jerk impulses caused by cubic (and linear) interpolation.

2.3.12 dro.c dro.h

Functions for handling Dan Mauch's digital readout board, used for encoder feedback in the smdromod.o "stepper motor DRO module" real-time motion control module.

2.3.13 kinematics.h

Declares the kinematics functions kinematicsForward() and kinematicsInverse(), which are implemented in such files as trivkins.c, puma560kins.c, and genhexkins.c, for machine tools, the PUMA 560 robot, and hexapods, respectively. (This used to be a part of emcmot.h)

2.3.14 emcmot.c emcmot.h

Declarations and definitions for the EMC motion controller. emcmot.h contains the commands to emcmot, and the status data structure. These are put into the Linux/RT Linux shared memory buffer. This is not NML, since the shared memory buffer didn't support semaphores for mutual exclusion and we had to use an ad hoc method using head/tail counts to detect collisions.

emc/src/emctask/{bridgeporttaskintf.cc,minimilltaskintf.cc} convert the NML messages resulting from either the GUI (jogs, homes, etc.) or the interp_list (GO, G1, F60, etc.) into emcmot commands and write them to the shared memory buffer. emcmot.c is the crux of the EMC motion control. In this file is the sequence from start to finish for the motion cycle. There are two parts: trajectory planning and axis control. These run in a single executable, but since the trajectory planning is done at a slower rate ([TRAJ] CYCLE_TIME in the .ini file) than the axis control ([AXIS_#] CYCLE_TIME), the trajectory planning cycle is called every nth cycle of the motion controller, where n is the ratio of the traj/axis cycle times. The motion cycle starts by latching the input positions of the axes through a call to the external interface function extEncoderReadAll(). For servos this reads the encoders, going through the dispatcher function (e.g., extstgmot.c for the Servo To Go board). For steppers this function just returns the accumulated pulses that have gone out. Trajectory planning means accepting commands for motions onto the motion queue; planning the accel/cruise/decel for the motion; blending two successive motions together by starting the second when the first begins its final decel; handling feed

rate override, pause, and resume; calling the inverse kinematics to convert XYZ to axis positions (trivial on a machine tool, complicated on a robot); and appending them on the cubic interpolator so that they can be smoothly interpolated at the faster axis rate. Axis planning means taking the interpolated points off the cubic interpolators, one for each axis; and running PID servo compensation for servos, or computing the number of pulses or frequency needed for the binning (`emcstepmot.c`) or frequency generator (`emcmot.c`) stepper motor algorithms. Finally the output is written, and the motion task goes to sleep until the next timer interrupt wakes it up. Note that the motion task is not connected to any board interrupt. It uses the native RTOS timer tasking. This makes it more portable between boards. There are two distinct modes in `emcmot`: free mode and coordinated mode. In free mode, each of the axes operates independently of the others. This is the mode used for jogging and homing, and corresponds to the manual mode of the EMC task level. In coordinated mode, the axes are run together using the machine's kinematics. This corresponds to the auto or MDI modes of the EMC task level. `Emcmot` also handles aborting, enabling/disabling control (machine on/estop reset, respectively, in the task level), homing, latching probe triggers, and logging of various data types.

2.3.15 `emcmotcfg.h`

Declarations for compile time defaults for the EMC motion globals. These are distinct from the EMC globals, and are specifically targeted toward the motion system. They include such things as `EMCMOT_MAX_AXIS` (currently 6), `DEFAULT_TRAJ_CYCLE_TIME`. Many of these have analogous EMC globals, but some, such as `DEFAULT_SHMEM_BASE_ADDRESS`, are specific to the motion controller. When the compile-time flag `STEPPER_MOTORS` is defined, which you'll see in the Makefiles as `"-DSTEPPER_MOTORS"`, it includes the code for stepper motors.

2.3.16 `emcmotglb.c` `emcmotglb.h`

Declarations and definitions for the EMC motion globals. Their default values are declared in `emcmotcfg.h`. This includes such things as `TRAJ_CYCLE_TIME`, `SHMEM_BASE_ADDRESS`.

2.3.17 `emcmotlog.c` `emcmotlog.h`

Declarations and definitions for the EMC motion logging data structure, types of data that can be logged, and functions to manipulate the log (e.g., open, start, stop, close).

`emcmotutil.c` Functions for manipulating the `emcmot` error log. Declarations for these are in `emcmot.h`. These functions are separated from `emcmot.c` so that they can be shared with other programs that want to access the error log in shared memory.

2.3.18 `emcsegmot.c`

A probably obsolete alternative to the `emcmot.c` motion planning, based on work done by Rogier Blom of the U of Twente in the Netherlands while

he was at NIST as a guest researcher. This fixes the problem in `emcmot.c` where performance degrades as more, smaller segments are blended together. With `emcsegmot.c`, we were able to run the huge contouring NC program from MasterCAM for the race car. Rogier had a few more things to work on when he left, and it's not complete yet.

2.3.19 emcstepmot.c

A version of `emcmot.c` that uses the "binning" algorithm for allocating pulses that causes resonance in some systems. This was the original stepper motor algorithm added to the servo-only `emcmot.c`, using the `STEPPER_MOTORS` compile time flag. This creates a second task that runs faster than the motion controller, typically 10 cycles for every motion cycle. During these cycles, the pulses resulting from the motion cycle's calculated position increment are spread out as best as possible. The trouble comes when, say, 3 or 7 pulses need to be sent out in 10 cycles. There are always some leftover time slots, and the result is erratic pulsing that causes resonance. In many systems this is not a problem. On a stepper motor Bridgeport machine that Matt Shaver hooked up, the inertia caused resonance. This was the genesis for the frequency stepping algorithm, which is part of `emcmot.c` if you define `STEPPER_MOTORS`.

2.3.20 extsmdromot.c

Implementation of external interface motion functions for stepper motors, using Dan Mauch's DRO board for encoder feedback.

2.3.21 extsmmot.c

Implementation of external interface motion functions for stepper motors, using accumulated pulses as the measure of axis position.

2.3.22 extstgmot.c

Implementation of external interface motion functions for the Servo To Go board.

2.3.23 genhexkins.c genhexkins.h

Declarations and definitions for the general hexapod kinematics. These are the kinematics for Stewart Platform parallel kinematic machines. They're used in the `minitetra.o` motion controller, for the small cable-driven platform.

2.3.24 lsutil.c lsutil.h

Least-squares best fit code, not used for anything other than to build the `testlsutil` binary and do some least-squares fitting. This was useful for taking DAC outputs and linearizing them for the `.ini` file. You can do this with a good calculator, which we were too lazy to get so we wrote this.

2.3.25 **mmxavg.c mmxavg.h**

Min-max-average routines for building a window of points in a ring buffer, and picking out the minimum, maximum, and average value. A new point added to the ring buffer drops off the oldest point. These stats are accumulated for three spots in `emcmot.c`, for the case when no control is done (in disabled mode, same as estop reset), when axis control is done, and when axis control plus trajectory calculations are done. These aren't converted into the NML status structure and hence are not visible from any GUI. You can see them using the "usrmot" utility for motion diagnostics, by typing "show times" at the prompt. We should put at least the trajectory stats in the NML trajectory status structure, so you can do some sort of perfmeter-like display and see how much CPU time you're soaking.

2.3.26 **pid.c pid.h**

Declarations and definitions for PID servo control. In `pid.c`, the important function is `pidRunCycle()`. If we were to substitute another servo algorithm, we would have to change the explicit call of `pidRunCycle()` in `emcmot.c` to something like `runControlCycle()`, and have a dispatcher that implements this as a call to the new servo algorithm (and have one for PID too). The problem is that the `runControlCycle()` function needs to be generic enough to handle both cases. There are also the other functions, like `pidSetGains()`, that would have to be generalized into a `controlSetGains()` that has enough args for both PID and the other algorithms. There is also the problem of .ini file params. If anyone has another algorithm they want to run, please chime in.

2.3.27 **puma560kins.c puma560kins.h**

Declarations and definitions for the kinematics for the PUMA 560 robot. This implements `kinematicsForward()` and `kinematicsInverse()`, which are declared in `emcmot.h` and called in `emcmot.c`. This lets us switch kinematics in `emcmot` by linking in different ones, without having to recode `emcmot.c`.

2.3.28 **segmentqueue.c segmentqueue.h**

An alternative to the trajectory planning algorithms in `tp.c`, `tc.c` that fixes their problem with blending more than two motions, as is done in contouring programs with lots of short motion segments.

2.3.29 **stg.h stg.c stg2.c stg2.h stg_v2_axis8.c stgdiag.c**

Servo To Go board functions. These are taken from the DOS drivers that accompany the board, and are Linux-ized. * `stg.h` contains the declarations for Model I boards. * `stg.c` contains the functions for 4- and 8-axis axis Model I boards, selectable with the `-DSTG_8_AXES` compiler flag. * `stg2.h` contains the declarations for Model II boards. * `stg2.c` contains the functions for 4-axis Model II boards. * `stg_v2_axis8.c` contains the functions for 8-axis Model II boards. Blame Will for the hideous name.

2.3.30 `tc.c tc.h`

Motion planning algorithms used by `emcmot.c` for planning trapezoidal velocity profiles for accel, cruising, and decel. It handles linear and circular (with helical) motions. Together with `tp.c`, it forms the basis for `emcmot` trajectory planning.

2.3.31 `testlsutil.c`

A testing wrapper for the `lsutil.c,h` files for least-squares best fitting. Not used in `emcmot`, but only as a command-line utility.

2.3.32 `testtp.c tp.c tp.h`

Blending algorithms used in conjunction with `tc.c` for blending two adjacent motions. This could use some serious work due to the problem blending more than two motions. There's no lookahead for slowdown of tight corners, etc. Rogier Blom's segment queue algorithm fixed this, but that code is about 90% there and still needs some work, according to Rogier. Perhaps some enterprising programmer will take Rogier's paper and look into this.

2.3.33 `trivkins.c`

Trivial implementations of the `kinematicsForward()` and `kinematicsInverse()` functions, for machine tools in which X = axis 1, Y = axis 2, and Z = axis 3.

2.3.34 `usrmot.c`

Command-line diagnostics program that peeks into `emcmot`, and sends commands and prints status. An analogy of the full-blown EMC diagnostics program `emcpanel.c`, but for motion only.

2.3.35 `usrmotintf.c usrmotintf.h`

Functions to handle the interface between the EMC motion controller and its supervisor, the EMC task level. This handles the gory details of connecting between user-level Linux processes (like the EMC task level) and RT-Linux processes (like the EMC motion controller). In here you'll see all sorts of communication goodies, like the original RT-Linux FIFOs that aren't used anymore, mapping of physical memory set aside by LILO at boot time, etc.

2.4 `src/emcsim`

2.4.1 `amplifier.c amplifier.h`

Declarations and definitions for a simulated amplifier. Parameters that can be set include load resistance, max output current, etc. Running a cycle of the simulation with `amplifierRunCycle()` will generate the amplifier's

output, which can be used to drive a DC servomotor simulator. It checks for current limits and will trip if they're exceeded. Linked into `simmod.o` (the realtime simulator) and `emcmotsim` (the non-realtime simulator).

2.4.2 dcmotor.c dcmotor.h

Declarations and definitions for a simulated DC servomotor, using the model from Benjamin C. Kuo, "Automatic Control Systems," Fourth Edition, pp. 176-186. You set motor parameters such as armature resistance, armature inductance, back emf constant, motor rotor inertia, and viscous friction coefficient. Running a cycle of the simulation with `dcmotorRunCycle()` will generate the motor's next output position, given an input voltage. Linked into `simmod.o` (the realtime simulator) and `emcmotsim` (the non-realtime simulator).

2.4.3 encoder.c encoder.h

Declarations and definitions for a simulated encoder. Converts motor revolutions from the `dcmotor` simulator to encoder counts. Generates an index pulse on a revolution boundary. Linked into `simmod.o` (the realtime simulator) and `emcmotsim` (the non-realtime simulator).

2.4.4 extsimaio.c extsimdio.c extsimmot.c

Dispatcher code for the simulated external interfaces to analog, digital, and motion-related IO. Uses internal variables to save output state. For environments that support it, uses the presence of files to cause changes in input state. So, you can "touch di3" in the directory in which the simulator is running, and it registers digital input 3 as being on. "rm di3" removes the file, and the simulator will register digital input 3 as off.

2.4.5 inisim.cc

A program for initializing the parameters for `dcmotors`, `encoders`, `amplifiers`, etc. from `.ini` file params.

2.4.6 sim.h simaio.c simdio.c simmot.c

Declarations and definitions for the implementations of simulated external interface functions, e.g., `simEncoderRead()`, `simDacWrite()`. These are called by the dispatchers in `extsimaio.c`, `extsimdio.c`, and `extsimmot.c` to give them the `extEncoderRead()`, etc. function names expected in the generic motion and IO controllers.

2.4.7 simmot_n.cc simmot_n.gen simmot_n.h

NML-ization of simulator commands, for sending these to controllers via a `.ini` file. The idea here is to send messages to load the simulation parameters, rather than have the simulator read them from `.ini` files. This is for RT Linux, which doesn't have access to files.

2.5 src/emcio

2.5.1 emcio.hh

Declarations for the main C++-based EMC IO controller defined in `bridgeportiomain.cc` and `minimilliomain.cc`.

2.5.2 `bridgeportaux.cc` `bridgeportcool.cc` `bridgeportlube.cc` `bridgeportspin.cc` `bridgeporttool.cc`

Controllers for the estop (aux), coolant, lubricant, spindle, and tooling for Bridgeport-style machines. These contain the logic for handling spindle brake and actuation interlocks, time delays, etc. They can be thought of as a collection of state machines written in C++ that are equivalent to a PLC program. A big win would be to replace these with a Linux-based PLC that lets you program in the IEC-1131 suite of PLC languages. Supposedly Puffin PLC is an open-source effort to do this.

2.5.3 `minimillaux.cc` `minimilltool.cc`

Same as above, for the smaller set of controllers required to run the minimill, which doesn't have automatic coolant, lube, or spindle control.

`bridgeportiomain.cc` `minimilliomain.cc`

Main controller that reads the NML buffers for the EMC IO controller and runs through the Bridgeport-specific controllers in `bridgeport{aux,cool,lube,spin,tool}.cc`, or the minimill-specific controllers in `minimill{aux,tool}.cc`.

2.5.4 `extbridgeportio.c` `extminimillio.c`

Dispatchers for the IO functions called in the `bridgeportio*.cc` and `minimillio*.cc` controllers. For the Bridgeport, they call parallel port functions. For the minimill, most of them stub out to the simulated functions, since there's no coolant, lube, etc. on the minimill.

2.5.5 `iosh.cc`

A Tcl/Tk wrapper for general-purpose IO, the windowing "IO shell". It contains functions for reading and writing IO points that can be called from a Tcl/Tk script, e.g., `inb`, `outb`, and also to read and write NML commands and status. You use this to build a full-blown PLC in a Tcl/Tk script, giving it graphical LEDs, etc. An example of this is in `tkio.tcl`.

2.5.6 `tkio.tcl`

A Tcl/Tk script that uses the functions in the windowing IO shell `iosh.cc` to build a controller for Bridgeport-style IO control.

2.6 src/rs274ngc

2.6.1 canon_stand_alone.cc

Implementations of the canonical functions, prototyped in `emc/src/emcnml/canon.hh`, that just print their names instead of doing any work. Used by the `standalone` interpreter for testing purposes.

2.6.2 driver.cc

Functions necessary to build the standalone interpreter used for testing purposes.

2.6.3 rs274ngc.cc rs274ngc.hh

Declarations and definitions for the RS-274/NGC interpreter. This is a set of functions called by the EMC task level controller to open, read, and execute NC code in the dialect specified by Rockwell Automation in their report for the National Center for Manufacturing Sciences (NCMS), "The Next Generation Controller Part Programming Functional Specification (RS-274/NGC)." The Next Generation Controller was an Air Force-sponsored program to define an standard CNC.

2.6.4 rs274ngc.var

Template file for the EMC `.var` file, used to save and restore interpreter variables. Variables are named by number, and range from 1 to 5399. Some are reserved, e.g., axes offsets for G92 are 5211-5213, offsets for G54 are 5221-5223, and latched probe positions are 5061-5063. Those variables you list in the `.var` file are saved and restored each time the EMC is run.

2.7 src/emctask

2.7.1 emcpanel (built in emc/src/emctask)

A terminal-based diagnostics program that connects to the EMC as does a typical GUI, and lets you type in commands and view status. `emcpanel` provides all you need to bring up and run your machine, but your fingers will tire quickly. Usually the full EMC (`task`, `io`, `motion`) should be running, but `emcpanel` can be used with the IO controller by itself for debugging.

2.7.2 inimot (built in emc/src/emctask)

A program that reads a `.ini` file and sends the motion-specific values from the `[TRAJ]` and `[AXIS_#]` sections to the EMC motion controller. It is intended to be used to initialize the motion controller for subsequent debugging with `usrmot`, e.g.,

```
you> insmod plat/rtnlinux_2_2/lib/stgmod.o ... you> plat/linux_2_2_14/bin/inimot
-ini youremc.ini ... you> plat/linux_2_2_14/bin/usrmot -ini youremc.ini
...
```

2.7.3 **bridgeporttaskintf.cc minimilltaskintf.cc**

Implementations of the EMC NML functions that send motion and IO commands to the subsystems. These include such things as `emcAxisJog`, `emcSpindleOn`, etc. The functions compose and send NML or `emcmot` commands to the IO or motion subsystem. There are two of these, one for Bridgeport-style machines and one for the minimill, since there is some difference in how each handles IO. Specifically, spindle commands can be routed to the unused 4th axis on a Servo To Go board to be used for analog spindle control on the minimill. In this case, the spindle commands that the `bridgeporttaskintf.cc` routes to the EMC IO controller are instead routed to the EMC motion controller as raw DAC commands by `minimilltaskintf.cc`.

2.7.4 **debug.cc**

As Will modestly describes it, "a pathetic copy of the DOS debug program. OK for testing io ports."

2.7.5 **tkemc.tcl**

AKA TkEMC, the main EMC GUI, which replaces the X windows and earlier interfaces. It's written in Tcl/Tk, a scripting language. The default windowing shell "wish" used for regular Tcl/Tk scripts won't work for EMC since it lacks commands for sending and reading NML commands and status. We wrote `emcsh.cc`, the windowing "EMC shell" which extends wish with EMC commands. The same sort of thing can be done for Perl/Tk and Python/Tk. Someone should look into this.

2.7.6 **genedit.tcl**

A general-purpose editor written in Tcl/Tk. This is used to edit NC programs from the TkEMC GUI, rather than having to open another editor window.

2.7.7 **tkbackplot.tcl**

A nifty tool path drawing application that reads the machine position from the EMC and draws the tool path. It has buttons that duplicate those on the `tkemc.tcl` GUI script so you can bring programs up and run them in the same application.

2.7.8 **emccalib.tcl emclog.tcl emctestng.tcl emctuning.tcl**

Scripts that can be called from the TkEMC Tcl/Tk-based GUI for calibrating PID settings, logging, testing and system identification. There are likely to be more of these now that TkEMC automatically builds a menu of all `.tcl` scripts it finds in the `tcl/scripts` directory.

2.7.9 emccanon.cc

Implementation of the canonical functions, called by the NC interpreter in `emc/src/rs274ngc/rs275ngc.cc`, that put messages on the `interp_list` queue for the EMC task controller in `emctaskmain.cc` to pull off and handle. Everything that happens in the interpreter has to go through here.

2.7.10 emcdummy.cc

A dummy version of the EMC, that creates the NML communication buffers, reads commands, and says it's done with the commands right away regardless of what they are. It's useful as an EMC stub for testing handshaking of external applications that send EMC commands. It's not useful for anything more than testing handshaking, since it doesn't fake much of the EMC status. You can't really use it for GUI development. Run the EMC simulator "emcmotsim" in conjunction with the EMC task and IO controllers for this sort of thing.

2.7.11 emcpanel.cc

A text-based interface to the EMC. You can type in commands and read status from the command line. One could build a GUI from this using a program like "expect" and Tcl/Tk, but it's slower than building EMC commands into a the "wish" windowing shell, which is what we did with `emcsh.cc`.

2.7.12 emcsh.cc

An extension of the Tcl/Tk windowing shell "wish" that adds EMC commands, hence the name "emcsh" for "EMC shell." `emcsh` connects to the EMC using NML, and provides all the Tcl string commands you need to send EMC commands and check status. It's meant to be the first line in Tcl/Tk scripts, namely the `tkemc.tcl` GUI script, so you don't invoke it manually. You can, however, and use it to type commands and view status similar to `emcpanel`.

2.7.13 emcsvr.cc

A network server for the EMC command and status buffers. This needs to be running for processes to connect remotely across the network. If you uncomment

```
[EMCSVR] EMCSERVER = emcsvr
```

in the `.ini` file, it will be run. Then, in the `.nml` file, you can specify that a process like `xemc` (which is what the GUIs call themselves) should connect remotely, e.g.,

```
P xemc emcCommand REMOTE localhost W 0 10.0 0 10 P xemc emcStatus
REMOTE localhost R 0 10.0 0 10 P xemc emcError REMOTE localhost R
0 10.0 0 10 P xemc toolCmd REMOTE localhost W 0 10.0 0 10 P xemc
toolSts REMOTE localhost R 0 10.0 0 10
```

where "localhost" is replaced with the hostname of the computer that the EMC is actually running on.

2.7.14 **emctask.cc**

Implementation of the EMC task level functions, e.g., `emcTaskSetMode()` for manual, auto, MDI; `emcTaskSetState()` for off, on, estop, estop reset; and `emcTaskPlanOpen()`, `emcTaskPlanRun()` for running NC files.

2.7.15 **emctaskmain.cc**

The main control loop for the EMC task level. It reads the NML buffers; handles commands from the GUI or other external source; calls the interpreter functions if a program is running; looks at the `interp_list` and checks to see what's next to send out and when it should go out; and writes the EMC status.

2.7.16 **inimot.cc**

A short program that reads the `.ini` file and sends the motion-specific values to the EMC motion controller. If you run the motion controller manually, without the task and IO controllers, you can use this to force `.ini` parameters into the motion system so that it's configured properly for subsequent testing with the `usrmot` utility.

2.7.17 **keystick.cc**

An ancient character-graphics "GUI". Uses the curses terminal control library to draw characters at various X-Y coordinates on the screen. This was done in advance of a full X Windows GUI, which was then made obsolete by the TkEMC.

2.7.18 **xemc.cc yemc.cc**

X Windows-based GUIs, written using the Xt toolkit. It's another example of how one can, but should not, write a GUI. `xemc` is the original; `yemc` is a successor that gives 6-axis status for use with the minitetra cable-based Stewart Platform. Use TkEMC instead. Others may be looking into a Motif/Lesstif version of this, and can use this source code as an example of how to connect to the EMC.

Chapter 3

Hardware Drivers

3.1 Wrapper Functions

The structure of the EMC sources are such that at present, two layers of code are expected between emcmot and the actual hardware. Listed below are the calls made by emcmot - These will then call the board specific functions in a separate source file unless the hardware does not support it. The instinctive route taken by most would be to encode the necessary functions directly. Having the abstraction layer gives the end user the option to mix and match hardware and only have to write one wrapper, or extend the functionality. The prospect of being able to mix stepper motors and servos without resorting to additional hardware is just one attraction of EMC.

For example, the initialization of the motion control card by emcmot would consist of calls to the following routines in extstgmot.c :-

```
#include <extint.h>
#include <stg.h>
int extMotInit(const char * stuff)
{
    return stgMotInit(stuff);
}
int extDioInit(const char * stuff)
{
    return stgDioInit(stuff);
}
int extAioInit(const char * stuff)
{
    return stgAioInit(stuff);
}
```

The real work of initialising the card would then be coded in stg.c :-

```
int stgMotInit(const char * stuff)
{
    int t;
    Initialize(5);
}
```

```

EncoderInit();
/* output 0's to amps */
for (t = 0; t < STG_MAX_AXIS; t++)
{
    /* write 0 to DACs */
    stgDacWrite(t, 0.0);
}
return 0;
}

int stgDioInit(const char * stuff)
{
    /* set digital IO bits for A,B input, C output */
    DIODirection(0x0C);

    return 0;
}

int stgAioInit(const char * stuff)
{
    /* nothing need be done */
    return 0;
}

```

The `stgAioInit` routine returns as if everything was configured correctly, but as the analogue IO is not used, only a dummy function is needed. Should we wish to add another card that supports additional analogue IO, a set of routines can be written in a separate source file and the calls included in `extstgmot`. Better still would be to create a new `extstgmot` (`extstgPlusmot.c` ?) with the additional calls included. Should we now want to use an alternative Digital IO card along side the Servo-To-Go and the new analogue cards, it is a simple matter to re-route the calls in `extstgPlusmot.c` :-

```

int extMotInit(const char * stuff)

{
    return stgMotInit(stuff);
}
int extDioInit(const char * stuff)
{
    /* Route Digital IO through a PCI card */
    return NewDioInit(stuff);
}
int extAioInit(const char * stuff)
{
    /* And use some new Analogue hardware */
    return PlusAioInit(stuff);
}

```

All this will be pulled together during the linking stage as described in the Makefile - This is a subject for another section.

Although this method of coding appears to be inefficient and could be reduced to just one `extstgmot.c`, it would require substantially more work

to maintain. Mixing and matching hardware would also become a major hacking exercise.

3.2 Hardware Functions

Functions required by emcmot running in realtime.

extMotInit(const char * stuff)

Args: optional.

Returns 0 for success, or -1 if initialization failed.

Call once before any of the other motion IO functions are called. 'stuff' argument can be used to pass a pointer to board-specific stuff like config files to the initializing routine.

extDioInit(const char * stuff)

Args: optional.

Returns 0 for success, or -1 if initialization failed.

Call once before any of the other digital IO functions are called, but after extMotInit. 'stuff' argument can be used to pass a pointer to board-specific stuff like config files to the initializing routine.

extAioInit(const char * stuff)

Args: optional.

Returns 0 for success, or -1 if initialization failed.

Call once before any of the other analog IO functions are called, but after extMotInit and extDioInit. 'stuff' argument can be used to pass a pointer to board-specific stuff like config files to the initializing routine.

extMotQuit()

Args: none

Returns void.

Calls the board quit routine once, after which point no other functions will be called until after a call to extMotInit().

extDioQuit()

Args: none

Returns void.

Call once before extMotQuit, after which point no other functions will be called until after a call to extDioInit().

extAioQuit()

Args: none

Returns void.

Call once before `extMotQuit` (and `extDioQuit`), after which point no other functions will be called until after a call to `extAioInit()`.

extMotDout

Args:

Returns

Internal function in `emcmot.c` that perhaps should be in the hardware wrapper.

extAmpEnable(int axis, int enable)

Args: Axis number, Enable flag

Returns 0 if OK, -1 if not valid (axis is out of range)

enables or disables amplifier for indicated axis; enable flag is 1 to enable, 0 to disable

extAmpFault(int axis, int * fault)

Args: Axis number, Pointer to fault flag

Returns 0 if OK, -1 if not valid (axis out of range)

Copies into `fault[axis]` the fault state of the amplifier. 1 is faulted, 0 is OK.

extMaxLimitSwitchRead(int axis, int * flag)

extMinLimitSwitchRead(int axis, int * flag)

Args: Axis number, Pointer to limit switch flag

Returns 0 if OK, -1 if not valid (axis is out of range). */

sets `*flag` to 0 if the limit switch is not tripped, i.e., everything is fine, 1 if the limit switch is tripped, i.e., the axis went too far in the associated direction.

Maximum is defined as the direction in which encoder values increase, minimum is the other direction.

extHomeSwitchRead(int axis, int * flag)

Args: Axis number, Pointer to home switch flag

Returns 0 if OK, -1 if not valid (axis is out of range).

sets `*flag` to 0 if the home switch is not tripped, i.e., everything is fine, 1 if the home switch is tripped.

extEncoderReadAll(int max, double * counts)

Args: Max axis, Pointer to Encoder array.

Returns 0 if OK or -1

Stores the range of encoders' counts in counts[0...Max axis-1] array. Returns 0 if OK or -1 if the max is greater than number of encoders. max is number of encoders, so for encoders 0..3 max would be 4.

extEncoderSetIndexModel()

Args: none

Returns 1 or 2

Returns the type of encoder indexing done by the board.

NOTE: This function is not called by emcmot.c, it is only listed here as reference.

The handling of index pulses for encoder homing is problematic. There are two models (at least) for how incremental encoders can be homed off the index pulse. The first assumes the encoder index pulse sets a flag when it occurs, but the count value doesn't change. The controller polls on this flag, and when it is seen to be latched the controller reads the position and uses this as an offset. There is a latency that makes this less accurate than the second method, in which the occurrence of the index pulse resets the count automatically. The problem is that the controller logic is different for the two methods, so if you replace a board which does it the first way with one that does it the second, you need to recode the logic. The function "extEncoderIndexModel()" returns the model used by the board, so at least you can detect it and if you have coded up both types you can switch automatically.

EXT_ENCODER_INDEX_MODEL_MANUAL indicates that the index pulse sets a latch flag, but you have to read this and then the encoder value and handle offsets yourself. The board won't change its count value on the index pulse.

EXT_ENCODER_INDEX_MODEL_AUTO indicates that the index pulse zeros the encoder count automatically. */

/* flags defined bit-exclusive for OR'ing if board can do multiple ways */

#define EXT_ENCODER_INDEX_MODEL_MANUAL 0x00000001

#define EXT_ENCODER_INDEX_MODEL_AUTO 0x00000002

extEncoderSetIndexModel(unsigned int model)

Args: Int=1

Returns 0 if OK or -1

NOTE: emcmot.c only calls this function with EXT_ENCODER_INDEX_MODEL_MANUAL during the init_module() process.

For boards that support multiple index models, select which one is to be used. Returns 0 if OK, -1 if model can't be supported. */

extEncoderReadLatch(int encoder, int * flag)

Args: Axis number, pointer to latch array

Returns 0 if OK or -1

For EXT_ENCODER_INDEX_MODEL_MANUAL, stores 1 if index has latched the flag in flag[Axis], 0 if not. For EXT_ENCODER_INDEX_MODEL_AUTO,

stores 1 if the encoder has been zeroed. Returns 0 if OK, -1 if not valid.

extEncoderResetIndex(int encoder)

Args: Axis number

Returns 0 if OK or -1

Resets index latching for the indicated axis. Returns 0 if OK or -1 if the index is out of range. This applies to both EXT_ENCODER_INDEX_MODEL_MANUAL and EXT_ENCODER_INDEX_MODEL_AUTO. For the first, it resets the latch flag. For the second, it enables zeroing on the next index. encoder is range 0..max encoder - 1.

extDioRead(int index, int *value)

Args: Bit index number, Pointer to array

Returns 0 if OK or -1

Reads value of digital input at index, stores in value[index].

Note. This is a bit read, and index starts at 0.

extDacWriteAll(int max, double * volts)

Args: Max DAC, pointer to volts array

Returns 0 if OK or -1

When emcmot commands a move, the raw DAC value is placed into an array of double floats. A call to extDacWriteAll is made passing the maximum number of DACs expected in the system along with a pointer to the raw DAC values. These two arguments are then passed to a routine to convert the raw values in to a suitable format before writing to the DACs. Any level shifting and output limiting should be done here before returning 0 for success or -1 for failure.

Additional notes

A look at the hardware specific sources will reveal a number of other functions not listed above. These would typically be internal routines or functions used by a standalone diagnostic program. The Servo-To-Go driver being ported from a DOS driver, is a prime example of this.

For realtime code, do NOT use any external libs such as stdout, nor should you attempt to use C++. Both of these will result in “undefined symbol” errors when an attempt is made to load the module.

Although the kernel functions can be used instead, a number of them do not work under realtime. One is iopl(), another is the print function printk. iopl() is not implimented in either rtlinux or rtai. printk is replaced with rtl_printk (for rtlinux) and rt_printk (for rtai), the equivalent stdout function would be printf - Although printing of floats does not work in realtime.

Linux Device Drivers by Alessandro Rubini & Jonathan Corbet is recommended reading, as is the kernel documentation (try 'make psdocs pdfdocs htmdocs' in the linux source directory).

Chapter 4

Makefiles

The Makefile is the key to compiling EMC - A study of each one is of great help in understanding which pieces of the source code are realtime or non-realtime. Lets look at a makefile for a proposed standalone module to impliment a jog handwheel - We need not concern ourselves with the actual code at present.

```
# Makefile for compiling EMC Drivers
# We always have to optimize because of some weirdness with
io.h
OPTIMIZE=1
# set the name of the module directory (this directory's name)
MODULE = drivers
# Target code
ifdef BUILD_REALTIMEONLY
SRCS =
HEADERS =
BINS =
OBS =
else
SRCS = handwheel.cc
HEADERS =
BINS = handwheel
OBS = handwheel.o
endif

# get all the platform and application definitions
include ../Makefile.inc

$(DEVP_BIN_DIR)/handwheel: \
    $(DEVP_LIB_DIR)/handwheel.o \
    $(COMPILER_SETUP); \
    $(CPLUSPLUS) \
    $^ \
    $(RCS_LINK_FLAG) \
    $(CPLUSPLUSLINK) -o $@

handwheel: $(DEVP_BIN_DIR)/handwheel

.PHONY: handwheel
```

Starting from the top, we have the `OPTIMIZE` flag set to 1. This is passed to the compiler with `-O` in order to determine how compact the final binary will be.

The `MODULE` name is nothing more than a handy reference to which directory the original sources are to be found.

Next are two sets of `SRCS`, `HEADERS`, `BINS`, and `OBJS` surrounded by a conditional statement. If the final objects are to be compiled for realtime, the headers and sources are copied to the relevant directories in `emc/plat/{$REALTIME}`. In this example, the module is entirely non-realtime so the files will be copied to `emc/plat/{$NONREALTIME}/include` and `emc/plat/{$NONREALTIME}/src` before compiling.

In order that the multitude of compiler and linker flags are correctly defined for the platform in use, `Makefile.inc` in the parent directory is included - This in turn will include `rcslib/etc/generic.def` which in turn includes the platform specific `rcslib/etc/{$PLAT}.def`. Once this convoluted route to the various compiler and linker flags has been reached, the module can finally be compiled. The `make` utility will automatically determine from the source extension if a C++ or a plain C compiler should be used.

Chapter 5

Customsing the software

Contents

- Re-compiling EMC along with various flags (yet to do)
- Feed Per Rev Example
- Adding NML codes

5.1 Re-compiling EMC along with various flags

(add this)

5.2 Feed Per Rev Example

From: Fred Proctor <frederick.proctor@nist.gov>

To: Multiple recipients of list <emc@nist.gov>

Subject: Re: emc development

Chris,

You wrote:

> I have been using EMC since RTL 1.1 and now have it running under
> RTL2.0 > and would like to contribute by adding a feed/rev gcode. I have
> C > programming experience and I am a control engineer by profession.
> Could > you please advise me of the correct procedure to follow so that my
> efforts can be easily integrated.

Wow, great, we could use some more programmers. The correct procedure is to get registered on SourceForge as an EMC developer, as described in the email to emc@nist.gov by Will Shackelford here at NIST that I appended to this email. Once you do this, you can start whacking away. You should also subscribe to emc@nist.gov so you can ping everyone with questions.

The hard part (and the SourceForge setup is not that easy) is figuring out how to change the EMC code to do what you want. (See chapter 1 here)

Specifically for your case, adding a feed per rev G code, here is my guide:

1. Tom Kramer (thomas.kramer@nist.gov) wrote the G code interpreter. It's in emc/src/rs274ngc/. For questions on the interpreter itself, he's

the one. Our philosophy was to implement a dialect of NC code based on a specification written by Rockwell Automation for the National Center for Manufacturing Sciences (NCMS). It's supposedly based on a Fanuc dialect.

2. Once the interpreter parses the G codes, it calls one of many "canonical functions" defined in `emc/src/emctask/emccanon.cc`. That is, when it sees a comment, it calls the `COMMENT()` function. When it sees a `G01`, it calls `STRAIGHT_FEED()`. In `emccanon.cc`, you'll see that these functions end up putting data structures on a global linked list, the `interp_list`. The data structures comprise the Neutral Messaging Language (NML), and is the format in which all commands and status flow around. The message set can be seen in all its gory detail in `emc/src/emcnml/emc.hh`.

3. The EMC task controller (`emc/src/emctask/emctaskmain.cc`) calls the interpreter with `emcTaskPlanRead()/emcTaskPlanExecute()` calls, then looks at the `interp_list` to see what was put on it. It then decides when the next message should be handled, and who gets it. This is the `checkPreconditions()`, `issueCommand()`, and `checkPostconditions()` sequence in `emctaskmain.cc`.

4. Either the motion controller or IO controller is sent the message, and does the actual work. In your case, feed per rev may be able to be converted into an equivalent straight feed at a rate calculated by looking at the S code. This wouldn't impact much at all, maybe just `rs274ngc.cc`. Talk to Tom Kramer for his advice.

Best regards,

-Fred

5.3 Customising NML

NML itself can be extended. This is done when new capabilities are added to the controller, so extending NML really means extending the EMC. The last one I added was the ability to override limits temporarily, to support homing off a limit switch. This involved coding it up in the motion controller (`emc/src/emcmot/emcmot.c`), adding a new message and status item for this (`emc/src/emcmot/emcmot.h`), adding it to NML (`emc/src/emcnml/emc.hh`), handling this in the EMC task controller (`emc/src/emctask/emctaskmain.cc`), adding it to the EMC windowing shell (`emc/src/emctask/emcsh.cc`), then updating the `tkemc` script to use it. Whew.

-Fred Proctor

Chapter 6

EMCSH

6.1 Adding Commands

The C++ program `emc/src/emctask/emcsh.cc` is the back-end of `tkemc`. This is a replacement for "wish", the default Tcl/Tk "windowing shell" that is used to run Tcl/Tk scripts. `emcsh` does everything that `wish` does, plus connects to the EMC's communication buffers, and provides new functions like `emc_mode`, `emc_estop` to send commands, and `emc_abs_act_pos`, `emc_rel_cmd_pos` to get status.

You can add new commands to `emcsh.cc`, following an example like `emc_estop` or `emc_rel_cmd_pos` for sending a command or retrieving status, respectively. The strings and their corresponding C++ functions have the same names, so you can search through the file and see what's going on. The complete list of available commands and status items provided by the EMC is in `emc/src/emcnml/emc.hh`. Many of these aren't yet in `emcsh.cc`, and therefore not accessible via `tkemc`. I only put in the ones that I thought would be needed in the GUI at first.

6.2 Working with emcsh

Adding new Tcl commands to `emcsh` and `iosh` is relatively straight forward if you are familiar with C and C++. All the Tcl/Tk functions can be called at a low level in C. For descriptions of each call, take a look at the man pages, copies of which can be found at <http://tcl.activestate.com/man/tcl8.0/> and <http://www.elf.org>.

6.3 Tcl command extensions for EMC

The following information has been extracted from `emcsh.cc`. For examples of the correct usage, look at the tcl scripts in `emc/src/task/tkemc.tcl` or `emc/plat/nonrealtime/bin/tkemc`.

To use the `emcsh` commands, use the following at the start of the script:

```
#!/bin/sh # the next line restarts using emcsh \ exec plat/nonrealtime/bin/emcsh "$@" "$@"
```

Using `emcsh`:

`emcsh {<filename>} {- -ini <ini file>}`

With `filename`, it opens NML buffers to the EMC, runs the script, closes the buffers, and quits.

With `- -ini <infile>`, uses `infile` instead of `emc.ini`. Note that the two dashes prevents Tcl from looking at the remaining args, which would otherwise trigger a Tcl error that it doesn't understand what `-ini` means.

Without `filename`, it runs interactively.

The files (or manual input) are Tcl scripts, extended with EMC-specific commands. These commands are all prefixed with `"emc_"`, which makes them somewhat inconvenient for typing but avoids name conflicts, e.g., `open`.

Some commands take 0 or more arguments. 0 arguments means they return the associated value; the argument would be to set the value.

Commands are sent to the EMC, and control resumes immediately. You can call a timed wait until the command got there, or a timed wait until the command completed, or not wait at all.

6.4 EMC commands:

- `EMC_INIFILE` Exported values of the EMC global of the same name
- `emc_plat` Returns the platform for which this was compiled, e.g., `linux_2_0_36`
- `emc_ini <var> <section>` Returns the string value of `<var>` in section `<section>`, in `EMC_INIFILE`
- `emc_debug {<new value>}` With no arg, returns the integer value of `EMC_DEBUG`, in the EMC. Note that it may not be true that the local `EMC_DEBUG` variable here (in `emcsh` and the GUIs that use it) is the same as the `EMC_DEBUG` value in the EMC. This can happen if the EMC is started from one `.ini` file, and the GUI is started with another that has a different value for `DEBUG`.
- With an arg, sends a command to the EMC to set the new debug level, and sets the `EMC_DEBUG` global here to the same value. This will make the two values the same, since they really ought to be the same.
- `emc_set_wait none | received | done` Set the wait for commands to return to be right away (`none`), after the command was sent and received (`received`), or after the command was done (`done`).
- `emc_wait received | done` Force a wait for the previous command to be received, or done. This lets you wait in the event that `"emc_set_wait none"` is in effect.
- `emc_set_timeout <timeout>` Set the timeout for commands to return to `<timeout>`, in seconds. Timeout is a real number. If it's `<= 0.0`, it means wait forever. Default is `0.0`, wait forever.
- `emc_update (none) | none | auto` With no arg, forces an update of the EMC status. With `"none"`, doesn't cause an automatic update of status with other `emc_` words. With `"auto"`, makes `emc_` words automatically update status before they return values.

- `emc_error` Returns the current EMC error string, or "ok" if no error.
- `emc_operator_display` Returns the current EMC operator display string, or "ok" if none.
- `emc_operator_text` Returns the current EMC operator text string, or "ok" if none.
- `emc_time` Returns the time, in seconds, from the start of the epoch. This starting time depends on the platform.
- `emc_estop (none) | on | off` With no arg, returns the estop setting as "on" or "off". Otherwise, sends an estop on or off command.
- `emc_estop_in` Returns the estop input setting as "on" or "off".
- `emc_machine (none) | on | off` With no arg, returns the machine setting as "on" or "off". Otherwise, sends a machine on or off command.
- `emc_mode (none) | manual | auto | mdi` With no arg, returns the mode setting as "manual", "auto", or "mdi". Otherwise, sends a mode manual, auto, or mdi command.
- `emc_mist (none) | on | off` With no arg, returns the mist setting as "on" or "off". Otherwise, sends a mist on or off command.
- `emc_flood (none) | on | off` With no arg, returns the flood setting as "on" or "off". Otherwise, sends a flood on or off command.
- `emc_lube (none) | on | off` With no arg, returns the lubricant pump setting as "on" or "off". Otherwise, sends a lube on or off command.
- `emc_lube_level` Returns the lubricant level sensor reading as "ok" or "low".
- `emc_spindle (none) | forward | reverse | increase | decrease | constant | off` With no arg, returns the value of the spindle state as "forward", "reverse", "increase", "decrease", or "off". With arg, sends the spindle command. Note that "increase" and "decrease" will cause a speed change in the corresponding direction until a "constant" command is sent.
- `emc_brake (none) | on | off` With no arg, returns the brake setting. Otherwise sets the brake.
- `emc_tool` Returns the id of the currently loaded tool
- `emc_tool_offset` Returns the currently applied tool length offset
- `emc_load_tool_table <file>` Loads the tool table specified by <file>
- `emc_home 0 | 1 | 2 | ...` Homes the indicated axis.
- `emc_jog_stop 0 | 1 | 2 | ...` Stop the axis jog
- `emc_jog 0 | 1 | 2 | ... <speed>` Jog the indicated axis at <speed>; sign of speed is direction
- `emc_jog_incr 0 | 1 | 2 | ... <speed> <incr>` Jog the indicated axis by increment <incr> at the <speed>; sign of speed is direction

- `emc_feed_override` {<percent>} With no args, returns the current feed override, as a percent. With argument, set the feed override to be the percent value
- `emc_abs_cmd_pos` 0 | 1 | ... Returns double obj containing the XYZ-SXYZ commanded pos in abs coords, at given index, 0 = X, etc.
- `emc_abs_act_pos` Returns double objs containing the XYZ-SXYZ actual pos in abs coords
- `emc_rel_cmd_pos` 0 | 1 | ... Returns double obj containing the XYZ-SXYZ commanded pos in rel coords, at given index, 0 = X, etc., including tool length offset
- `emc_rel_act_pos` Returns double objs containing the XYZ-SXYZ actual pos in rel coords, including tool length offset
- `emc_joint_pos` Returns double objs containing the actual pos in absolute coords of individual joint/slider positions, excludes tool length offset
- `emc_pos_offset` X | Y | Z | R | P | W Returns the position offset associated with the world coordinate provided
- `emc_joint_limit` 0 | 1 | ... Returns "ok", "minsoft", "minhard", "maxsoft", "maxhard"
- `emc_joint_fault` 0 | 1 | ... Returns "ok" or "fault"
- `emc_joint_homed` 0 | 1 | ... Returns "homed", "not"
- `emc_mdi` <string> Sends the <string> as an MDI command
- `emc_task_plan_init` Initializes the program interpreter
- `emc_open` <filename> Opens the named file
- `emc_run` {<start line>} Without start line, runs the opened program from the beginning. With start line, runs from that line. A start line of -1 runs in verify mode.
- `emc_pause` Pause program execution
- `emc_resume` Resume program execution
- `emc_step` Step the program one line
- `emc_program` Returns the name of the currently opened program, or "none"
- `emc_program_line` Returns the currently executing line of the program
- `emc_program_status` Returns "idle", "running", or "paused"
- `emc_program_codes` Returns the string for the currently active program codes
- `emc_joint_type` <joint> Returns "linear", "angular", or "custom" for the type of the specified joint

- `emc_joint_units <joint>` Returns "inch", "mm", "cm", or "deg", "rad", "grad", or "custom", for the corresponding native units of the specified axis. The type of the axis (linear or angular) is used to resolve which type of units are returned. The units are obtained heuristically, based on the `EMC_AXIS_STAT::units` numerical value of user units per mm or deg. For linear joints, something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom". For angular joints, something close to 1.000 is deemed "deg", $\text{PI}/180$ is "rad", $100/90$ is "grad", otherwise it's "custom".
- `emc_program_units emc_program_linear_units` Returns "inch", "mm", "cm", or "none", for the corresponding linear units that are active in the program interpreter.
- `emc_program_angular_units` Returns "deg", "rad", "grad", or "none" for the corresponding angular units that are active in the program interpreter.
- `emc_user_linear_units` Returns "inch", "mm", "cm", or "custom", for the corresponding native user linear units of the EMC trajectory level. This is obtained heuristically, based on the `EMC_TRAJ_STAT::linearUnits` numerical value of user units per mm. Something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom".
- `emc_user_angular_units` Returns "deg", "rad", "grad", or "custom" for the corresponding native user angular units of the EMC trajectory level. Like with linear units, this is obtained heuristically.
- `emc_display_linear_units emc_display_angular_units` Returns "inch", "mm", "cm", or "deg", "rad", "grad", or "custom", for the linear or angular units that are active in the display. This is effectively the value of `linearUnitConversion` or `angularUnitConversion`, resp.
- `emc_linear_unit_conversion {inch | mm | cm | auto}` With no args, returns the unit conversion active. With arg, sets the units to be displayed. If it's "auto", the units to be displayed match the program units.
- `emc_angular_unit_conversion {deg | rad | grad | auto}` With no args, returns the unit conversion active. With arg, sets the units to be displayed. If it's "auto", the units to be displayed match the program units.
- `emc_log_open <file> <type> <size> <skip>{<axis>}` Open a log into file named <file>. <type> is one of `axis_pos`, `axes_inpos`, `axes_outpos`, `axis_vel`, `axes_ferror`, `traj_pos`, `traj_vel`, `traj_acc`, `pos_voltage`. <size> is size in entries. <skip> is how many to skip. <axis> pertains only to `axis_pos` and `axis_vel` types.
- `emc_log_start emc_log_stop` Starts or stops logging into the log, which must be already opened with `emc_log_open`.
- `emc_log_close` Close the log and dump contents to the file specified in `emc_log_open`.
- `emc_probe_index` Which wire is the probe on or which bit of digital IO to use? (No args gets it, one arg sets it.)

- `emc_probe_polarity` Value to look for for probe tripped? (0 args gets it, one arg sets it.)
- `emc_probe_clear` Clear the probe tripped flag.
- `emc_probe_tripped` Has the probe been tripped since the last clear.
- `emc_probe_value` Value of current probe signal. (read-only)
- `emc_probe` Move toward a certain location. If the probe is tripped on the way stop motion, record the position and raise the probe tripped flag.
- `emc_teleop_enable` Should motion run in teleop mode? (No args gets it, one arg sets it.)
- `emc_kinematics_type` returns the type of kinematics functions used `identity=1, serial=2, parallel=3, custom=4`

6.5 Additional Comments

Further on in `emcsh.cc` is some additional comments regarding units.

Unit conversion

Length and angle units in the EMC status buffer are in user units, as defined in the INI file in `[TRAJ] LINEAR,ANGULAR_UNITS`. These may differ from the program units, and when they are the display is confusing.

It may be desirable to synchronize the display units with the program units automatically, and also to break this sync and allow independent display of position values.

The global variable `"linearUnitConversion"` is set by the Tcl commands `emc_linear_unit_conversion` to correspond to either `"inch"`, `"mm"`, `"cm"`, `"auto"`, or `"custom"`. This forces numbers to be returned in the units specified, in program units when `"auto"` is set, or not converted at all if `"custom"` is specified.

Ditto for `"angularUnitConversion"`, set by `emc_angular_unit_conversion` to `"deg"`, `"rad"`, `"grad"`, `"auto"`, or `"custom"`.

With no args, `emc_linear/angular_unit_conversion` return the setting.

The functions `convertLinearUnits` and `convertAngularUnits` take a length or angle value, typically from the `emcStatus` structure, and convert it as indicated by `linearUnitConversion` and `angularUnitConversion`, resp.

Chapter 7

IOSH

7.1 Adding Extra Inputs and Outputs.

The PLC is possibly the weakest part of the EMC. Right now, there are two supplied "PLCs" that control the discrete I/O (estop, lube, coolant, spindle): `bridgeportio` and `minimillio`. `minimillio` just controls our NIST desktop `minimill` that doesn't have coolant or lube, so it's a stripped-down version of `bridgeportio`. `bridgeportio` is written in C++, not a very common PLC programming language. We did this since we have in-house tools for programming hierarchical control systems using C++, which we have used for many other projects, so it made sense for us.

For EMC users, it's not a very popular choice. `Bridgeportio` and `minimillio` are slightly customizable (without resorting to C++ programming) via the `.ini` file, by setting the location and polarity of input/output bits and some other parameters like delays for spindle brake engage/release. Adapting them for anything else, like an automatic tool changer, means coding in C++ or replacing them with something else.

Looking at the EMC I/O controller as a black box, there are three things it must do:

1. [TOP] Create NML buffers for commands to it from the EMC task controller and status from it to the EMC task controller. This is what connects it to the EMC.
2. [MIDDLE] Read NML commands that come in from the EMC task controller (e.g., coolant on, etc.), do what's requested, and write status back (done, executing, error, and values like coolant is on/off, etc.)
3. [BOTTOM] Talk to real-world I/O points.

As an alternative to the C++-based `bridgeportio`, we wrote a Tcl/Tk-based PLC. Here's what we did:

1. Extended the Tcl/Tk windowing shell "wish" with EMC-specific commands, creating the "IO shell" `iosh`. This was done similarly to how we built the EMC shell `emcsh`, used for building Tcl/Tk GUIs like `tkemc.tcl`. `iosh` creates NML buffers (requirement (1) above, the TOP), and adds Tcl words "inb" and "outb" for doing PC-bus byte I/O (requirement (3) above, the BOTTOM). `iosh` can be thought of as the PLC engine, that can be programmed in Tcl/Tk to implement any arbitrary PLC. It doesn't have any particular PLC logic program, for say the `Bridgeport`. You have to write this

yourself, in Tcl/Tk, as you would write a ladder logic program for another PLC.

2. Wrote a Tcl/Tk script that does the actual PLC logic (requirement (2) above, the MIDDLE). This is tkio.tcl. It implements the Bridgeport I/O controller, just like bridgeportio. It's plug-compatible with bridgeportio: you can specify "tkio" instead of "bridgeportio" in the .ini file, e.g.,

```
[EMCIO] EMCIO = tkio
```

I tried it out a while ago and it looked like it worked, but the "plug compatibility" might not be 100% due to bugs. I'll have to check it out on our machine.

Note that since we're extending wish to build iossh, we get some GUI stuff automatically. That is, in tkio.tcl, there's some script code for popping up a window with some red and green LEDs that show IO status. So, iossh lets you program both the PLC logic and the GUI appearance in the same script. So, unlike bridgeportio, which doesn't pop anything up, iossh pops up a blank canvas that your script can populate with whatever you program. If you choose to write just a straight PLC program, the canvas is just a little annoying blank square.

If you want to use a PC as a PLC without running the EMC, you can still use iossh and your own script. The NML buffers will be created and simply ignored, and you can write your PLC and accompanying GUI for any purpose whatsoever. Or, you can take iossh.cc, strip out all the NML stuff, and just leave the inb/outb commands there for your own stripped-down shell. Or, you can use wish out of the box, write two short C programs that create "inb" and "outb" (we did this), put the in /bin, and call these within wish. This shows how a Linux box can be converted into a PLC in about 10 minutes. Of course, you program in Tcl/Tk, not ladder logic. Now, I like Tcl/Tk and if I had to write a PLC with a gun to my head I'd use wish and the /bin/{inb,outb} programs real quick-like.

7.2 Working with iossh

Tcl command extensions for EMC IO

The following information has been extracted from iossh.cc. For examples of the correct usage, look at the tcl scripts in emc/tcl/scripts.

To use the iossh commands, use the following at the start of the script:

```
#!/bin/sh # the next line restarts using iossh \ exec plat/nonrealtime/bin/iossh "$@" "$@"
```

Using iossh:

```
iossh {<script>} [-i <ini file>]
```

With filename, it opens NML buffers to the EMC IO, runs the script, closes the buffers, and quits.

Without filename, it runs interactively.

With -i <inifile>, uses inifile instead of emc.ini. Note that the two dashes prevents Tcl from looking at the remaining args, which would otherwise trigger a Tcl error that it doesn't understand what -i means.

7.3 EMC IO commands:

- `emc_io_connect` `emc_io_disconnect` Open or close the NML buffers to the command in, status out, and error out. Returns 0 if OK, or -1 if not.
- `emc_io_read_command` Peek the NML command buffer. Returns 0 if OK, -1 if not.
- `emc_io_get_command` Puts the command string, e.g., "emc_aux_estop_off", or "none". Returns 0.
- `emc_io_get_command_type` Puts the command NML number. Returns 0.
- `emc_io_get_serial_number` Puts the command serial number. Returns 0.
- `emc_io_write_status` Write the EMC_IO_STAT structure out to NML. Returns 0 if OK, -1 if error.
- `emc_io_write_error` Write the error string to the error NML buffer. Returns 0 if OK, -1 if error.

7.3.1 IO status commands, set associated field in the NML status structure

- `emc_io_status_heartbeat` <number>
- `emc_io_status_echo_serial_number` <number>
- `emc_io_status_status` done | exec | error
- `emc_io_status_estop` on | off
- `emc_io_status_mist` on | off
- `emc_io_status_flood` on | off
- `emc_io_status_lube` on | off
- `emc_io_status_lube_level` ok | low
- `emc_io_status_spindle_speed` <speed>
- `emc_io_status_spindle_enabled` on | off
- `emc_io_status_spindle_direction` <pos> <neg> 0
- `emc_io_status_spindle_increasing` <pos> <neg> 0
- `emc_io_status_spindle_brake` on | off
- `emc_io_status_tool_prepped` <number>
- `emc_io_status_tool_in_spindle` <number>

7.3.2 IO commands:

- `inb <address>` Reads and returns the byte at `<address>`. If address begins with `0x`, it's interpreted as a hex number, otherwise it's decimal.
- `outb <address> <value>` Writes the byte `<value>` to `<address>`. If address or value begins with `0x`, it's interpreted as a hex number, otherwise it's decimal. Returns nothing.
- `inw <address>` Reads and returns the short at `<address>`. If address begins with `0x`, it's interpreted as a hex number, otherwise it's decimal.
- `outw <address> <value>` Writes the short `<value>` to `<address>`. If address or value begins with `0x`, it's interpreted as a hex number, otherwise it's decimal. Returns nothing.
- `inl <address>` Reads and returns the long at `<address>`. If address begins with `0x`, it's interpreted as a hex number, otherwise it's decimal.
- `outl <address> <value>` Writes the long `<value>` to `<address>`. If address or value begins with `0x`, it's interpreted as a hex number, otherwise it's decimal. Returns nothing.

Chapter 8

TKIO

8.1 Introduction

Tkio is a Bridgeportio that is written in Tcl/Tk. It includes a small X-Windows widget that displays some of the internal states of the system created when tkio is run. Tkio uses an extended wish shell. This shell, iosh, is written in C++ and provides a connection to the NML used by the EMC. It also permits direct inspection and manipulation of I/O ports.

Tkio creates a system that approximates a state machine. It does this by setting a number of global variables and then working from the values assigned to these each time a relevant NML command is seen. These variables are grouped by specific kind of task. Each group includes a ?group?Command, ?group?Status, and ?group?State variable. There is also a global variable (done_status) that indicates whether tkio is executing (exec), has raised an error (error), or is done (done). Tkio broadcasts this variable's value to the NML structure so that other processes can set their own wait state based on its state.

A hierarchy of processes is implemented.

1. The loop process, doit is the basic controller. It watches for NML commands, sorts them into groups and sets a new message flag (NEW_COMMAND) in each group. It also sends it's own state command back to the NML channel.
2. The second level functions are based around each group of tasks. These include spindle, tool, lube, cool, and estop. An unknown command category is included for unexpected messages or error messages. Each of these groups has a supervisory process named for the group (?group?Controller) that interprets a command to the group and triggers a specific group logic process based on the meaning of the command.
3. The lowest level group processes encapsulate specific actions that are performed when the process is commanded. These processed include standard commands like abort so that when an abort command is found the ?group?Abort processes are triggered. The specific processed included in a group depend upon the nature of the group. In general they include init and halt processes. It is at this level that NML commands are issued to complete an action.

There are also processes included that read and write to parports using inb and outb and a process that reads the ini file and uses variable names and values from that file to determine the pin number and polarity for a signal. A listing of all the processes in tkio in the order that they are written is included as Section 8.9

This system when it is called to run, creates it's own copy of a tcl interpreter. What this means is that variables named here and processes created here are not normally available to other tcl systems that might also be running. This system connects to the EMC only through NML. This means that a command to turn on mist coolant issued by tkemc gets put into the NML channel by emcsh. It is then read by iossh and picked up by tkio. While this logic may seem a bit round about it is consistent with the NIST philosophy of making all of the parts of a running EMC available on a common communication channel.

8.2 Start Tickle Using

A standard method has been developed for starting either a tcl shell or a wish shell from an executable file. For tkio, these first three lines are;

```
#!/bin/sh
# the next line restarts using iossh \
exec plat/UNKNOWN_PLAT/bin/iossh "$0" "$@"
```

During the standard install of an EMC the UNKNOWN_PLAT in line three gets replaced with the proper directory name as the emc/src/emcio file is copied into that bin directory. If you plan to commit your edited tkio to the repository it is a good idea to follow this pattern within your file. You will need to edit the emc/scripts/generic/install file in order to be certain that your file gets copied correctly.

A quick and dirty work around for this is to use the nonrealtime link that is made during a startup. In that case the path to insert here is

```
exec plat/nonrealtime/bin/iossh "$0" "$@"
```

8.3 IniLoad

This process uses an extended command emc_ini to query the ini file used by the current run. It creates tcl variables using the list included in the foreach section of the process. These names are the same names that are included in the ini file in the EMCIO section.

IniLoad takes a list of {<section> <var> <default value>} and calls emc_ini to load them with their specified value or the default value if an ini value was not found. Table 8.1 shows the stock list of ini variables read in when this process is called. During the first foreach loop, section becomes EMCIO, var becomes CYCLE_TIME, and def becomes 0.100. This means that for this loop, emc_ini looks in the EMCIO section of the ini associated with the run and tries to find a variable there named CYCLE_TIME.

By customizing the list in this process, the user can select any ini file variable. It would be possible to add variables to the emcio section of the ini file that are read and used only by the edited tkio.

Table 8.1: Ini Variables List For iniLoad

EMCIO CYCLE_TIME 0.100	EMCIO SPINDLE_INCREASE_INDEX 9
EMCIO TOOL_TABLE smhome.tbl	EMCIO ESTOP_WRITE_INDEX 10
EMCIO PARPORT_IO_ADDRESS 0x278	EMCIO SPINDLE_BRAKE_INDEX 11
EMCIO SPINDLE_OFF_WAIT 1.0	EMCIO SPINDLE_ON_INDEX 3
EMCIO SPINDLE_ON_WAIT 1.5	EMCIO SPINDLE_FORWARD_POLARITY 0
EMCIO ESTOP_SENSE_INDEX 1	EMCIO SPINDLE_REVERSE_POLARITY 0
EMCIO LUBE_SENSE_INDEX 2	EMCIO MIST_COOLANT_POLARITY 0
EMCIO ESTOP_SENSE_POLARITY 1	EMCIO FLOOD_COOLANT_POLARITY 0
EMCIO LUBE_SENSE_POLARITY 1	EMCIO SPINDLE_DECREASE_POLARITY 1
EMCIO SPINDLE_FORWARD_INDEX 1	EMCIO SPINDLE_INCREASE_POLARITY 1
EMCIO SPINDLE_REVERSE_INDEX 0	EMCIO ESTOP_WRITE_POLARITY 1
EMCIO MIST_COOLANT_INDEX 6	EMCIO SPINDLE_BRAKE_POLARITY 0
EMCIO FLOOD_COOLANT_INDEX 7	EMCIO SPINDLE_ENABLE_POLARITY 1
EMCIO SPINDLE_DECREASE_INDEX 8	

8.4 Parport I/O

The initial application of the EMC to a Bridgeport mill by Matt Shaver used a parport to control the auxiliary functions. Since the motion control was through a Servo-To-Go card, this left a PC's parport that could be used. It was this decision that led to the development of bridgeport task and bridgeportio. Tkiio uses the same task controller but implements the same parport interface as bridgeportio. Three tkiio processes use inb and outb to query and set pins.

Three addresses are included in a standard parport. The base address connects with the eight primary output pins. Base plus one addresses the five input pins. Base plus 2 allows you to address an extra four output pins. Two of the parport processes in tkiio are written to read and write all output pins. The third reads the input pins.

8.4.1 parportSetBit <bit> <val>

Sets bit 0-11 to 0 or 1. This uses the PARPORT_IO_ADDRESS global as the base. Bits 8-11 get mapped to PARPORT_IO_ADDRESS + 2 automatically.

8.4.2 parportCheckBit <bit>

Reads bit 0-11 and returns 0 or 1. This uses the PARPORT_IO_ADDRESS global as the base. Bits 8-11 get mapped to PARPORT_IO_ADDRESS + 2 automatically.

8.4.3 parportGetBit <bit>

Reads bit 0-4 and returns 0 or 1. This uses the PARPORT_IO_ADDRESS global as the base, read the next byte up, and shifts the result right by 3 to align the input bits properly.

Using a similar arrangement of new processes it is possible to make tkiio read and write to almost any arrangement of I/O ports or cards.

8.5 doit

This process is the central loop at the heart of the tkio system. It watches NML for commands and directs those commands to the proper process for execution.

8.5.1 Global Variables

Doit sets up and uses a group of global variables

```
global command_string command_type command_number
global echo_serial_number done_status heartbeat
global cycletime
global toolCommand toolState toolStatus
global spindleCommand spindleState spindleStatus
global coolantCommand coolantState coolantStatus
global auxCommand auxState auxStatus
global lubeCommand lubeState lubeStatus
```

With the exception of the NML related variables on the first line these are pretty obvious. Values for these are shown in the relevant sections.

8.5.2 NML

The NML read is done by the following set of commands.

```
if {[emc_io_read_command] == 0} {
    set command_string [emc_io_get_command]
    set command_type [emc_io_get_command_type]
    set command_number [emc_io_get_serial_number]
```

Each of these commands are defined in `iosh.cc`. Those definitions are listed below.

- `emc_io_read_command` – Peek the NML command buffer. Returns 0 if OK, -1 if not.
- `emc_io_get_command` – Puts the command string, e.g., "emc_aux_estop_off", or "none". Returns 0.
- `emc_io_get_command_type` – Puts the command NML number. Returns 0.
- `emc_io_get_serial_number` – Puts the command serial number. Returns 0.

Since each of these commands return the value indicated, tkio sets that return value as the value of the variable in each tcl set command.

Once these variables have been set, the process compares the new serial number with the old and if different it assumes a new command. It then copies the type and number into global variables and reports to the NML that it has found that command.

```

if {$command_number != $echo_serial_number} {
    # got a new command
    # copy the command type into NML status
    emc_io_status_command_type $command_type
    # save the serial number for checking new com-
mands next time
    set echo_serial_number $command_number
    # copy the serial number into NML status
    emc_io_status_echo_serial_number $echo_serial_number
}

```

8.5.3 Matching the command string

In the code listed below, doit matches the command string against a set of expected string values and finds the group to which the command belongs. When it finds a group it sets that ?group?State variable to the value NEW_COMMAND.

```

if {[string length $command_string] > 0} {
    if {[string match "emc_tool_*" $com-
mand_string]} {
        set toolCommand $command_string
        set toolState NEW_COMMAND
    } elseif {[string match "emc_spindle_*" $com-
mand_string]} {
        set spindleCommand $command_string
        set spindleState NEW_COMMAND
    } elseif {[string match "emc_coolant_*" $com-
mand_string]} {
        set coolantCommand $command_string
        set coolantState NEW_COMMAND
    } elseif {[string match "emc_aux_*" $com-
mand_string]} {
        set auxCommand $command_string
        set auxState NEW_COMMAND
    } elseif {[string match "emc_lube_*" $com-
mand_string]} {
        set lubeCommand $command_string
        set lubeState NEW_COMMAND
    } else {
        # FIXME--
        handle unknown commands better, perhaps with
        # emc_io_write_error
        puts stdout [format "unknown com-
mand \"%s\"" $command_string]
    }
}

```

8.5.4 Running the controllers

Doit has now sorted out the nature of the command and is ready to run the I/O controller processes. These include;

- toolController
- spindleController

- coolantController
- auxController
- lubeController

At this point, the tcl interpreter is a linear device so it will execute each of these processes and the processes that they call in sequence. When this set of task is complete the main control actions have been done.

8.5.5 Housekeeping

There are two major housekeeping tasks that doit also initiates. The first of these is to write the status of tkio to NML using this set of commands.

```
incr heartbeat
emc_io_status_heartbeat $heartbeat
emc_io_write_status
```

It then updates local labels by looking at the values in each ?group?Status variable and setting the value of the variable done_status. It reports this status to NML using

```
emc_io_status_status $done_status
```

and sets itself to run again after the value set in the variable cyletime.

8.6 Group Controllers

A group controller includes branching for all of the possible NML commands that could be issued to that group. Based on the NML command found the controller will execute an action process that includes the logic necessary. Then having completed the action, the controller calls other processes to check the status and report it back to NML. A quick look at the coolantController process will illustrate how this is done.

```
proc coolantController {} {
    global coolantCommand

    set cmd [lindex $coolantCommand 0]

    if {![string compare $cmd "emc_coolant_init"]} {
        coolantInit
    } elseif {![string compare $cmd "emc_coolant_halt"]} {
        coolantHalt
    } elseif {![string compare $cmd "emc_coolant_abort"]} {
        coolantAbort
    } elseif {![string compare $cmd "emc_coolant_mist_on"]} {
```

```

        coolantMistOn
    } elseif {![string compare $cmd "emc_coolant_mist_off"]} {
        coolantMistOff
    } elseif {![string compare $cmd "emc_coolant_flood_on"]} {
        coolantFloodOn
    } elseif {![string compare $cmd "emc_coolant_flood_off"]} {
        coolantFloodOff
    }
}

```

Again the tcl interpreter is working in linear fashion so the process called by this if statement will complete before the commands below are executed.

```

emc_io_status_mist [isMist]
emc_io_status_flood [isFlood]

```

These processes check the value of these I/O pins and report the value to NML. Once again we see the dependence on testing the value rather than assuming the value based on an executed command. This structure allows for multiple processors and multiple processes all working on the same EMC and reporting the status of it correctly at any location.

8.7 Group Processes

The group processes are where the final sorting of commands is done and real changes are made to the state of output pins. Again we will use the coolant group as an example. In section 8.6 we saw a coolant NML command get branched into init, halt, abort, mist on, mist off flood on, and flood off. Each of these branches are represented by a process below. Note that init and halt call the do?? processes with the same argument value. The net effect is to turn off all coolant. The abort command does nothing here except reset its variables in anticipation of the next command. It could also be made to turn off all coolant by adding the do?? commands with the appropriate value in their arguments.

```

proc coolantInit {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        doMist 0
        doFlood 0
        set coolantState S0
        set coolantStatus DONE
    }
}

proc coolantHalt {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {

```

```

        doMist 0
        doFlood 0
        set coolantState S0
        set coolantStatus DONE
    }
}

proc coolantAbort {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        set coolantState S0
        set coolantStatus DONE
    }
}

```

The four processes below pass an on or off argument to the do?? processes.

```

proc coolantMistOn {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        doMist 1
        set coolantState S0
        set coolantStatus DONE
    }
}

proc coolantMistOff {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        doMist 0
        set coolantState S0
        set coolantStatus DONE
    }
}

proc coolantFloodOn {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        doFlood 1
        set coolantState S0
        set coolantStatus DONE
    }
}

proc coolantFloodOff {} {
    global coolantState coolantStatus

    if {$coolantState == "NEW_COMMAND"} {
        doFlood 0
        set coolantState S0
    }
}

```

```

        set coolantStatus DONE
    }
}

```

The two do?? processes below are the work horses in this group. They take the argument and turn mist or flood on or off using the default values that were found by iniLoad.

```

proc doMist {on} {
    global MIST_COOLANT_INDEX MIST_COOLANT_POLARITY
    if {$on} {
        parportSet-
    Bit $MIST_COOLANT_INDEX $MIST_COOLANT_POLARITY
    } else {
        parportSet-
    Bit $MIST_COOLANT_INDEX [expr ! $MIST_COOLANT_POLARITY]
    }
}

proc doFlood {on} {
    global FLOOD_COOLANT_INDEX FLOOD_COOLANT_POLARITY

    if {$on} {
        parportSet-
    Bit $FLOOD_COOLANT_INDEX $FLOOD_COOLANT_POLARITY
    } else {
        parportSet-
    Bit $FLOOD_COOLANT_INDEX [expr ! $FLOOD_COOLANT_POLARITY]
    }
}

```

In similar fashion, the is?? processes look to see what the state of a pin is and returns on or off depending on the pin and polarity values found by iniLoad.

```

proc isMist {} {
    global MIST_COOLANT_INDEX MIST_COOLANT_POLARITY

    if {[parportCheck-
    Bit $MIST_COOLANT_INDEX] == $MIST_COOLANT_POLARITY} {
        return on
    }
    return off
}

proc isFlood {} {
    global FLOOD_COOLANT_INDEX FLOOD_COOLANT_POLARITY

    if {[parportCheck-
    Bit $FLOOD_COOLANT_INDEX] == $FLOOD_COOLANT_POLARITY} {
        return on
    }
    return off
}

```

Some groups do not require all of the intermediate processes that are required in the coolant group because it includes both mist and flood commands. Other groups like spindle may include many more processes in order to sort out the specific action to be taken on the basis of a single NML command.

8.8 Widgets

The widgets associated with tkio are really unnecessary but they provide some visual confirmation that the system is alive and working. It is easily possible to add widgets to the display so that you can test additional processes or view NML commands or see variable values during a run. The layout is detailed below.

8.8.1 Create the main window top frame

```
set top [frame .top]
pack $top -side top -fill both -expand true
```

8.8.2 Build the top menu bar

```
set menubar [menu $top.menubar -tearoff 0]
set filemenu [menu $menubar.file -tearoff 0]
$menubar add cascade -label "File" -
menu $filemenu -underline 0
$filemenu add command -label "Exit" -
command {after cancel doit; destroy . ;
exit} -accelerator Alt+X -underline 1
bind . <Alt-
x> {after cancel doit ; destroy . ; exit}
set helpmenu [menu $menubar.help -tearoff 0]
$menubar add cascade -label "Help" -
menu $helpmenu -underline 0
$helpmenu add command -label "About..." -
command {popupAbout} -underline 0

. configure -menu $menubar
```

8.8.3 Create frames to hold label and value pairs and create labels

```
frame $top.iohb
label $top.iohb.lab -text "Heartbeat:" -anchor w
label $top.iohb.val -textvariable heartbeat -
anchor e
frame $top.iocmd
label $top.iocmd.lab -text "Command:" -anchor w
label $top.iocmd.val -textvariable command_type -
anchor e
frame $top.ionum
label $top.ionum.lab -text "Command #:" -anchor w
```

```

label $stop.ionum.val -
textvariable echo_serial_number -anchor e
frame $stop.iostatus
label $stop.iostatus.lab -text "Status:" -anchor w
label $stop.iostatus.val -textvariable done_status -
anchor e

```

8.8.4 Use to pack geometry manager to place widgets in top frame

```

pack $stop.iohb $stop.iocmd $stop.ionum $stop.iostatus -
side top -fill x
pack $stop.iohb.lab -side left
pack $stop.iohb.val -side right
pack $stop.iocmd.lab -side left
pack $stop.iocmd.val -side right
pack $stop.ionum.lab -side left
pack $stop.ionum.val -side right
pack $stop.iostatus.lab -side left
pack $stop.iostatus.val -side right

```

8.8.5 Adding widgets

Widgets can easily be added. The example code below adds a frame and two labels. The value shown is parport address used by tkio.

```

frame $stop.addr
label $stop.addr.lab -text "parport address" -
anchor w
label $stop.addr.val -
textvariable PARPORT_IO_ADDRESS -anchor e
pack $stop.addr -side top fill x
pack $stop.addr.lab -side left
pack $stop.addr.val -side right

```

It is also very easy to add buttons that call processes. The next examples might be the start of an auxiliary machine control panel.

```

frame $stop.mist
button $stop.mist.on -text "Mist On" -
command {doMist 1}
button $stop.mist.off -text "Mist Off" -
command {doMist 0}
pack $stop.mist -side top -fill x
pack $stop.mist.on -side left
pack $stop.mist.off -side right

```

Your imagination is probably the limiting factor in the development of widgets for tkio.

8.9 Process List from tkio in the order written

8.9.1 parport processes

```
proc parportSetBit {bit val}
proc parportCheckBit {bit}
proc parportGetBit {bit}
```

8.9.2 Tool processes

```
proc toolController {} {
proc toolInit {} {
proc toolHalt {} {
proc toolAbort {} {
proc toolPrepare {tool} {
proc toolLoad {} {
proc toolUnload {} {
```

8.9.3 Spindle processes

```
proc doSpindleDirection {forward} {
proc doSpindleIncrease {increase} {
proc doSpindleBrake {on} {
proc isSpindleDirection {} {
proc isSpindleIncrease {} {
proc isSpindleBrake {} {
proc spindleController {} {
proc spindleInit {} {
proc spindleHalt {} {
proc spindleAbort {} {
proc spindleOn {speed} {
proc spindleOff {} {
proc spindleForward {} {
proc spindleReverse {} {
proc spindleStop {} {
proc spindleIncrease {} {
proc spindleDecrease {} {
proc spindleConstant {} {
proc spindleBrakeRelease {} {
proc spindleBrakeEngage {} {
```

8.9.4 Coolant processes

```
proc doMist {on} {
proc doFlood {on} {
proc isMist {} {
proc isFlood {} {
proc coolantController {} {
proc coolantInit {} {
proc coolantHalt {} {
proc coolantAbort {} {
proc coolantMistOn {} {
proc coolantMistOff {} {
```

```
proc coolantFloodOn {} {  
proc coolantFloodOff {} {
```

8.9.5 Estop processes

```
proc doEstop {on} {  
proc isEstop {} {  
proc checkEstop {} {  
proc auxController {} {  
proc auxInit {} {  
proc auxHalt {} {  
proc auxAbort {} {  
proc auxEstopOn {} {  
proc auxEstopOff {} {
```

8.9.6 Lube processes

```
proc isLubeLevel {} {  
proc lubeController {} {  
proc lubeInit {} {  
proc lubeHalt {} {  
proc lubeAbort {} {  
proc lubeOn {} {  
proc lubeOff {} {
```

8.9.7 Misc processes

```
proc popupAbout {} {  
proc doit {} {  
proc iniLoad {} {
```


Appendix A

README

A.1 Organization of this directory

This file has been produced from a master that is written using LyX and is kept in the documents module at Sourceforge.net. That document is written using a plain book layout and the bookman font. This SourceForge module contains the EMC documentation.

A.2 Master LyX Files

There are three main or master files that are used to build the three handbooks.

- *Master_Developer.lyx*
- *Master_Integrator.lyx*
- *Master_User.lyx*

These are the master docs from which it is hoped that future handbook publications in HTML and PDF formats will be produced. These files use the LyX include command to put together sets of files which serve as chapters or appendices within each. Other combinations of these chapters could be developed using similar master files.

A.3 Chapter Files

Chapter files intended for the user handbook have been named *User_???.lyx* to reflect the notion that they are intended to be placed within that handbook. Dev and Int are used in similar matter for chapters intended for those handbooks as well.

Each chapter file is intended to cover a logical unit. Each begins with a single chapter name.

A.4 General Files

Several general files like the glossary, faq, and copyright chapters are also included here and can be used in each of the master documents.

A.5 LyX Versions

LyX 1.1.6fix4-1 was used to write these docs - If you wish to add to them, LyX can be downloaded from www.lyx.org.

The latest release lyx-1.3.2 will read the docs, but according to the press release, files saved with lyx-1.2+ can not be read by lyx-1.1.6. Please bare this in mind should you wish to add to this repository.

A.6 LyX Notes

A.6.1 Path Names

LyX seems to be unable to handle path names starting with `../` or `~/` so make a symbolic link `/usr/share/lyx/EMC_images/` pointing to your documents/images/ directory. When generating a html document, absolute path names will be used, so you might want to do a search and replace on `/usr/share/lyx/EMC_images`.

A.6.2 Thoughts for contributors

There are a few notes below for developers who wish to edit this text.

1. Cut'n paste table and figure methods to the relevant chapters.
2. Although *.eps graphics are claimed to be required for rendering - These are spec'ed as *png in HTML output - Images can in fact use any valid extension which will be echoed in the output.
3. Bare in mind that if a pdf document is required, the images need to be in a postscript format first.
4. Perhaps an effort should be made to convert ALL images to png as a matter of course.

A.6.3 Image conversion

Conversion of images is a simple matter of using Imagemagic from the command line with :-

```
convert image.gif image.eps
convert image.gif image.png
```

Using anything other than *.eps can cause problems with ghostscript on startup, and also some problems with generating pdf docs.

A.6.4 Fonts

- Bookman is the default font used in all EMC handbooks.
- Normal text - block as standard 10 point.
- Excerpts from ini files and sources - Smaller or Smallest Typewriter
- Commands typed in at a command line - Smaller or Smallest Typewriter Italic
- Output from above commands - Smaller or Smallest Typewriter Black

A.6.5 White space

When more than one blank space is required between characters, use one or more “Protected Blank” from Insert -> Special Character. You can also add these using `<control><spacebar>` from the keyboard. This will then be displayed correctly in the final printout. Failure to use Protected Blanks will result in ‘strange’ characters being shown and printed in pdf documents!

A normal enter starts a new paragraph in *LyX*. These will show up with extra lines between if you are using block mode or as an indented line if you are using that layout. To avoid extra lines being inserted between each line of code or command output, terminate each line with a LineBreak (Ctrl-Return) instead of the usual Return.

It is not a good idea to use Protected Blanks at the beginning of a paragraph as they do not get reproduced very well.

Appendix A

Legal Section

Handbook Copyright Terms

Copyright (c) 2000 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and one Back-Cover Text: "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

A.1 GNU Free Documentation License Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.1.1 GNU Free Documentation License Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a

way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, \LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

A. 1. GNU FREE DOCUMENTATION LICENSE VERSION 1.1, MARCH 200077

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms

of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine

A.1. GNU FREE DOCUMENTATION LICENSE VERSION 1.1, MARCH 200079

any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B

Index

Index

SSH, 10
ssh, 10–12, 15