# Linux development on the PlayStation 3, Part 3: Slimming down X11 with tiny tools

## Renders great, less filling

Skill Level: Intermediate

Peter Seebach (developerworks@seebs.plethora.net)
Freelance author
Plethora.net

08 Apr 2008

The Sony PlayStation 3 (PS3) runs Linux®, but getting it to run well requires some tweaking. In the third and final article of this series, on PS3 Linux, Peter Seebach talks about ways to get X11 slimmed down to fit on a smaller memory budget.

In Parts 1 and 2, you saw how to use the runlevel system and related tools to dramatically reduce memory usage on a PS3 running Linux, leaving more memory available for compilation and similar services. To wrap this up, let's look at a few more things that are worth doing, including at least one fairly hefty task: getting X11's footprint trimmed.

The problem with X11 is that you may well need it. This is a problem because the lowest memory footprint I saw for the X server itself on my PS3 was around 40MB, which is way, way, too much of available memory to sacrifice just for some pretty pictures. (Yes, I did learn to use UNIX® on a plain text terminal. Why are you looking at me funny?) That said, it really is very useful sometimes, and there are programs that don't make sense to run without graphics.

## Run X11 on another machine

So you need X11, but the PS3 doesn't have enough memory to run it comfortably. You have one sneaky option: run over the network. X clients could be configured to run on your PS3, with an X server on a remote system. This isn't really exceptionally

fast—the hypervisor seems to slow down networking a bit from the nominal gigabit performance the hardware has—but if you don't need the highest possible speed, it's actually pretty livable. You have a few options when setting this up. First, you're going to need to run an X server on another machine.

### About this series

This series of three articles looks at PS3 Linux as a prospective development environment.

This first article, Part 1, introduces the basic configuration knobs and widgets specific to the PS3, shows you how to use them effectively, and suggests the kind of trickery that might get improved performance or a more usable display.

Part 2 and this article delve into some of the performance and tuning issues that, while they might apply on any system, are particularly useful for turning your PS3 from a proof-of-concept demo into a working system.

There's a never-ending rumor that OS X will run on a PS3, because one of the Cell/B.E. technology demo movies was once played on a Mac at a trade show, and people keep posting a picture of this and insisting that it's proof. Well, time to make that even worse; I'll use the native X11 server on an OS X machine to demonstrate this. In fact, X is nicely portable, and the essential tricks are the same everywhere.

First, start your X server, and get yourself a prompt; run a terminal program, of whatever sort you like. I picked xterm. Now, there are two ways forward. One is to use ssh's X tunneling feature to let you forward X requests from the client (that's the PS3) to the server (that's the Mac, in this case). The other is to use X directly over the wire. They each have strengths and weaknesses; in the case of a local network, over the wire may be easier to explain and use. First, figure out what your X server's DISPLAY is. In your X terminal, echo the environment variable DISPLAY; it'll probably be ":0.0". This means that your X server is running a display named "0.0". The colon separates the hostname from the display name. If your laptop were named "laptop," the full name of the display would be "laptop:0.0". (When the name is omitted, X uses clever hacks to access a local display faster.)

### Wait, which one's the server?

In most of computing, the "server" is the program that doesn't have a direct interface to users, and the "client" is the program with the graphical interface. However, in X, this is reversed—and if that confuses you, don't feel too bad. A lot of users have been confused by it at first.

To understand why the graphical program is the "server," and the program doing all the computation is the "client," ask yourself the question: What is it that is being provided? The X server is providing a graphical display. Thus, the server handles incoming requests from clients such as "draw me some pixels" or "give me a window,"

and the clients use these services as they wish.

Now, if you were to just pop over to the PS3, and set the environment variable DISPLAY to, for instance, "laptop:0.0", and run an xterm, you'd discover a tragic flaw in this scheme: Permission denied. By default, X will not allow arbitrary clients on a remote host to run on the local display. This is a security feature; obviously, you'll want to subvert it. The simplest way to do this is to allow clients on your PS3 to access your X server. On the local machine, run the command `xhost +<machine>`, where `<machine>` is the hostname or IP address of your PS3. I never bothered to set up DNS for my unroutable dynamic block, so I used `xhost +10.10.10.134`. With that done, running X commands on the PS3 works; they show up on the Mac, without the substantial memory overhead of an X server running on the PS3.

## Forward X11 requests

Now, there's another option. The `ssh` command (you did leave sshd running, right?) can forward X11 requests. When you run `ssh` with access to an X server, passing it the `-X` option causes it to forward X requests; in fact, the `-Y` option may be preferable, as it enables "trusted" forwarding, which bypasses more security features. (You want to bypass them because otherwise you may not be able to, say, open windows.) Optionally, you might look at the `-C` option, which specifies compression of data, including X11 packets. This is useful on slow networks, but not so useful on fast networks. Try it both ways to see. The syntax for this one is easier in some ways; just `ssh -X <ps3>`, and on the PS3, start running commands that use X. This way, the PS3 doesn't have the memory overhead of the server, only that of the clients. Time to start konquerer, right? But konqueror takes up about 20MB, plus 7.5MB for kded, 5.8MB for klauncher, 5MB for kio_file, 4.7MB for kdeinit...

This, of course, highlights a secondary problem: even without the server, X apps can chew up a lot of memory. Offloading the server does improve available memory, but it doesn't completely eliminate the problem.

## Try to trim the server

Remote access just may not be fast enough, or you may not have a convenient X server to use. There is another option: run the server locally, but without KDE or Gnome. To do this, you'll probably have to omit the runlevel 5 X display manager and login window; instead, log in on a console and start X yourself. The classic way to do this is to just run the `xinit` program, which runs the commands in your .xinitrc file, found in your home directory. Note that these commands are run sequentially; if you want more than one program at once, the first few need to be backgrounded (place an & at the end of the line). For instance, here's a .xinitrc that will do you well:

**Listing 1. X server, plain, hold the mustard**

```
xterm &
exec twm
```

X terminates when the program it was running as its "session" terminates; in this case, that's the twm program that the xinitrc exec'd as its last command. Some people prefer to use an xterm as the session program, and run the window manager in the background. It's up to you.

If you use that .xinit file, and run `xinit`, you'll likely see a grey stippled background, and a mouse cursor with a sort of odd nine-paneled, uh, thing. That's an xterm, waiting for you to decide where to place it. The twm window manager's primary feature is that it's tiny; it doesn't do much for you, it mostly trusts you to know what you want. On the other hand, its total memory footprint of a tad over 2MB is noticeably smaller than that of something larger, like Gnome or KDE. On my system, the xterm program actually took up more space than twm, although the bulk of the space went to the X server, with a virtual size of 62MB, and a resident set of 34MB actually in RAM. That's a bit large. What can we do about it?

Not much. The first obvious thing to do would be to disable modules. If you've ever had to do this before, you're doubtless familiar with the process. Go into /etc/X11, edit xorg.conf (you need root privileges to do this), and comment out the modules you don't need.... Only there aren't any modules in xorg.conf. In modern X.org servers, the default behavior is to load everything and see what sticks. Conveniently, it's easy to override this; if you provide a Modules section, only the things you ask for will be loaded. On a system with no functioning graphical hardware, the various GL-related options are pretty much worthless; here's what I ended up using:

**Listing 2. Fewer modules**

```
Section "Module"
        Load "extmod"
        Load "type1"
        Load "freetype"
EndSection
```

That doesn't actually change things much, I'm afraid, It got my X server from 34MB down to, wait for it, 33MB. The real problem, of course, is the huge amount of memory used by the internal buffer that's being written onto the framebuffer device: 1280x768 or so, and the framebuffer only works in 32-bit mode, really. Reducing the size of the framebuffer actually can help quite a bit—at substantial cost to your screen real estate, of course. For instance, switching to a 480p, non-fullscreen mode (which is substantially less than 480 pixels tall), I found that X was using only about 10MB of real memory; even 720p, non-fullscreen, used less (25MB) than I was

getting in WXGA mode. If you're close to the edge, it might be worth scaling back a bit. It's just a shame the PS3 doesn't have a broader range of supported video modes.

## Other ways to trim space

There's a little more you can do to trim space. If you aren't using the multiple login consoles of your PS3, you can turn off the ones you aren't using. Unlike other runlevel-related features, these are actually encoded directly in /etc/inittab. The default configuration for Fedora is to ship with six consoles enabled; I think the screen's more useful. Here's what you'd change in /etc/inittab to turn off five of them:

**Listing 3. One console is enough, thank you**

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:off:/sbin/mingetty tty2
3:2345:off:/sbin/mingetty tty3
4:2345:off:/sbin/mingetty tty4
5:2345:off:/sbin/mingetty tty5
6:2345:off:/sbin/mingetty tty6
```

Changes to the runlevel files are recognized when you change to a new runlevel. What about changes to inittab? To let init know that you've changed inittab, run `/sbin/init q`. The q argument, rather than indicating a runlevel, tells init to reload its configuration files. (I think maybe it stands for query, but it's not documented.)

### Shell size

While bash has many strengths—a feature set that's too large to explain, compatibility with lots of things, multiple different run modes for compatibility with different standards, and more—it does not run in a small amount of memory.

One alternative, useful for running scripts (although not ideal as an interactive shell) is the stripped down dash, a variant of Kenneth Almquist's shell clone (called ash) that was developed by the Debian maintainers. The goal of dash is to have a small and fast shell that can run scripts. It doesn't have every extra feature of bash, but it runs nearly all portable shell scripts, and does so in a small amount of memory, quickly.

On my test system, a typical bash invocation was running around 1.7MB of storage by the time it printed the first prompt; dash ran around 560KB. This can be especially useful if you can arrange for scripts or makefiles to use dash instead of bash; unfortunately, some programs will experience compatibility difficulties if /bin/sh isn't really bash. Still, if you need extra space, that's a place to go looking.

If you don't need regular background jobs, you can shut down anacron and crond; if you don't mind hard-coding your network settings, you can shut down dhclient. A particularly promising avenue might be to replace bash with dash (see Shell size). Still, you're pretty much at the margins here; past this, the only way to reclaim memory is to change the underlying structure of the ps3 framebuffer, or the hypervisor, and that's not practical.

On the other hand, starting with a system that had already begun to swap before you got a prompt, and ending with a system that has over 100MB free storage with a couple of shells, a top process, and sshd running, is no small feat. While it's easy to look down on this from the lofty heights of laptops with 2GB or more of memory, the fact is that an awful lot of software development becomes quite feasible when your development platform has 100MB of free memory; that's enough that many builds will be able to stay entirely within the buffer cache, saving you a great deal of time, to say nothing of being able to mostly avoid swapping except under extreme circumstances.

## Conclusion

With these tweaks, the PS3 becomes a viable, and even sort of roomy, development environment. (It's one of the most accessible environments for someone looking to dabble with the Cell Broadband Engine.) It stays responsive during compiles, and compiles run noticeably faster than they used to. While the memory provided isn't so great for a desktop system, with video players and Web browsers and e-mail clients all running at once, it's certainly adequate for a development effort, and indeed, with the overhead of GNOME and KDE out of the way, it can even manage some desktop work. It's quite possible that future updates to the PS3 firmware will improve things, as might new kernel versions or drivers. Just remember to stay focused on what you need out of the system, and go ahead and disable features you aren't using.

# Resources

### Learn

- See all parts in the *Linux development on the PlayStation 3* series.
- Twm is your good friend if you're in a tight spot, memory-wise.
- Embedded systems often use BusyBox to trim space requirements. BusyBox is more useful in terms of disk space than memory, but it helps with everything.
- Learn more than you ever needed to know about cron and anacron, and bash and dash, and these other fine Unix utilities from Wikipedia. For dhclient, see man dhclient.
- Learn more about ssh.
- The Cell Broadband Engine Resource Center is the definitive resource for all things Cell/B.E.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

### Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

### Discuss

- Get involved in the developerWorks community through blogs, forums, podcasts, and community topics in our new developerWorks spaces.

# About the author

Peter Seebach
Peter Seebach has been collecting video game consoles for years, but has only been running Linux on them recently. He is still not sure whether this is a Linux machine which plays video games, or a game machine which runs Linux.

# Trademarks

DB2, Lotus, Rational, Tivoli, and WebSphere are trademarks of IBM Corporation in the United States, other countries, or both.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.