Advanced search

Linux Journal Issue #110/June 2003

Archive Index

Advanced search

Game Programming with the Simple DirectMedia Layer

Bob Pendleton

Issue #110, June 2003

Put the library behind Tux Racer and the Linux version of Civilization into your game.

Simple DirectMedia Layer (SDL, www.libsdl.org) is a simple, yet powerful, cross-platform game and multimedia development library. The library was developed by Sam Latinga while he was working for Loki Software, Inc. and was used in their commercial game projects. SDL was developed to meet the needs of game developers working in a multi-OS environment and was used in the Linux versions of *Maelstrom*, *Hopkins FBI*, *Civilization: Call to Power*, *Descent 2*, *MythII: Soulblighter*, *Railroad Tycoon II*, *Tux Racer* and many more. The SDL web site lists hundreds of games and applications written using SDL.

SDL officially supports Linux, Windows, BeOS, Mac OS, Mac OS X, FreeBSD, OpenBSD, BSD/OS, Solaris and IRIX. SDL also works with Windows CE, AmigaOS, Atari, QNX, NetBSD, AIX, Tru64 UNIX and SymbianOS. However, those OSes are not yet officially supported. This means if you write your application using SDL, you can port it with minimal rework to all those OSes. SDL provides a portable way to write games and multimedia applications on every major OS currently in use.

Installing SDL

If you are using a recent version of Linux, you probably have a complete SDL installation. In fact, a quick check of /usr/bin using **ldd** on my Red Hat 8.0 system found eight programs that depend on SDL.

The following commands show whether the SDL libraries and C/C++ include files are installed on your system:

```
locate SDL.h
locate libSDL
locate sdl-config
```

If all of these commands report the file was found, most likely you have a complete SDL installation, and you need only to make sure it is up to date. The sdl-config program checks the SDL version and acquires compile and link flags for your SDL applications. If sdl-config was found, run:

```
sdl-config --version
```

to see which version of SDL you have. If sdl-config reports a version less than 1.2.4, you should install newer libraries. Like most open-source projects, SDL is under constant development, so if you are using SDL for development, check for new versions regularly or join one of the SDL mailing lists to keep track of library updates.

If SDL is not installed, you need to download and install it. Your distribution probably has precompiled SDL packages, so you can check your regular source of packages first. If it's up to date, the easiest way to get started is to install the devel or dev packages for SDL from your distribution.

The file sdl-install.sh included with the source code used in this article is a shell script that downloads and installs version 1.2.5 of SDL and all its add-on libraries. The script must be run as root in the directory where you want the source for SDL. The script downloads the following:

- SDL—the core of SDL (www.libsdl.org/download.php)
- SDL_net—the network I/O library (www.libsdl.org/projects/SDL_net)
- SDL_image—the image reading library (www.libsdl.org/projects/SDL_image)
- SDL_mixer—the sound file loading and mixing library (www.libsdl.org/projects/SDL_mixer)
- SDL_ttf—the TrueType font library (www.libsdl.org/projects/SDL_ttf)

If you don't use sdl-install.sh, visit the web pages listed above, download the files, unpack them and follow the instructions in the appropriate README files to install the libraries. Test your new installation by running:

```
sdl-config --version
```

If it doesn't run or gives a version number lower than the version you installed, the installation didn't work. In my experience, this happens when I don't follow the instructions or leave an old version of SDL installed in a different place. If **locate sdl-config** lists more than one location, either delete the old SDL installation, something I hate to do, or re-install over the old version. The sdl-install.sh file shows how to use **./configure --prefix** to install SDL anywhere you want, but it's safest and easiest to install in the default location.

SDL documentation can be found at www.libsdl.org/docs.php. On-line documents are at sdldoc.csn.ul.ie. Support library documentation is either linked from their download pages, included with the source code or embedded in the .h files. Sample programs are included with SDL, and its support libraries are great starting places for your own projects.

SDL Example

The file bounce.cpp [available at ftp.linuxjournal.com/pub/lj/listings/issue110/6410.tgz] is a game written using SDL for input and graphics and SDL_ttf to load TrueType fonts. The game itself is a little over 1,300 lines of C++, and the complete package includes the source code, images, a TrueType font, a makefile, sdl-install.sh and the license files for the font and images used in the game. Finding fonts, graphics and sounds that you can use legally in your games can be more work than writing the game.

To get started learning SDL, download the *Bounce* source code from the *Linux Journal* FTP site and unpack it with **tar -xzvf bounce.tar.gz**. Then run **make** to build the program. Run the program by typing **bounce** at the command line. You can run it in full-screen mode by typing **bounce -fullscreen**. The plot of the game is that Earth has started wandering around the solar system and is in danger of falling into the Sun. Your job is to keep Earth out of the Sun by hitting it with the Moon. You score a point each time you hit the Earth with the Moon, and the game scores every time the Earth hits the Sun. The game is designed to show off features of SDL, not to be the most interesting game you've ever seen.

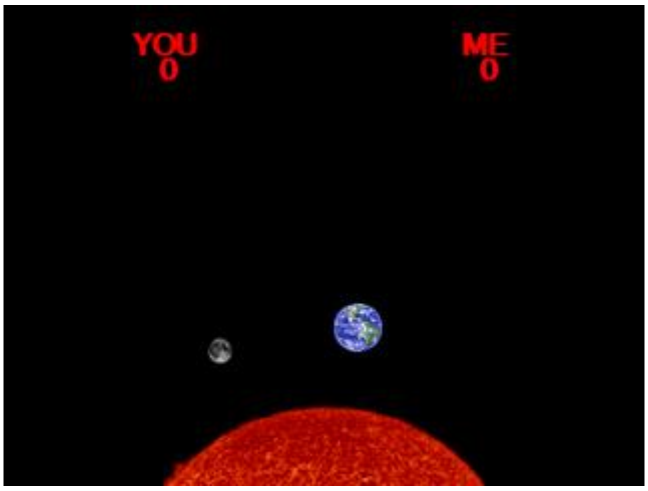


Figure 1. The Game *Bounce*

Initialize SDL

SDL must be initialized before any SDL functions are used by calling SDL_Init():

```
if (-1 == SDL_Init((SDL_INIT_VIDEO |
                    SDL_INIT_TIMER |
                    SDL_INIT_EVENTTHREAD)))
```

```
  {
    ...
  }
```

The parameter to SDL_Init() identifies the subsystems that need to be initialized. Here, I tell SDL to initialize the video, timer and subsystems and to use thread-based event processing. I also could have used the catch-all SDL_INIT_EVERYTHING, but you should initialize only the parts of SDL that your program uses. There is no reason to initialize the joystick or CD-ROM if you are not going to use them. You can initialize and shut down subsystems at any time by the use of the SDL_InitSubSystem() and SDL_QuitSubSystem() functions.

It is important to shut down SDL with a call to SDL_Quit() before your program shuts down. SDL_Quit() shuts down all SDL subsystems, frees all system resources used by SDL and restores the video mode. It is good practice to use atexit() to make sure that SDL_Quit() runs when your program terminates. Failure to call SDL_Quit() can leave your computer in a strange video mode.

Set the Video Mode

When selecting a video mode, decide whether to run in a window or as a full-screen application. Then, choose the size of the window or screen. If you go with a window, decide whether the user can resize it. Then, choose how to adapt to the color depth of the screen. In *Bounce* I use something like:

```
  options = SDL_ANYFORMAT |  SDL_FULLSCREEN;
  screen = SDL_SetVideoMode(640, 480,
                            0,
                            options);
```

The first two parameters specify the width and height, in pixels, of the screen or window in which the program runs. To use a particular width and height in full-screen mode, the screen section of your XF86Config-4 (or XF86Config for some versions of X) file must list the specified size. If *Bounce* won't run in full-screen mode on your machine, it is most likely because you don't have a 640 × 480 mode set up in the screen section of your XF86Config-4 file.

The third parameter specifies the number of bits per pixel. Or, if it is set to 0 (zero), it tells SDL to use the current display depth. It is best to adapt the game to the current display depth rather than counting on every machine on which the code will ever run to support your desired pixel format.

The last parameter lets you give SDL detailed instructions on how to set up the video mode. There are nearly a dozen options from which to chose. In *Bounce*, I use SDL_ANYFORMAT to let SDL pick the best available mode. This option forces your code to adapt to whatever pixel depth you have, but using it can provide better performance at the cost of some extra coding. The SDL_FULLSCREEN option tells SDL to set a full-screen mode.

The value returned by SDL_SetVideoMode() is a pointer to an SDL_Surface structure. This structure describes the screen in great detail. If the pointer is NULL, the video mode you requested is not available. But, getting a non-NULL value doesn't mean you got everything you wanted. Check the flags field of this structure against the options you specified. I have found it is best to ask for little and work with what I receive, that way I avoid hard wiring my machine and OS restrictions in my code.

Now that the video mode is configured, use SDL_WM_SetCaption() to set the window title and icon name. This isn't necessary; it's one of those touches that make the program a little easier to use:

```
SDL_WM_SetCaption("Bounce", "Bounce")
```

Loading Resources

Before *Bounce* can start up, it must load and initialize the resources it uses. *Bounce* has to initialize colors, load a few images and load the font it uses to draw text. Because the video mode was set using SDL_ANYFORMAT, all of these resources have to be converted to match an arbitrary display format. The following code creates a red pixel in the format we need:

```
SDL_PixelFormat *pf = screen->format;
int red = SDL_MapRGB(pf, 0xff, 0x00, 0x00);
```

The SDL_PixelFormat structure is a description of the screen pixels, and SDL_MapRGB() converts a standard 24-bit RGB color representation into a pixel value that shows that color when drawn on that screen.

Loading images is slightly more complex:

```
SDL_Surface *s0, *s1;
s0 = SDL_LoadBMP(name);
s1 = SDL_DisplayFormat(s0);
SDL_SetColorKey(s1,
                (SDL_SRCCOLORKEY |
                 SDL_RLEACCEL),
                black);
SDL_FreeSurface(s0);
```

Core SDL includes SDL_LoadBMP(), which loads a .bmp format image as an SDL_Surface. SDL_image provides routines for loading many other image formats. The image is in the format in which it was created. We convert it to the display format using SDL_DisplayFormat(). SDL_SetColorKey() is used to tell SDL that when it copies (blits) this surface into another surface, it should ignore all the black pixels. I do this so that when I copy an image of the Earth onto the screen, none of the black background gets copied, and only the pixels inside the round shape of the Earth are touched. The SDL_RLEACCEL flag tells SDL to run length encode (RLE) the image. Using RLE-encoded images speeds up image copying.

Bounce uses one TrueType font but in three different sizes, two different colors and three different styles. Using the SDL_ttf library, I wrote a routine that loads a TrueType font, renders each of the ASCII characters in the range of 0-127 as an SDL_Surface, converts each character to match the screen and saves the height, width and advance of each letter so I can draw strings on the screen.

The Main Loop

SDL provides an event-based input system, much like that used by X, Mac OS and Windows. When a key is pressed or the mouse is moved, an event is placed in a queue. The program can either wait for events using SDL_WaitEvent() or poll for events using SDL_PollEvent(). The main loop must process events, update the game state, draw the next frame and repeat until done.

The decision to wait or poll for events affects the overall structure of the game. I chose to wait for events and use a heartbeat timer to drive the action. I like this combination because it lets the program handle events whenever they occur while controlling CPU usage. Both of those qualities are important in networked games.

The timer is initialized using:

```
   timer = SDL_AddTimer(10, timerCallback, NULL);
```

This tells SDL to call a routine named timerCallBack every ten milliseconds. My timer callback uses SDL_PushEvent() to send an event. Because timer callbacks run in a separate thread, they can send events even though the game is stopped, waiting for events. When it receives a timer event, *Bounce* checks to see if it is time to draw another frame. The timer makes sure the program doesn't try to draw more than 100 frames/second, while allowing the game to run at a slower rate if it must. On my machine, it runs at 85 frames/second, which matches the refresh rate of my monitor.

Bounce is organized into several different pages. The main loop handles events that are common among all the pages, such as quitting the program when you press Esc or pausing the game when you press F1. After the main loop has looked at an event, it passes the event to the current page. Each page is a function that takes an SDL_Event as its parameter. Each page has the responsibility to handle events, keep track of the time and draw the screen. Although this approach leads to some duplicate code, it gives the programmer greater flexibility, and it lends itself to an object-oriented design where each page is an instance of a page class. The following example shows parts of the main loop and illustrates how events are passed to the individual pages:

```
while ((!done) && SDL_WaitEvent(&event))
{
  switch (event.type)
  {
  case SDL_QUIT:
    done = true;
    break;
  case SDL_KEYDOWN:
    switch(event.key.keysym.sym)
    {
    case SDLK_ESCAPE:
      done = true;
      break;
    case SDLK_F1:
      play = !play;
      break;
    }
    break;
  }
  if (play &&
      (!done) &&
      (NULL != currentPage))
  {
    currentPage(&event);
  }
}
```

The global variable currentPage points to the implementation of the current page. When one page wants to start another page, it initializes the new page and sets the pointer to that page. *Bounce* has three pages: the welcome page you see when the program starts, another page handles game play, and the "You Won/You Lost" message is the third page.

The event handler in the welcome page looks like:

```
switch (e->type)
{
  case SDL_USEREVENT:
    switch (e->user.code)
    {
    case MY_TIMEREVENT:
      now = SDL_GetTicks();
      dt = now - lastTime;
      if (dt >= minFrameTime)
      {
        drawWelcome(dt);
        lastTime = now;
      }
      break;
    }
    break;
case SDL_MOUSEBUTTONDOWN:
  initBounce();
  currentPage = bounce;
  break;
}
```

When this code sees a timer event, it checks how long it has been since it last updated the screen and calls drawWelcome() to animate the screen. When it sees that a mouse button has been pressed, it switches to the game page by calling initBounce() to get it ready and then sets currentPage to point to the game page. The next time through, the main loop bounce() will be called.

Animation

The animation routine uses the dirty pixels technique so only a small portion of the screen is redrawn for each frame. With this technique, we keep track of the last position in which an object was drawn and the new position. When *Bounce* draws the Earth, first it erases the dirty pixels where it was by filling them with the background color, and then it draws the Earth in its new location. We fill rectangles and draw images using:

```
   SDL_FillRect(screen, rectangle, color);
   SDL_BlitSurface(image, NULL, screen, rectangle);
```

SDL_FillRect() fills a rectangle in an SDL_Surface, like the screen, with a color. The rectangle is specified using an SDL_Rect structure, and the color is created using SDL_MapRGB(). SDL_BlitSurface() copies a rectangle from one surface into a rectangle in another surface. If the source rectangle is NULL then the whole surface is copied. SDL_BlitSurface() is the routine that applies the color key and takes advantage of RLE encoding.

Summary

SDL reduces the time it takes to write games on Linux. It is small enough that learning it is a project, not a career, and it is powerful enough for commercial applications. I hope that between the information in this article and the source code for *Bounce*, you have learned enough of SDL to start modifying *Bounce* and building your own SDL games.

Resources

email: bob@pendleton.com

Bob Pendleton's first programming assignment was to port games from an HP minicomputer to a UNIVAC mainframe, and he has been fascinated by computer games ever since. He has been working with various versions of UNIX and Linux since 1981. He is an independent software developer and writer. You can reach him at Bob@Pendleton.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Embedding an SQL Database with SQLite

Michael Owens

Issue #110, June 2003

If you want the convenience of SQL without the size and setup hassles of a database server, embed SQLite right in your program, whatever your favorite language.

SQLite is a powerful, embedded relational database management system in a compact C library, developed by D. Richard Hipp. It offers support for a large subset of SQL92, multiple tables and indexes, transactions, views, triggers and a vast array of client interfaces and drivers. The library is self-contained and implemented in less than 25,000 lines of ANSI C, which is free to use for any purpose. It is fast, efficient and scalable, and it runs on a wide variety of platforms and hardware architectures ranging from ARM/Linux to SPARC/Solaris. Furthermore, its database format is binary-compatible between machines with different byte orders and scales up to 2 terabytes (241 bytes) in size.

Hipp conceived of the idea of SQLite while working with a team from General Dynamics on a program for the US Navy for use onboard the DDG class of destroyers. The program ran on HP-UX and used an Informix database. As they began the project, they quickly found that Informix can be rather difficult to work with. It runs fine once you get it going, but it can take a full day for an experienced DBA to install or upgrade.

At the time, the team was using Linux and PostgreSQL for development work. PostgreSQL required considerably less administration, but they still wanted to be able to produce a standalone program that would run anywhere, regardless of what other software was installed on the host platform. In January 2000, Hipp and a colleague discussed the idea of writing a simple embedded SQL database engine that would use GDBM as its back end, one that would require no installation or administrative support whatsoever. Later, during a funding hiatus, Hipp started writing it on his own, and SQLite version 1.0 soon came to life.

General Dynamics started using SQLite in place of PostgreSQL right away. SQLite allowed them to generate a standalone executable that could be installed quickly and easily on wearable computers and laptops for display at tradeshows and sales meetings. Informix still is being used for shipboard operation; however, the Naval office in charge of ongoing maintenance of the program recently has been e-mailing Hipp for help in compiling SQLite on an HP 9000. So, things may be changing.

Major changes came about with version 2.0. Version 1 used GDBM for storage, which uses unordered keys (aka hashing). This limited what SQLite could do. Furthermore, GDBM is released under the GPL, which discouraged some from trying it. In January 2001, Hipp began working on his own B*Tree-based back end to replace GDBM. The new B*Tree subsystem stores records in key order, which permits optimizations such as logarithmic time MIN() and MAX() functions and indexed queries with inequality constraints. It also supports transactions. The end result was a much more capable database. Version 2.0 was released into the public domain in September 2001.

SQLite started to take off with version 2.0. Dozens of people began writing in to tell how they were using it in commercial and free products. A few even contracted Hipp for technical support or custom modifications. According to Hipp, at least one widely used program for Windows incorporates a modified version of SQLite in recent releases. SQLite also is being used at Duke Energy and in other branches of the US military, besides the Navy project that originally inspired it.

Since its public release a year and a half ago, SQLite has been gaining features and users at a speedy clip. A quick look at the SQLite Wiki reveals many more applications whose developers have discovered SQLite and put it to use in their software. It even has its own Apache module (mod_auth_sqlite), which seems to be a sign of success in its own right. As the creators of PySQLite, a Python extension for SQLite, Gerhard Häring and I have been surprised to see more than 3,000 downloads in less than a year. Currently, SQLite is the highest-rated database engine on <@url>freshmeat.net.

Architecture

SQLite has an elegant, modular design. It can be divided into eight primary subsystems (Figure 1), some of which take rather interesting approaches to relational database management.

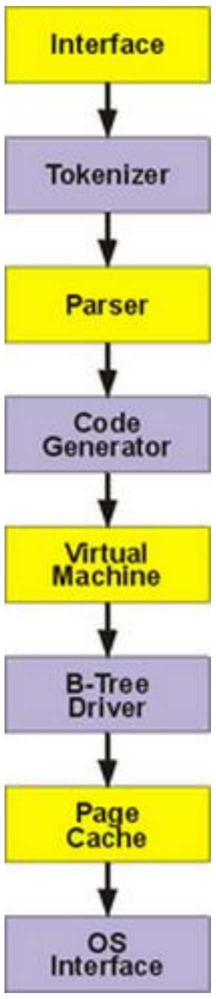


Figure 1. SQLite Architecture

At the top of the diagram is the parser and tokenizer. SQLite includes its own highly optimized parser generator, called the Lemon parser, which produces fast, efficient code, and by virtue of its novel design, it is especially resistant to memory leaks. At the bottom is an optimized B-Tree implementation, based on Knuth, which runs on top of an adjustable page cache, helping to minimize disk seeks. The page cache, in turn, operates on an OS abstraction layer that helps make the library more portable.

At the library's core is the virtual database engine. The VDBE performs all operations related to data manipulation and is the broker through which information is passed between client and storage. In many ways, it is the heart of SQLite. The VDBE comes into play after the SQL is parsed. The code generator takes the parse tree and translates it into a mini-program, which is made up of a series of instructions expressed in the VDBE's virtual machine language. One by one, the VDBE executes each instruction, which ends by fulfilling whatever request was specified in the SQL statement.

The VDBE's machine language consists of 128 opcodes, all centered around database management. There are opcodes for opening tables, searching indexes, storing and deleting records and many other database operations.

Each instruction in the VDBE consists of an opcode and up to three operands. Some instructions use all three operands; others use none. It all depends on the nature of the instruction. For example, the Open instruction, which opens a cursor on a table, uses all three operands. The first operand (P1) contains the ID by which the cursor will be identified. The second operand (P2) refers to the location of the root (or first) page of the table, and the third operand is the table's name. The Rollback instruction, on the other hand, requires no operands at all. The only thing the VDBE needs to know in order to perform a rollback is whether or not to do one.

For any given SQL statement, you can view the generated VDBE program using the explain command in the SQLite shell. Listing 1 shows a simple example.

<u>Listing 1. **explain** Results for a Simple Query</u>

explain is not only useful for gaining better insight into the workings of the VDBE but also for practical matters like query optimization. The VDBE is really a subject in itself. Fortunately, for those who are interested, it is well documented, and its theory of operation along with its opcodes are covered in great detail on the SQLite web site.

With respect to physical storage, each database is stored in a single file. That is, all database objects that comprise an individual database (views, triggers, indexes, tables, schema and so on) reside together in one file that defines an SQLite database. Database files are made up of uniformly sized pages. Page size is set upon database creation, and valid sizes range from 512b-4Gb. By default, SQLite uses a 1Kb page size, which seems to offer the best overall performance.

Transactions are implemented using a second file called the journal, which only exists when there is one or more active connections to the database. Each database has exactly one journal file. It holds the original (unmodified) pages that were changed in the course of a transaction. When the transaction commits, the journal pages are no longer needed and are summarily discarded. Rollbacks are performed by restoring pages from the journal file to the database file. The use of the journal file ensures that the database always can survive a crash and be restored to a consistent state. The first client to connect to a database after a crash triggers a rollback of the previous transaction. Specifically, when the client connects, SQLite tries to create a new journal file, only to find that a previous one exists. When this happens, it infers a crash must have occurred and proceeds to copy the contents of the old journal file back into the database, effectively restoring it to its original state before the crash. Then it gives the client the go-ahead to start working.

Simple API, Many Languages

SQLite has an extremely easy-to-use API that requires only three functions with which to execute SQL and retrieve data. It is extensible, allowing the programmer to define custom functions and aggregates in the form of C callbacks. The C API is the foundation for the scripting interfaces, one of which (the Tcl interface) comes included in the distribution. The Open Source community has developed a large number of other client interfaces, adapters and drivers that make it possible to use SQLite in other languages and libraries.

Using the C API requires only three steps. Basically, you call sqlite_open() to connect to a database, in which you provide the filename and access mode. Then, you implement a callback function, which SQLite calls for each record it retrieves from the database. Next, call sqlite_exec(), providing a string containing the SQL you want to execute and a pointer to your callback function. Besides checking for error codes, that's it. A basic example is illustrated in Listing 2.

Listing 2. Basic C API Example

One of the nice things about this model that differs from other database client libraries is the callback function. Unlike the other client APIs where you wait for the result set, SQLite places you right in the middle of the result-gathering process, in the thick of things as they happen. Therefore, you play a more active role in fetching data and directly influence the retrieval process. You can aggregate data as you collect it or abort record retrieval if you want. The point is, because the database is embedded, your application is essentially as much the server as it is the client, and SQLite takes full advantage of this through the use of its callback interface.

In addition to the standard C API, an extended API makes it even easier to fetch records, using sqlite_get_table(), which does not require a callback function. This function behaves more like traditional client libraries, taking SQL and returning a rowset. Some of the features of the extended API are functions to extend SQL by adding your own functions and aggregates, which is addressed later in this article.

Finally, if for some reason you need an ODBC interface, I am pleased to inform you that one is available, written by Christian Werner. His ODBC driver can be found at www.ch-werner.de/sqliteodbc.

Auto-increment Columns

While SQLite does not support sequences per se, it does have an auto-increment key and the equivalent of MySQL is mysql_insert_id(). A primary key can be set to auto-increment by declaring it INTEGER PRIMARY KEY. The value of the last inserted record for that field is obtained by calling sqlite_last_insert_rowid().

BLOBs

You can store binary data in SQLite columns with the restriction that it only stores up to the first NULL character. In order to store binary data, you must first encode it. One possibility is URL-style encoding; another is base64. If you have no particular preference, SQLite makes life easy for you through two utility functions: sqlite_encode_binary() and sqlite_decode_binary().

Thread Safety

SQLite is as threadsafe as you are. The answer more or less centers around the SQLite connection handle returned by sqlite_open(). This is what should not be shared between execution contexts; each thread should get its own. If you still want threads to share it, protect it with a mutex. Likewise, connection handles should not be shared across UNIX fork() calls. This is more common sense than anything else. Bottom line: thread or process, get your own connection handle, and everything should be fine.

SQLite uses the concept of a pragma to control runtime behavior. Pragmas are parameters that are set using SQL syntax. There are pragmas for performance tuning, such as setting the cache size and whether to use synchronous writes. There are some for debugging, like tracing the parser and the VDBE, and others still are for controlling the amount of information passed to client callback functions. Some pragmas have options to control their scope, having one variant that lasts only as long the current session and another that takes effect permanently.

SQLite sorts a column lexigraphically if, and only if, that column is declared as type BLOB, CHAR, CLOB or TEXT. Otherwise, it sorts numerically. SQLite used to make decisions on how to sort a column solely by its value. If it “looked” like a number, then it was sorted numerically, otherwise lexigraphically. A tremendous amount of discussion about this appeared on the mailing list, and it eventually was refined to the rules it uses today, which allow you to control the method of comparison by the declared type in the schema.

Scripting Interfaces

As mentioned earlier, many client interfaces have been developed for SQLite. To give you a taste, a Python version of the previous C example is illustrated in Listing 3, and its Perl counterpart is shown in Listing 4. It doesn't get any easier. SQLite also can be used from the shell, which makes it amenable to system administration tasks. A shell version of our stock example is provided in Listing 5.

Listing 3. Python Example

Listing 4. Perl Example

Listing 5. Shell Example

Finally, because I am not a Java, Tcl, Ruby, Delphi, Lua, Objective C, PHP, Visual Basic, .NET, Mono, DBExpress, wxWindows, Euphoria or REXX programmer, I will have to refer the likes of you who are, to the SQLite Wiki to find your respective interfaces. See cvs.hwaci.com:2080/sqlite/wiki?p=SqliteWrappers for your preferred way to talk to SQLite.

Extending SQLite

SQLite includes a nice C framework in which to create your own functions and aggregates that can be called from SQL. Some wrappers, such as the Python wrapper, allow you to use this feature to implement them in the extension's language. SQL, such as **INSERT INTO orders purchase_date values CURRENT_TIME()**, is a simple matter of writing a callback function that looks something like Listing 6. Then, register the function and use it as shown in Listing 7.

Listing 6. Implementation of CURRENT_TIME()

Listing 7. Using CURRENT_TIME()

All of SQLite's built-in functions, such as avg(), min(), max() and sum(), with the exception of the magical typeof(), are implemented using this API. User-defined aggregates can be added just as easily. Doing something like **SELECT variance(age) from population** uses a very similar approach to creating functions. This, however, is left as an exercise for the reader. Hint: the file func.c includes some excellent examples. Like functions, SQLite uses the API to implement its aggregates as well.

Administration

For administration, SQLite offers an intuitive utility program conveniently named sqlite with which users of MySQL and PostgreSQL will feel perfectly at home. It has both shell and command-line modes. Within the shell, you can view a database's tables, schema and indexes, as well as execute SQL on the command line and in external files. It also has some nice modes for viewing data and VDBE output.

Though loading and unloading data can be done within the shell, it is even easier on the command line. Given a file containing valid DDL/DML (call it dump.sql), you can load it into a database (call it db), like so:

```
sqlite db < dump.sql
```

This creates the database db if it doesn't exit. The reverse process to dump a database would be:

```
sqlite db .dump > dump.sql
```

SQLite is powerful. Its wide application, ease of use, portability, speed, scalability, small footprint and clean code base make it a library that all programmers should have in their arsenals. And given its license, there is simply no reason not to. The SQLite Project is always looking for new users and developers, and it welcomes new ideas and engaging discussion. I hope you enjoy learning about and using it as much as I have.

Resources

Michael Owens (mike@mikesclutter.com) is a chemical engineer turned programmer. He works for a real estate firm in Dallas/Fort Worth, Texas using Linux and open source to develop in-house software. He is the creator and codeveloper of PySQLite.

A Template-Based Approach to XML Parsing in C++

John Dubchak

Issue #110, June 2003

Using the Xerces library and a little C++ code, you can parse an XML file to get only the information you need as easy-to-handle objects.

XML is a markup-based data description language designed to allow developers to create structured documents using descriptive custom tags. The intent of XML is to separate the description of the data from its intended use and allow the transfer of the data between different applications in a nonplatform- or architecture-specific way. Another useful application of XML is to describe a process in a logical and meaningful manner that can be carried out by the application at runtime.

Parsing XML

In order for an XML file to be parsed successfully, the developer must first create a file that can be processed by a parser. A parser is a set of shared objects or a library that reads and processes an XML file.

The parser may be one of two types: validating or nonvalidating. A validating parser scans the XML file and determines if the document is well formed, as specified, by either an XML schema or the document type definition (DTD). A nonvalidating parser simply reads the file and ignores the format and layout as specified by either the XML schema or the DTD.

The most widely used parsers represent two different approaches: event-driven and tree-based. The event-driven parser is called SAX (simple API for XML processing). A tree-based parser creates a DOM (document object model) tree in memory at the time the XML file is read and parsed.

The DOM implementation is difficult to navigate and does not allow for clean mapping between XML elements and domain-specific objects. SAX provides the events to allow developers to create their domain-specific objects at the time the XML file is read and parsed. This article delivers a framework design using the SAX API for XML parsing.

XML Parsers for C++

The two most commonly used parsers for C++ are the open-source Xerces of the Apache Project and XML4C created by IBM's alphaWorks Project. XML4C is based on Xerces.

Both parsers essentially provide the same layout of source and libraries and therefore can be used interchangeably. They also support both DOM- and SAX-based XML parsing.

This document describes an implementation using the SAX parser with the Xerces parser. The Xerces source or binaries related to XML parsing can be downloaded from the Xerces web site (see Resources).

Parsing XML Files Using SAX

In order to begin parsing an XML file using the SAX API, the layout of the SAX C++ object interactions must be understood. SAX is designed with two basic interfaces:

SAXParser

setDoValidationsetDoNamespacesetDoSchemasetValidationFullSchemaCheckingsetDocumen

and

HandlerBase

warningerrorfatalErrorstartElementcharactersignorableWhitespaceendElement

Close examination of the methods in the HandlerBase object reveals two different categories of methods: error handling and document processing. The error-handling methods include warning, error and fatalError, whereas the parsing methods consist of startElement, characters, ignorableWhitespace and endElement. These behaviors can be separated into individual objects, as shown later.

The SAXParser class takes care of setting basic properties and the desired behavior to be enforced at runtime.

The following sample code illustrates the basic steps for parsing an XML file using the SAX parser in C++:

```
// Create a new instance of the SAX parser
SAXParser parser;
// Initialize the behavior you desire
parser.setDoValidation(true);
parser.setDoNamespaces(true);
parser.setDoSchema(true);
parser.setValidationSchemaFullChecking(true);
// Add handlers for document and error processing
parser.setDocumentHandler(&docHandler);
parser.setErrorHandler(&errorHandler);
// Parse file
parser.parse("MyXMLFile.xml");
```

At the time the parsing occurs, the classes you've instantiated, docHandler and errorHandler, are forwarded the events that get triggered from the parsing. These classes are derived from the Xerces base class HandlerBase and have overridden the appropriate methods for handling the events based on their categorized function.

Now that we've been exposed to parsing XML using SAX, let's explore how our XML framework has been implemented to take advantage of the facilities provided within the API.

Policy Classes

A policy class, as described and made popular by Andrei Alexandrescu's *Modern C++ Design* (see Resources), “defines a class interface or a class template interface. The interface consists of one or all of the following: inner type definitions, member functions and member variables.”

The usefulness of policy classes, in this XML framework, are realized when created using a template-based C++ design. A policy allows you to parameterize and configure functionality at a fine granularity. In this design, policies are created to accommodate the following behaviors: document handling, error handling, domain mapping and parsing.

Configuring these elements as policies allows the creation of more concise code that is easier to maintain by any developer experienced in C++ and the use of templates.

The principal class of the XML-parsing framework is the XMLSAXParser. It's a custom-designed class template that implements the XMLParserInterface and includes a SAXParser object as a member variable. The template parameters include policy classes for both the document and error handlers. All parsing is eventually delegated to the SAXParser member variable after the various handlers and other properties have been set.

Implementing custom handlers, as policy classes, is a relatively trivial task using the framework. The advantage of this type of design is that the same framework can be used with different parsing APIs and different domain-mapping objects by altering one or more of the policies—an exercise that is not implemented in this article.

In order to create custom handlers, derive newly created custom classes from HandlerBase and override the virtual methods of interest. The following two types of custom handlers have been created in the XMLFactory framework:

XMLSAXHandler

startElementcharacterignorableWhitespaceendElement

and

XMLSAXErrorHandler

warningerrorfatalError

XMLSAXHandler handles document event processing, and XMLSAXErrorHandler handles the various error callbacks.

Mapping XML Tags to Domain Objects

The next aspect of our XML-parsing framework is converting XML tags into domain-related objects that can be used within the application by using templates and a loose definition of policy classes.

The XMLDomainMap template accepts a single template parameter, called an XMLNode. The interface for the domain-mapping object is as follows:

XMLDomainMap

createaddupdateAttribute

The XMLNode acts as both a leaf and a root in a tree structure that aggregates its children as a subtree. The XMLNode's interface is:

XMLNode

operator==operator!
=operator=addChildhasChildrennumChildrenvaluenamegetChildCountgetChildgetParent

The key here is the design of the public interface of the object. There are several operator overloads, specifically operator equals (operator==), operator not equals (operator!=) and the assignment operator (operator=). The benefit to this is the object now can be used with many standard template library (STL) containers and algorithms, which allows for the use of advanced features with the C++ language.

Linking our Classes Together—An XML Façade

Thus far, the focus has been on individual classes and describing the templates that have been created for our XML-processing framework. The next step is to link the disparate interfaces together and make them appear to function as a single cohesive unit by using the façade design pattern.

The façade design provides a simple and elegant way to delegate parsing functionality from an outside client to the internal policy class that will be used for performing the parsing.

In *Design Patterns*, the authors define the intent as to “Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.”

The XMLProcessor is the façade that has been created. It is defined with the following interface:

XMLProcessor

parsegetParseEngine

Once all the source has been written, an XML file and a test client will be needed to run our sample.

Parsing an Actual XML File

The following simple XML file, showing the basic layout of a customer record with a name and account number, has been created to illustrate the simplicity of using the framework:

```
<?xml version="1.0"
encoding="iso-8859-1"?>
<customer>
    <name>John Doe</name>
    <account-number>555123</account-number>
</customer>
```

For now, create this file with a text editor and save it as MyXMLFile.xml.

The Public Interface—Writing the Client Application

The framework's functionality will be used as a mechanism to provide a succinct interface to the client application.

The primary methods that a client of the framework would make use of can be described with an actual, albeit small, sample of C++ source code:

```
// ---------------------------------------
//  Sample source for parsing an XML doc
// ---------------------------------------
#include "XMLProcessor.hpp"
#include "XMLDomainMap.hpp"
#include "XMLSAXParser.hpp"
#include "XMLNode.hpp"
#include "XMLCommand.h"
#include "XMLSAXHandler.hpp"
#include "XMLSAXErrorHandler.hpp"
#include <iostream>
using namespace std;
using namespace XML;
// Let's get the ugly stuff out of the way first
typedef XMLSAXHandler<XMLDomainMap<XMLNode> >
  DOCHANDLER;
typedef XMLSAXErrorHandler ERRORHANDLER;
typedef XMLSAXParser<DOCHANDLER, ERRORHANDLER>
  PARSER;
typedef XMLProcessor<PARSER> XMLEngine;
// Create a basic test client
int main(void)
{
    // Define a string object with our filename
    std::string xmlFile = "MyXMLFile.xml";
    // Create an instance of our XMLFactory
    XMLEngine parser(xmlFile);
    // Turn off validation
    parser.doValidation(false);

    // Parse our XML file
    parser.parse();
    // Get the tree root
    XMLNode root = parser.getXMLRoot();
    // Print the name of our object
    cout << "Root = " << root.name() <<
endl;
    return 0;
}
```

Now that an instance of an XMLNode object representing the root of the tree has been parsed, the child elements of the root XMLNode can be accessed.

Compiling the Test Client

The last step is to compile the client. Simply perform the compile at the command line:

```
g++ -o testClient -I. -I/path/to/xerces/include \
-I/path/to/xerces/include/xerces testClient.cpp \
-L/path/to/xerces/lib -lxerces-c
```

This compiles the client application. The next step is to run a test. Be sure to set your LD_LIBRARY_PATH environment variable to point to the location of your Xerces installation's lib directory. Because the shared libraries are from this directory, the application loader needs a way to import the required symbols at runtime in order for everything to function correctly.

When testClient is run, the following output is expected:

```
$>testClient
Adding child name
Adding child account-number
Root = customer
```

You now have a fully functional XML-parsing framework using C++ templates that will allow you to incorporate XML into your new or existing applications. Sample code is available at ftp.linuxjournal.com/pub/lj/listings/issue110/6655l1.tgz.

Resources

email: jdubchak@qwest.net

John Dubchak is a senior software developer working as a consultant in the St. Louis area. He's been programming in C++ for the past 12 years and can't believe how bad his first lines of C++ actually were. His wife says that his hobby is “sitting at the computer writing little programs”.

Speeding Up the Scientific Process

Sam Clanton

Issue #110, June 2003

Learn how to optimize your Matlab project by converting parts to C.

As a research staff member at NASA Ames Research Center at Moffett Field, in the heart of Silicon Valley, California, I was a part of a team that used Linux for some interesting and advanced research. I worked in the Neuro Engineering Lab at Ames in support of the construction of a brain-computer interface, a system by which EEG (electroencephalogram—brain wave) signals can be used to control electronic systems and robotic devices. My job was to take ideas and prototype code from primary lab researchers and develop and evaluate efficient implementations of them for use in real-time data processing with human subjects. Often, I would be handed only a rough sketch of an algorithm or a fragment of code to see if it could be used on the brain wave data we had been collecting.

Matlab and the free software GNU Octave were great tools for doing this work; they allowed me to develop effective methods for data processing and data visualization that would have been a real pain to construct in C or, heaven forbid, Fortran. Ease of implementation is a great concern when dealing with large amounts of experimental code that may or may not end up as a finished product.

When a process did indeed fit the bill, and it was time to start thinking about using it in our real-time data processing system, it would become immediately apparent that the advantages of Matlab in programming ease did not come without cost. The cost was speed. The time to process data representing a single second, for instance, could take minutes or even hours. Obviously, this would not do in a real-time system. Also, any code deemed worthy would have to end up in C or C++ to fit into our existing code base. To address both of these issues, we rewrote much of our Matlab code in C.

Now, if you have some experience with Matlab, you might think, “but Matlab already exports to C on its own” or, “what about the new Matlab JIT compiler?” Although the new JIT compiler may speed up code in places (looking at the documentation, there are many exceptions to what it will try to optimize), it cannot equal the efficiency of well-written, compiled C code. As for the C export feature of Matlab, the code exported by Matlab is as slow as the interpreted code running inside the Matlab environment and is fairly difficult to merge into existing projects without a bit of interface work. And, none of this helps users of GNU Octave or those that can't keep up with expensive Matlab upgrades. In general, it seems the best way to work something originally developed in Matlab into fast, production-level code, is to do it by hand.

This article first offers a few tips on how to write somewhat more efficient Matlab code. Then it illustrates the process of integrating C code into a Matlab program using MEX functions, in order to speed up program execution while still tweaking and evaluating it in the Matlab environment. From there, it is a relatively short step to bring the entire project into C or C++. Most of the information here is available in different places on-line; this article is presented as a sort of a HOWTO or a personal account of bringing a piece of Matlab experimental code into the real world.

For this article, I use as an example a piece of code developed to isolate rapid changes of voltage measured on the surface of the head. The code uses an algorithm called multicomponent event-related potential estimation, or simply mcERP. I first looked into porting Matlab code to C when working on this algorithm. When testing the algorithm with different configuration parameters and input data sets, I usually would have to let it run overnight. No amount of optimization inside Matlab was able to drastically cut down its execution time.

After full conversion to C, it usually would take on the order of tens of seconds to execute with a large input data set. I view this as an extreme time savings, due to the highly nested, looping nature of the algorithm (see Listing 1). I would not expect most algorithms to speed up this much. Even so, this performance still is not quite good enough for real-time operation, but it is close enough that we could start to look at data reduction techniques, parallelizing the code and other tricks to pare it down to something closer to the speed we need.

Optimizing Code in Matlab

The main area in which the performance of Matlab suffers greatly is looping. Matlab abhors the loop; it was written to be more efficient to do many loop-type operations by vectorizing the code, applying functions over a range of data in a matrix, than it is to iterate through the data. Unfortunately, this only works with certain kinds of operations. When dealing with high-dimensional matrices, this often produces code that is hard to read and understand. Looping happens to be an area in which C excels—iterating through a matrix using pointer arithmetic is an extremely efficient and sometimes more understandable way to do operations over large chunks of data. Most of the effort of C optimization of Matlab code is spent trying to optimize nested loop structures.

Other ways to code in Matlab more efficiently include:

1. Make sure to allocate all, even moderately sized, arrays using the zeros() function before assigning values to the array, instead of having Matlab append data to existing arrays as values are assigned.
2. As mentioned in the Matlab documentation, store all of your code in functions instead of scripts. This offers about a two- to threefold speed increase.
3. Organize data such that operations over a range of a matrix operate in a column-major fashion. Matlab stores arrays like Fortran does, in that data in a particular matrix column is contiguous in memory. This is unlike C, where data in a matrix row are contiguous in memory. If you are going to apply functions over a range of data, store that data in a column rather than along a row in the matrix. This is completely anecdotal and may be false, but it seems to make sense.
4. Try to avoid internal-type conversions that happen over and over. This is another one where I don't have hard proof, but as Matlab usually does not make you explicitly label the data types of variables, it is sometimes easy to have a loop of repeated implicit type conversions. It is better to convert to a common data type first, then do your repeated operations. This is like programming in C or C++ but harder to detect right away, because variables are almost never explicitly typed in Matlab.

That being out of the way, let's take a look at a code snippet from the mcERP algorithm (Listing 1). This represents one of the many nested loop structures within the code. The mcERP algorithm relies on a complicated process of iterative Bayesian waveform estimation. A number of the following loopy bits are in the code, all of which are run repeatedly to hone in on waveform shapes present at the data.

<u>Listing 1. Nested Loops in the mcERP Algorithm</u>

One can see how this sort of structure would not run so quickly with an interpreter that does not perform well with loops. However, because of the inner if statement, the code cannot be vectorized without adding an inner function call—which can't be any better. This code, then, is a prime candidate for translation to C/C++. However, it is nice to have a foot in Matlab when developing the algorithm, because it is easy to produce pretty pictures like those in Figure 1. So, we write something called a MEX function. That way, we can have the core fast bits run quickly while retaining interface points around those parts in Matlab that tune and inspect the overall algorithm.

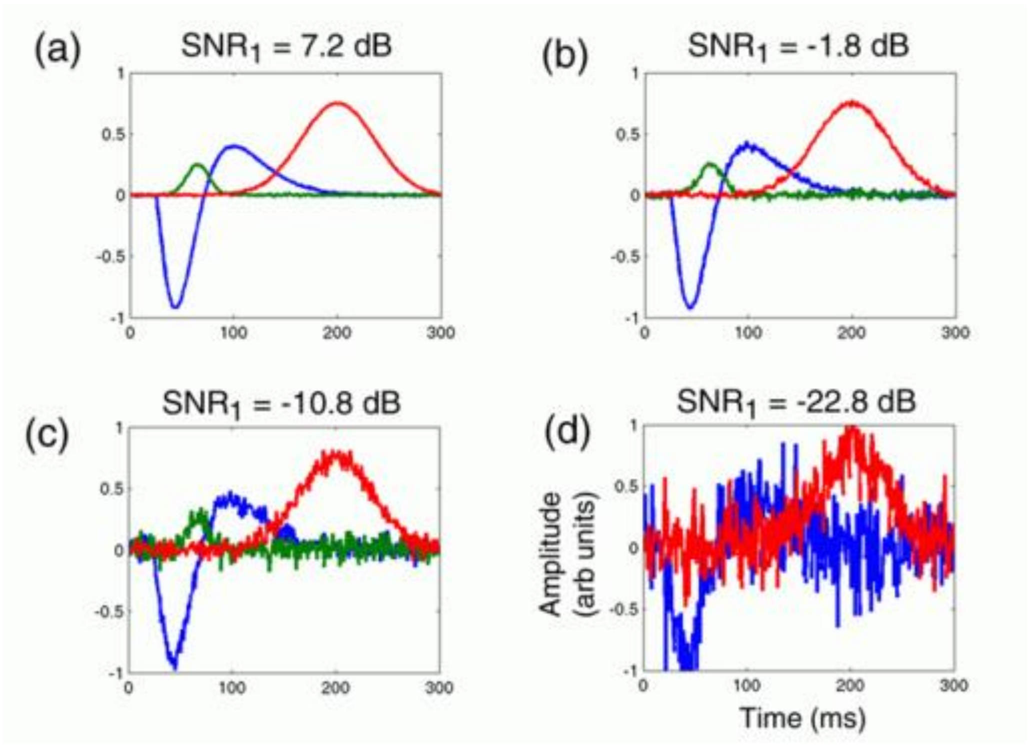


Figure 1.

Figure 1 is an example output from the mcERP algorithm, showing estimates of the fundamental waveforms driving real-time potential readouts at scalp electrodes during simulated experimental trials. Each of these waveshapes is the result of many iterations of progressively accurate Bayesian waveshape estimation, requiring many calculations per iteration. These results can take many hours to achieve with Matlab but take seconds or minutes if portions of the algorithm are rewritten in C.

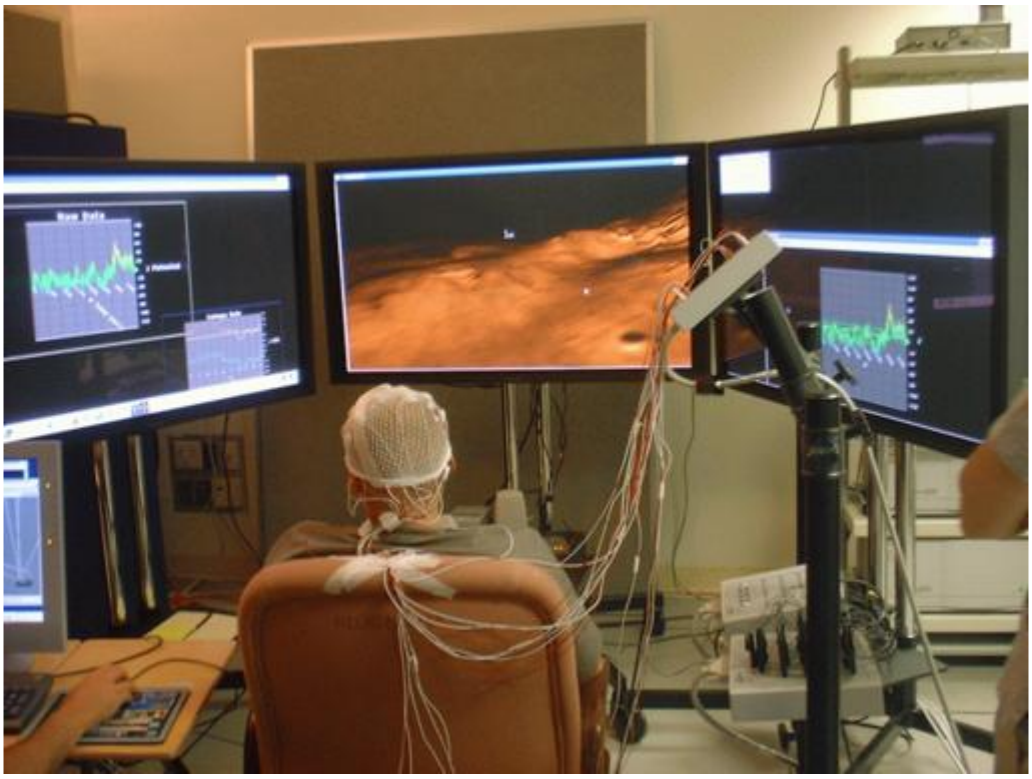


Figure 2.

The photograph in Figure 2 shows our experimental setup for conducting experiments in brain-computer interface with real-time feedback. With the three large displays, we have complete control over what the subject sees within most of his or her field of view. All of the number crunching and display software was developed in-house and runs on Linux.

Setting Up the C Environment for Matlab

A fair amount of documentation is available on the MathWorks web site (see Resources), and you should take a look at it after reading this article if you are serious about developing MEX functions. This article emphasizes optimization and some of the sticking points of getting things working.

To develop MEX functions in Linux, go to the source of all that is good, the command line, and type **mex -start**. When that doesn't work, search for the MEX script within the Matlab install directory. Your system administrator may have made a link in your path only to the Matlab binary. Running MEX, you are presented with a choice of compiler. To take advantage of some compiler optimizations that I will detail later, it is a good idea to use GCC rather than the Matlab built-in LCC. MEX will create the file ~/.matlab/R12/mexopts.sh, which is sourced when you compile external code for Matlab using the MEX utility. It is useful and instructive to take a look at the mexopts.sh file, under the appropriate section for your platform/compiler. In the case of x86 Linux/GCC, look at the glnx86 section of the main switch statement. Any changes made outside this section do not have any effect when compiling code. Place any compiler switches there with which you wish to compile your C functions. For the purpose of optimization, you might want to try:

```
COPTIMFLAGS='-O3 -funroll-loops
-finline-functions'
```

(this is aggressive—be careful) or whatever flags you desire. To use these flags in compilation, you later must run MEX with the -O option. As with some makefiles, include here any header directories you wish to include by appending:

```
-I/path/to/header/directory
```

to the end of CFLAGS. Indicate libraries you wish to link with by adding:

```
-l[libname]
```

and:

```
-L/path/to/library/directory
```

to the end of LDFLAGS.

Once this is done, set up a directory to hold your C files to be compiled with MEX. I suggest not working on C-based and Matlab-based code within the same directory. Now, add that directory, or a third build directory that you have created, to the path within the Matlab environment. Now you are ready to think about writing code.

How to Code in C for Matlab

First and foremost, think about what your goals are for optimization and what parts of your program will benefit the most by being rewritten in C. Since the prime area in which C is much faster than Matlab is the evaluation of loops, it makes sense to look for loops over a lot of data in your code. It is not worth it to code for something that loops three times, but if you are iterating over each voxel in a 500 × 500 × 500 volume, coding in C can shave off tons of time. Especially look for simple operations in nested loops, like the code fragment in Listing 1. Anything that performs a complex operation in a nested loop—anything that looks hard to implement yourself or that you cannot find a third-party library for—is probably not a good starting point for optimization. It is possible to call Matlab functions from within your C code, but this won't help your execution time, for obvious reasons.

Now, the general method of creating C MEX files is to functionize a block of code in your algorithm or to choose a function you have already written to optimize. Now, it is time to create the C version of the function. The general procedure is to create a generic Matlab interface function and then a meat function representing the actual procedure for which you are coding. See Listing 2 for an example of a MEX function. The meat function it calls corresponds to the Matlab file in Listing 1, and parts are available on the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue110/6722.tgz].

mexFunction() is a sort of main() of the MEX-file programming world. It is the actual function called when you call your function in Matlab. The actual name of the function is defined by the name of your compiled .c files, usually the name of the first .c file that you pass to MEX for compilation. On the Linux x86 platform, MEX files have the extension .mexglx. When Matlab is run on the Linux x86 platform, Matlab looks for .mexglx files in the same path and in the same way as it looks for normal .m files, so .mexglx and .m files are interchangeable. A good way to switch between Matlab and optimized code is to change Matlab's search path. I compile c_mLAT.c to c_mLAT.mexglx, and then I can run the compiled code simply by calling c_mLAT() within the Matlab environment. It's a fairly slick system.

Listing 2. A MEX Function

Things get a little complicated when trying to pass data back and forth between Matlab and C. You will notice two double pointers in the argument list of the mexFunction. *plhs[] refers to the return values of the function (Matlab functions can have multiple return values), and *prhs[] refers to the input arguments. The number of input and return arguments also are passed to the function as nlhs and nrhs. The return matrices of the function must be allocated within the mexFuntion() using routines such as mxCreateDoubleMatrix() in order to be passed back correctly to the Matlab environment. The mx functions create memory in the Matlab environment and are handled by the Matlab memory manager, so there is no need to worry about freeing memory created by the mx functions.

Functions beginning with mex are called within the Matlab environment, and with mexCallMATLAB() it is possible to call arbitrary Matlab functions from within your code. From the mexFunction(), you then call your meat function after allocating the output, formatting the input and doing things like argument checking.

Something quite frustrating for C programmers is not dealt with well in the Matlab documentation, however, and that is the fact that Matlab stores its data in column-major format. This can be extremely annoying because you want to be able to use easy-to-understand pointer arithmetic to iterate over the multidimensional input matrices. But it is frustrating and bug-causing to figure out how far to step per iteration and so on. There are three solutions to this, as I see it.

1) Figure out the amount you need to step per iteration manually and think it out. This seems doable and is probably the best solution, but it leads to huge headaches with arrays of dimension equal to or greater than three.

2) Reformat the code before entering the MEX function so it is organized correctly for iterating through the way you want in C. This can be fairly expensive in Matlab, and many times you want to have a drop-in replacement for your original version.

3) Do what I do, and create macros or macro-type items to access memory in Matlab arrays. This is slower than stepping through the arrays and might seem to be an inelegant solution. In my experience, though, it ends up being easy to read and plenty fast. For instance, I created a file named pops.h that contains functions like:

```
extern inline double num3d(double *start, int rows,
                           int cols, int x, int y,
                           int z)
{
    return(*(start + rows * cols * z
                 + rows * y + x));
}
```

which returns the value in a 3-D Matlab array given the array, the number of rows and columns of the array and the xyz location of the data you want to retrieve. It's a little unwieldy, but not too bad. When the code is optimized, it is inserted into the code the same as a preprocessor macro. One could use macros just as well, but I find this method much easier to create and debug. In the end, the speed improvement for doing the loops in C far outweighs the relatively small loss from doing array access this way.

Other than that, the creation of MEX files is not difficult. When it comes time to compile, run the MEX program with the names of the C files you wish to compile. A list of MEX options also is available at the MathWorks web site. After running **mex X.c Y.c Z.c**, you will have a file called X.mexglx that, if it is in your path, you can call as X() on the Matlab command line.

From here, you can rewrite larger and larger portions of your code in C. When it is time to do the full C implementation, it often is beneficial to use the C export feature of Matlab to export the outer Matlab code, because the important parts also have been optimized by you. If things are still not fast enough, it might be a good idea to redo the outer function to deal with memory in a C-friendly fashion. Then you can speed up the loops in the inner-C code, using optimized pointer iteration to access the array values.

In general, the use of Matlab to prototype and develop code can speed things up greatly. However, when you find yourself waiting overnight for Matlab to produce results, only to find that you messed up a small input value, the process of hand-optimizing pieces of the code can be extremely beneficial to making your algorithms practical for use.

Resources

email: sclanton@oeic.net

Sam Clanton is currently an MD/PhD student at The University of Pittsburgh/ Carnegie Mellon, maintaining an affiliation with NASA and QSS Corporation at Ames Research Center in Mountain View, California. He spends his time looking at problems in biological signal processing, computer vision and medical robotics, and is most interested in building information systems like nature does. Sam spends most of his time checking his e-mail (but never writing back) and drinking too much coffee.

Archive Index Issue Table of Contents

Advanced search

Lighting Simulation with Radiance

Anthony W. Kay

Issue #110, June 2003

Go from a sketch to a rendered scene in a matter of hours.

When I wanted to design a log home on my computer to see what it would look like under actual lighting conditions, I tried AutoCAD, 3D Studio Max and numerous off-the-shelf home design packages. None of them provided the realistic output or easy support for dealing with the log walls I desired. I had been playing with a lighting simulation package from the Lawrence Berkeley National Laboratory (LBL) known as Radiance and decided I could get what I wanted much faster by adding utilities to it.

So What Is Radiance?

Radiance is a physical lighting simulation system written primarily by Greg Ward Larson. It has been around since the early 1990s and recently changed licensing from a free-for-noncommercial-use license to the open-source model. The package produces great-looking images that are output in a special format that records both the texture and physical lighting of a scene, much like the professional products LightScape and VIZ 4 by Autodesk.

The packages used for movie and game making are really the graphics equivalent of junk food factories. The end result may be attractive and popular, but it isn't substantial. The physical details of lighting simply aren't as important as speed to movie and game makers, because they have a lot of pixels to push. A two-hour movie has 172,800 individual frames, and games have to run in real time. As a result, light becomes an artifact of an artistic algorithm in most graphics systems and has little basis in reality.

Radiance output is considered a lab-quality simulation of the physics of light (as long as your input is realistic) and has been rigorously tested in the professional world.

Installing Radiance

You can obtain the Radiance source code from radsite.lbl.gov. I recommend getting the source tarball, as the compiled RPMs do not include any of the auxiliary files. Once you have the tarball:

```
   $ tar xzf rad3R4.tar.gz
   $ cd ray
   $ ./makeall install
```

Then, simply answer the questions about where you want to put the software. I use $HOME/radiance/bin for the binaries and $HOME/radiance/lib for the auxiliaries.

The makeall script doesn't install the sample scenes or the documentation, so you have to move those files to a good spot also. For example:

```
   $ mv doc/man $HOME/radiance
   $ mv obj $HOME/radiance
```

Be sure to add these things to the MANPATH and PATH variables in your profile. One caveat: there is an important utility called rview in the package. Unfortunately, Vim also has a utility of the same name, so use a PATH modification or rename Vim's rview. Do *not* rename the Radiance utility, because it is called indirectly by other Radiance utilities.

Radiance Input Basics

New users of Radiance first will notice the lack of an included CAD system for generating the scene description. The package was written for research purposes under UNIX in the early 1990s, and if you look at the file formats, it is obvious they were written for command-line junkies like myself who love the power of pipes and plain-text processing (my own initials are AWK, after all).

Nevertheless, there are utilities for translating geometry from formats like DXF, Wavefront and MGF so you can use any utility that will output such a format. Many of the modelers listed in the application archive of linux.org will output one of these. A Windows-based AutoCAD/Radiance module called Desktop Radiance is also available from the Radiance web site if you happen to own a compatible version of AutoCAD.

The input files of Radiance are human readable, which makes them good candidates for script generation. However, be warned: occasional terms in the documentation will cause accelerated heart rates in passing physicists, such as “watt per square meter per steradian”. Be sure to check out all the documentation on the web site. If you decide to do more than play, you might want to track down a copy of *Rendering With Radiance* by Greg Ward Larson, et al. It is currently out of print, so check with used book dealers.

Rolling Your Own—A Sample Scene

Listing 1 is a scene that includes sky and ground, the material for brass and a sphere with the brass material applied. The sky and ground are standard. The only thing you need to edit for your own scenes are the options to gensky. The values in the listing correspond to noon on November 25 at 33° latitude north and 80° longitude west. Use negative numbers for south and east.

Listing 1. Sample Input Scene for Radiance

Each item in the scene description has the same format. The first line declares an existing material that will be applied to the entry (or void if that doesn't apply), a type name for a material or geometric primitive (like sphere, polygon, plastic or metal) and a user-defined name. The next three groups are the string, integer and real (floating point) parameters for the entry. Each of these starts with an argument count, followed by the actual arguments. They can be spread over as many lines as necessary.

Most entries have only real parameters. This explains the two zeros in the middle of most of the entries; they have no string or integer parameters. The 5 in the last line of brass indicates five real parameters, and the 4 in the last line of the sphere indicates four real parameters. The parameters are straightforward. For example, a sphere needs a center (x, y, z) and a radius.

Materials can be the hardest part of a scene. It is easiest to start by copying existing materials and modifying them to your needs. Read refman.pdf from the web site for more details.

The gensky line at the top of Listing 1 is an embedded command-line utility. Placing an exclamation point at the beginning of a line in a Radiance scene tells the system to run the line as a shell command and use the output as part of the scene. Radiance comes with a number of these utilities, and I've found that writing your own can make scene generation quick and easy.

Moving Stuff Around

Most of the generators put the object at the origin, which isn't likely to be the spot you wanted. The xform utility addresses this issue. Its syntax is:

```
xform -t transx transy transz -rx angle\
      -ry
      -s scalefactor optional_scenefile
```

xform can transform everything in a file, or you can pipe the output of a generator utility to it. It can scale the objects (**-s** ***`factor`***), rotate around an axis (**-ry** ***`angle`*** means rotate around y axis) and translate to new positions (**-t** ***`x y z`*** means translate by *x y z*). You can use the different options multiple times, in any order. The operations are done in the order that they appear on the command line. Also, be sure to pay attention to the exact default position used by a generator. Most of them put a corner at the origin.

Figure 1 shows the effects of xform on a set of cubes generated by genbox. In this image, the viewer is on the +z axis looking back toward the origin. The red axis is +x, and green is +y.

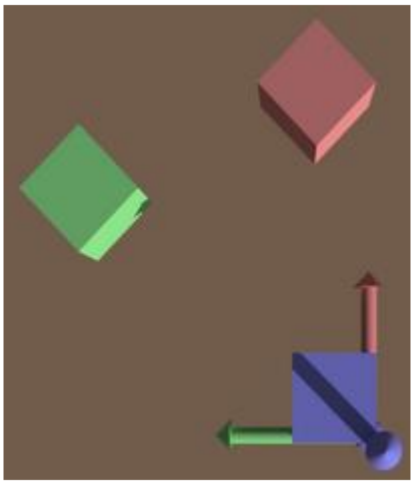


Figure 1. Behavior of xform. The axes are color-coded: +x = red, +y = green and +z = blue. The blue box was not transformed. The red box was rotated, then translated. The green box was translated, then rotated.

The blue box is the unmodified output of a call to genbox. The red box is the same genbox with an xform:

```
!genbox redplastic box1 .5 .5 .5 \
    | xform -rz 45 -t 2 0 0
```

The green box is:

```
!genbox greenplastic box2 .5 .5 .5 \
    | xform -t 2 0 0 -rz 45
```

The materials (like redplastic) were defined right before these calls but are not shown in the listings. You can see the way the order of parameters affects the operations and, thus, the output.

More Complex Scenes

I've written a number of generators in Perl that can be used to put together log cabins and log homes (available electronically). In this article I use genlogwall, genlog and genroof. All of these output in units of inches, even though they sometimes take arguments in feet as input for convenience. The materials I use also are included in the electronic distribution.

The genlog utility generates a capped cylinder, centered along the +x axis (Figure 2). It requires several parameters:

```
genlog material name length_ft diam_inches
```

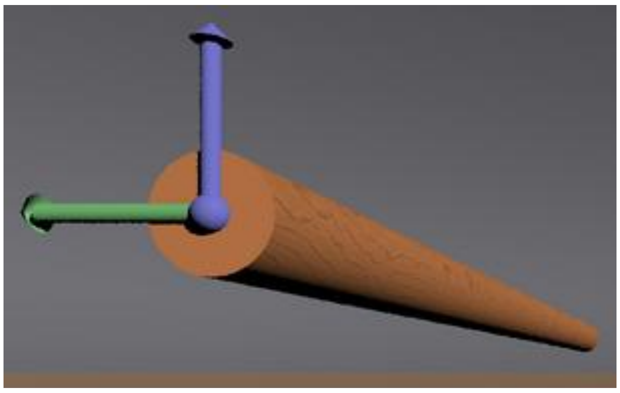


Figure 2. Unmodified output from genlog. Axes are in RGB/xyz order (R = +x, G = +y and B = +z). The x-axis is inside of the log.

The ***material*** should be predefined in your scene, and ***name*** should be something you make up. The predefined materials file I supply has three wood materials, oriented for proper-looking logs: xpine, ypine and zpine. You should choose the material that matches your eventual alignment for the log.

If you wanted to make a ten-feet tall, eight-inch diameter pole pointed in +z, with its base at (15ft, 0ft, 0ft), you'd do this:

```
!genlog zpine mypole 10 8 | xform -ry 90 -t 180 0 0
```

Remember to use the correct units for xform: 180 inches is 15 feet.

The utility genlogwall takes the following form:

```
genlogwall material name length_ft height_ft \
           logdiam_inches [hole_data_file]
```

The optional parameter is a data file that indicates what holes should be in the wall and what should be in each hole, such as a window or door. At this point, it will help if we work from the floor plan in Figure 3.

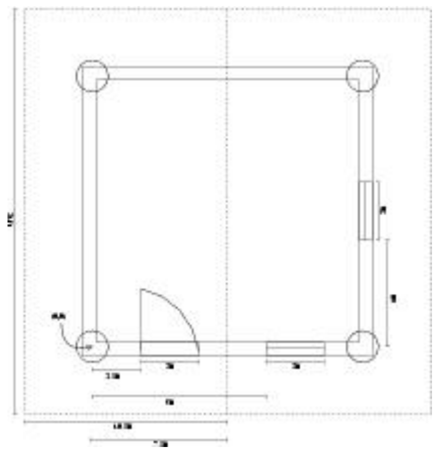


Figure 3. A floor plan for a cabin. Measurements for the holes are taken from the endpoints of the wall in which they appear.

There are four walls, each 15 feet long. I chose the southwest corner of the cabin to be (0,0,0), increasing x to the east, increasing y to the north and increasing z up. This orientation faces the building south, according to the standard-generated sky from gensky. genlogwall always places the generated wall at (0,0,0) along the x axis, as shown in Figure 4.

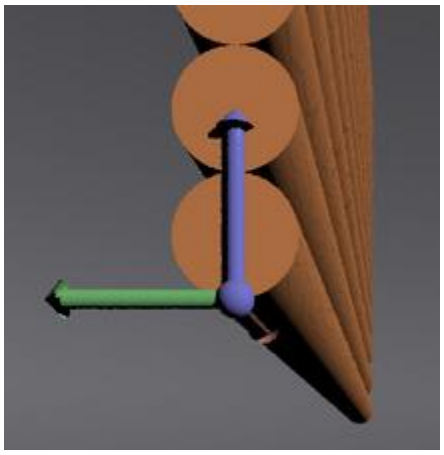


Figure 4. Nontransformed wall output from genlogwall. The axes' colors are in RGB/xyz order.

The hole data file is simple. One hole description per line:

```
holebottom_ft holetop_ft holestart_ft
width_ft[:w|d]
```

The first two numbers are measured from the floor and the latter two from the beginning edge of the wall (x = 0, x increasing). The optional tag on the end indicates that you want the wall generator to fill the hole with a window or door. Our floor plan calls for two such hole description files (see Listings 2 and 3). You can use the same data file for multiple walls, but only if you want them to have the same set of holes.

Listing 2. Hole Description File (holes/southwall.holes) for the South Wall

Listing 3. Hole Description File (holes/eastwall.holes) for the East Wall

The final touch to our cabin is the roof. Generic roof generation is tricky, so the genroof tool makes you do a bit more work than the others.

genroof generates planar pieces of roof; use it multiple times with xform to generate a whole roof.

A data file is required for genroof. In the data file, provide the x-y coordinates (in feet) of the vertexes of the piece of roof, as read from the floor plan in counterclockwise order around the edge. The vertexes all must be in the positive quadrant, and any edge of the peak must run parallel to the x axis.

To figure out the points from the plan, rotate it so the peak of the roof runs left to right. Now ignore the bottom half of the roof, and think of the lower-left corner of the top half as your new origin. Your points are then (0,0), (21,0), (21, 10.5) and back to (0,0). Not too bad.

The points should be entered one per line, separated with a space, ending with the letter b, mp or p to indicate whether the point is at the bottom, somewhere in the middle or the peak of the roof section. This is necessary because it is possible to generate odd-shaped roof sections for unusual roofs.

You also need coordinates for the end cap if you want the genroof utility to fill in the ends of the roof with logs. Looking at the floor plan in our roof orientation, you easily can see that the caps should be along the walls at (3,0)®(3,7.5) and (18,0)®(18,7.5). Add this information to the end of the roof data file prefixed with the marker c:. The completed roof data file for the cabin is shown in Listing 4.

Listing 4. The Data for Our Roof Generator (roofdata)

The command-line call for genroof is:

```
genroof -o overhang_ft typename name \
        roofdatafile height_ft thickness_in
```

The overhang parameter allows the utility to adjust the position of the piece so you can transform it to the height of the wall without worrying about meeting the slope of the roof. The completed cabin scene in Listing 5 shows the z transform for the roof pieces matches the height of the walls, even though the overhangs will droop below the top of the walls.

Our roof is symmetrical, so we use the same genroof with a different xform to make the other half. That's easy.

Listing 5. The Scene File for a Complete Cabin

Viewing a Scene

Radiance comes with a utility called rad that works like UNIX make. The input file to rad has a series of variables that tell it how you want to render a scene, and it figures out how to call the many other programs used to simulate the light. A commented sample is shown in Listing 6.

Listing 6. cabin.rif: an Input File Telling the rad Utility What Options and Files Are Used to Generate Pictures

Most of the variables that take filenames can be defined as many times as there are files that apply. The view variable also can be defined many times. Each view definition causes the generation of a picture. You should include a name for the view, where you want to put your eyeball (**-vp** ***`viewpoint`***), what direction you want to look (**-vd** ***`view direction vector`***) and which way is up (**-vu** ***`view up vector`***). I also like the -vt option for generating angular (fish-eye) views.

Using the H value with the various quality settings can take a long time (more than ten hours on 2GHz machines). Most times, the H setting is overkill, and M works fine. Use L for interactive rendering. The documentation and a little experimentation will help you figure out what is best for your scene.

To view a scene interactively, use the command:

```
$ rad -o x11 cabin.rif
```

The scene may appear bright and washed out when interactively viewing. Type **e**, press Enter, then click on a bright spot in the image to fix the exposure. You don't have to wait for the render to finish.

You can experiment with the exposure as much as you want. The dynamic range of Radiance image data far exceeds the dynamic range of your monitor. This means you can end up with a completely dark or completely white image that can be adjusted to your display without loss of data. This is drastically different from normal image files where adjusting the brightness too far can cause permanent loss of information.

You can load an alternate view from the rif file while interactively rendering with the **L** command. For example, if you have a view named interior in your rif file, typing **L interior** loads it. You can input a view manually by typing **v** and pressing Enter. Simply answer the prompts. Type **q** and press Enter to quit the interactive renderer.

To generate images of all of your views use:

```
  $ rad cabin.rif
```

Then view the images with:

```
  $ ximage *.pic
```

You can adjust the exposure of an image in ximage by clicking on the image and pressing A for auto-exposure, H for human eye response or = to adjust based on the pixel you clicked.

Figures 5 and 6 show two completed daylight simulations of our cabin.

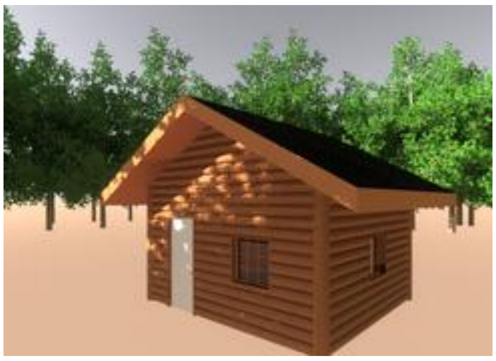


Figure 5. Perspective view of the cabin in daylight with some trees added for interest. Rendering at 3300 × 2200 for publishing quality took about five hours on a 1.7GHz laptop.

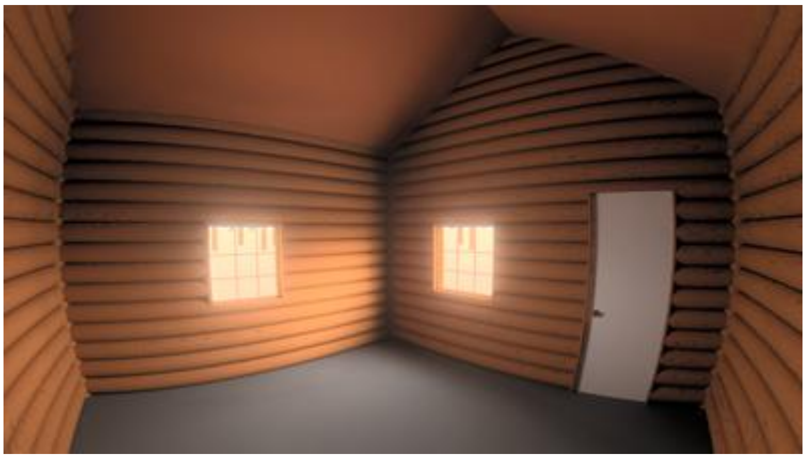


Figure 6. An angular view of the cabin interior lit only by window daylight. Rendering at 3300 × 2200 for publishing took about five hours on a 1.7GHz laptop.

You can take light readings averaged over an area from ximage. Simply drag out a box and press L for luminance or Enter for radiance values. For a quick physics tutorial of the meaning of these numbers, see www.intl-light.com/handbook/rad.html. Press Q to quit an image.

I've covered a very small part of Radiance in this article due to space limitations. If you want to further populate your scenes with the clutter of daily life, be sure to check out the links from the Radiance web site for furniture and plants.

email: awkay69@hotmail.com

Anthony W. Kay is a computer programming consultant in Eugene, Oregon. When he's not simulating trees as building materials he goes hiking among the live ones.

Archive Index Issue Table of Contents

Advanced search

Linux for Science Museums

Len Kaplan

Issue #110, June 2003

Put your knowledge to good use by helping develop innovative, fun and educational exhibits for a whole new audience.

Over the past several years, my son and I have spent a significant amount of volunteer time creating exhibit software for the Sciencenter in Ithaca, New York. This article discusses how this project is similar to other software development projects, how it is different and how using Linux has been beneficial. It's not only about Linux or software; it's also about the processes involved, and what you may encounter if you have the opportunity to work in this type of environment.

The Target Audience

Science museum exhibit software must be designed with a different audience from the typical computer program, as the software generally is used in a different manner. My earlier Microsoft Windows-based exhibits—*Measurement Factory* and *Fabulous Features*—were designed for children up to approximately sixth grade, while the Linux-based exhibits—*Sound Studio* and *Traffic Jam*—were designed for fifth to eighth graders. The software must be simple enough that the target audience understands a significant amount of what they see on the screen and how it relates to the exhibit. A computer in an exhibit is not necessarily the exhibit; it may act as a guide or simply be another “manipulative” along with real-world objects.

A second “audience” also must be considered, the museum staff. Floor personnel (many of whom are volunteers) shouldn't have to be trained in the vagaries of each computer. Exhibit software and hardware also should come up and run automatically when the systems are powered on first thing in the morning, as museums tend to turn off at least some circuit breakers when they close in the evening. Trust me, you want the museum staff to like your exhibit—

if it makes their life difficult, the exhibit may sit there with a sign saying "broken".

The Competition

Possibly the most important thing to keep in mind when you're developing your dream exhibit is this: when you build a non-hidden computer into a museum exhibit aimed at younger visitors, you're competing with every video game they've played and every television show or movie they've watched. Your exhibit, therefore, must be extremely cool.

Measurement Factory is cool because visitors can weigh themselves, measure their height, test their grip strength, compare themselves to others of their age and get a certificate when they're done.

Traffic Jam (Figures 1-3) is cool because visitors can play with traffic lights and prevent traffic jams—or not, if they prefer.

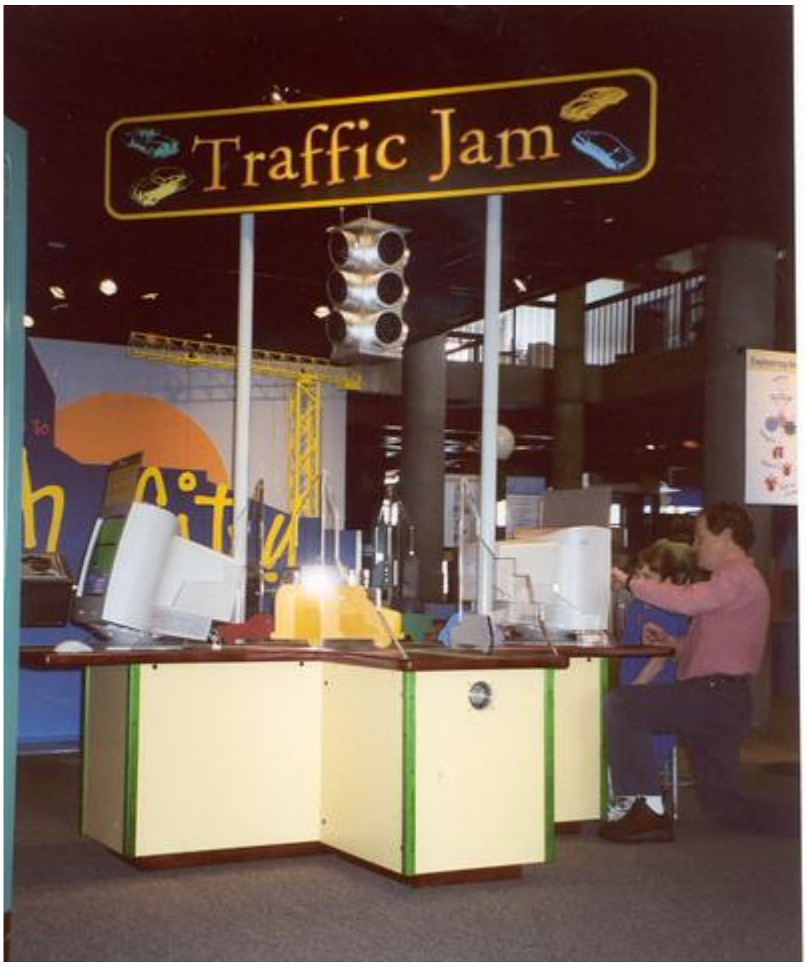
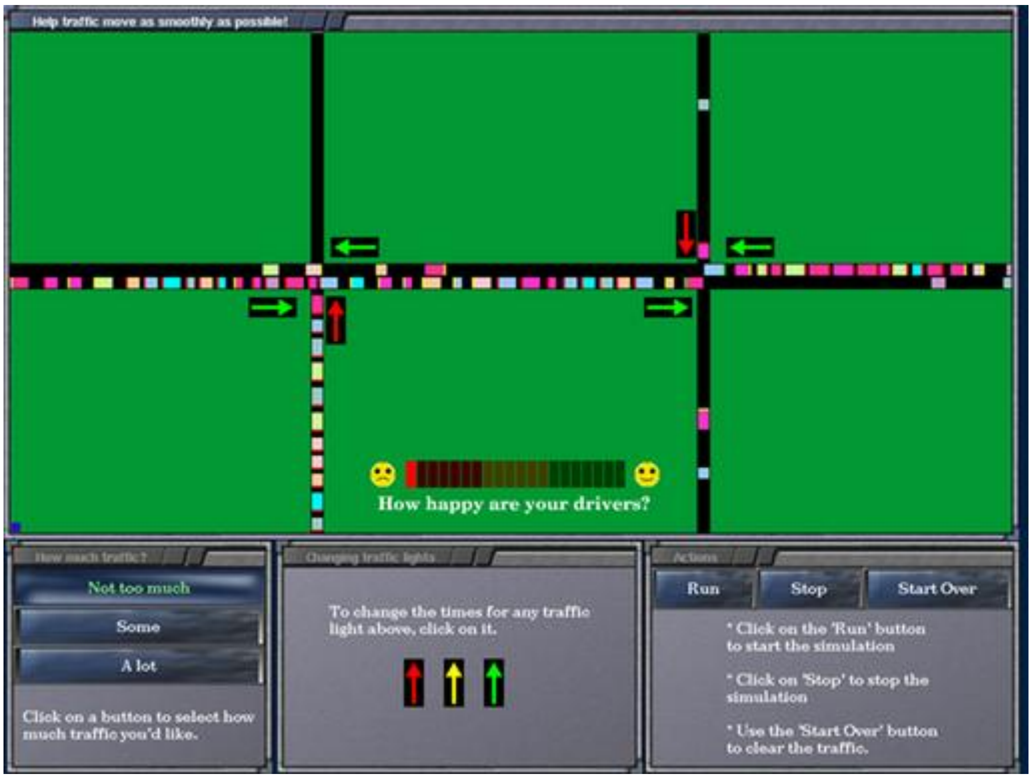


Figure 1.



Click on a button to select how much traffic you'd like.

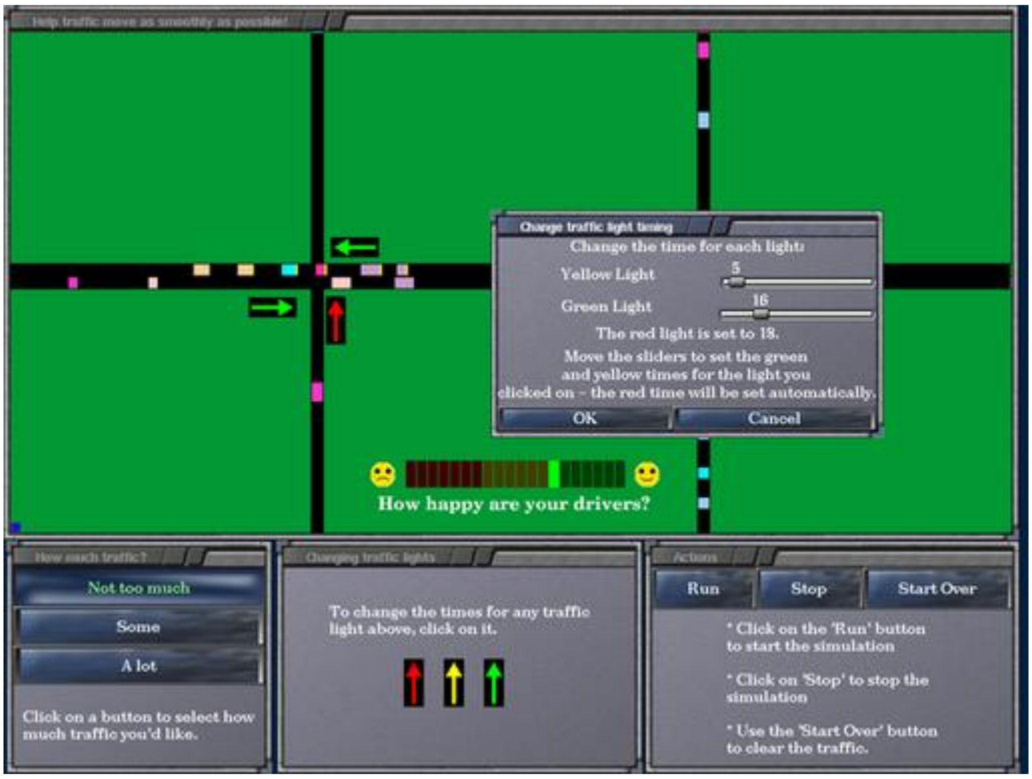
To change the times for any traffic light above, click on it.

* Click on the 'Run' button to start the simulation

* Click on 'Stop' to stop the simulation

* Use the 'Start Over' button to clear the traffic.

Figure 2. Heavy Traffic in



Change the time for each light:

The red light is set to 18.

Move the sliders to set the green and yellow times for the light you clicked on – the red time will be set automatically.

Click on a button to select how much traffic you'd like.

To change the times for any traffic light above, click on it.

* Click on the 'Run' button to start the simulation

* Click on 'Stop' to stop the simulation

* Use the 'Start Over' button to clear the traffic.

Figure 3. Changing Traffic Light Times in

Sound Studio (Figures 4-6) lets visitors record themselves and their friends on a multitrack recorder and play with simple special effects, such as echoes.

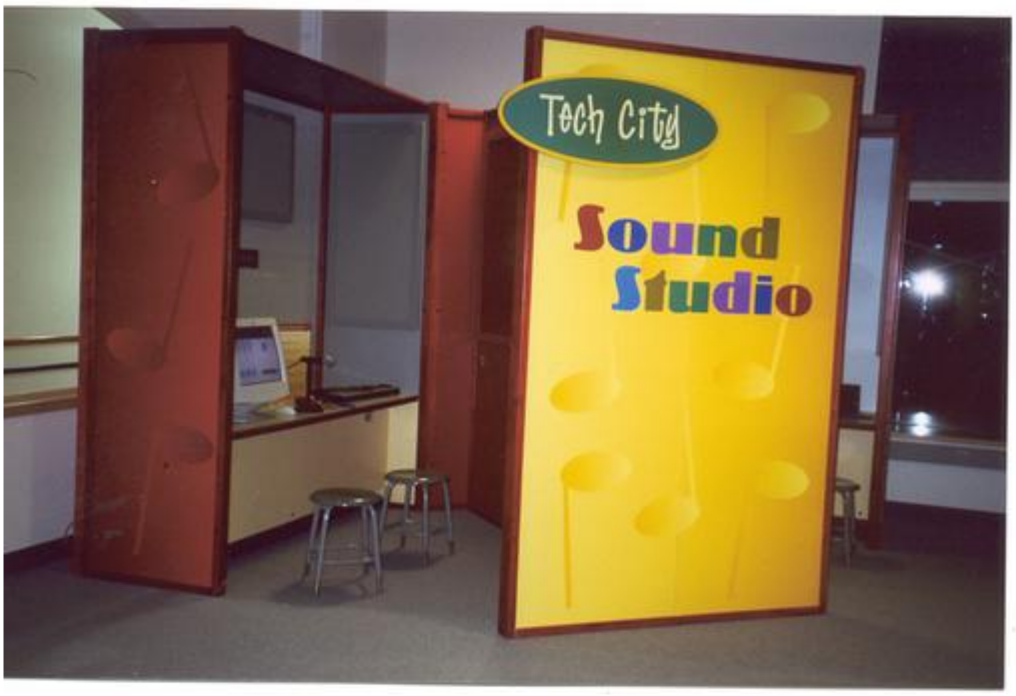


Figure 4.

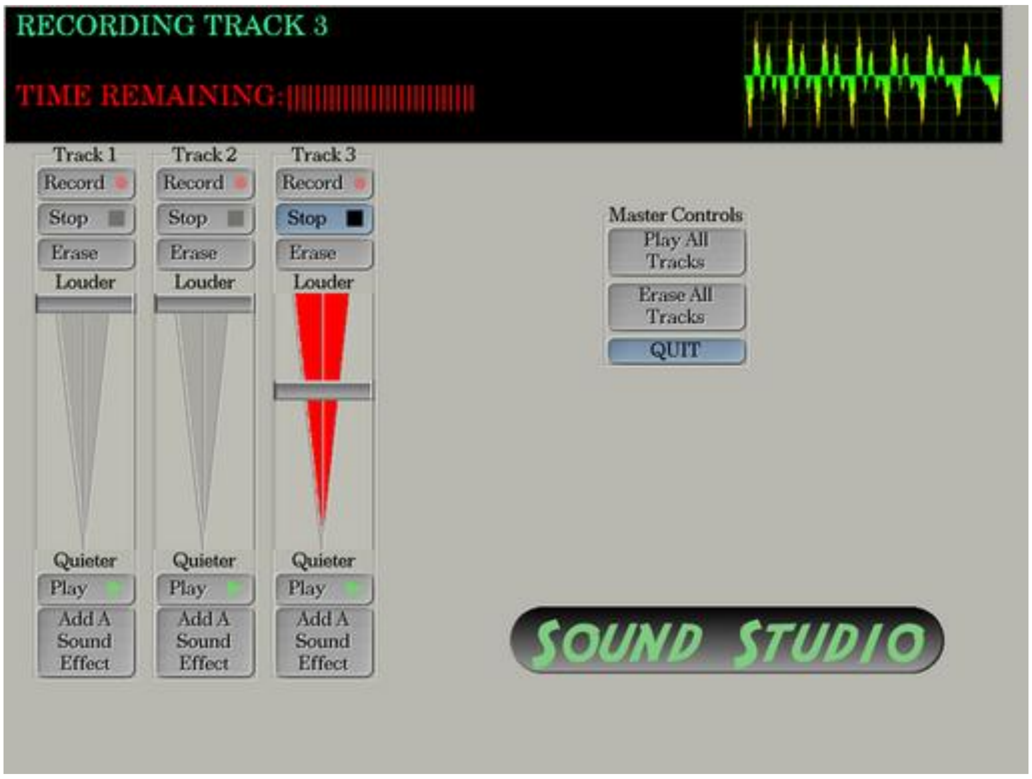


Figure 5. Recording Using

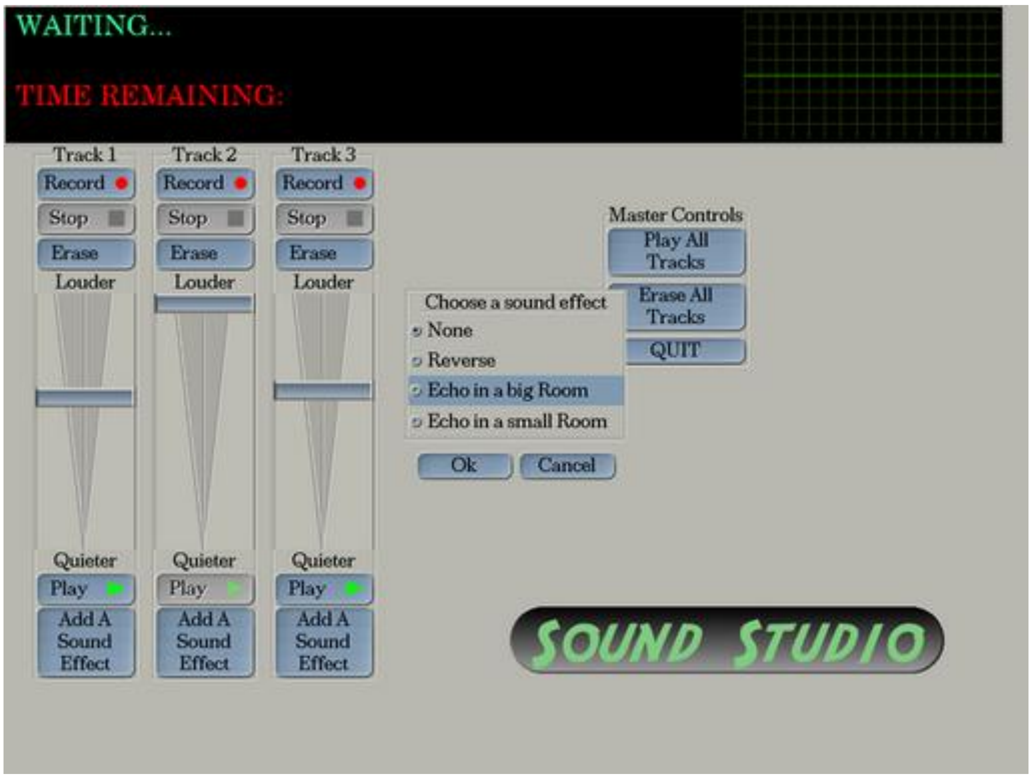


Figure 6. Choosing a Special Effect with

Cool has a definite limit, however. For instance, it's not a good idea to make the “you made a mistake” annunciator—sound, animation, whatever—too cool; otherwise visitors will consider that the goal. Don't defeat the purpose of the exhibit.

The Development Process

An orderly development process is important here because of the educational goals and target audience. Some of the following actually may be quite informal, but it is important nonetheless. Your development cycle may not be exactly like this, of course.

The kickoff meeting: if you're building a standalone exhibit that is not part of a coordinated exhibition, this may not happen. *Tech City*, however, was to consist of around a dozen coordinated exhibits, with a central theme of engineering. Therefore, an all-day kick-off meeting was held to discuss project goals and to arrive at an initial focus. I've attended several meetings of this type, and all were well worth the time. In addition to getting a feel for the project, you'll pick up a few good ideas and meet people who may be helpful later on in the process.

Brainstorming: put your feet up and don't even worry about Linux until you come up with a few good ideas. On the other hand, if you know of a good existing application that could be modified and made useful, mention it and make note. While kicking around your cool ideas, keep handy a list of the project requirements. These can range from something quite simple, such as “demonstrate a type of engineering”, to something more difficult, like “demonstrate Heisenberg's Uncertainty Principle for elementary schoolchildren in a nonstructured, graphical manner using virtual manipulatives.” Always keep the requirements in mind, but I can guarantee that they'll change right up to the day you ship.

Prototype construction: this phase is the same as building prototypes anywhere else, but it can be even more fun because of the people you're working with. The museum people we've worked with over the years have been intelligent, creative and enthusiastic about their jobs. Don't be too shocked by the appearance of the kiosk prototype if your exhibit requires one. There's probably going to be a lot of duct tape and cardboard involved.

First review: this is where you show off your prototype for the first time, for critiquing. Don't worry if the adults involved in the process don't think your exhibit is as cool as you know it is. Remember, most adults don't think like kids. Show your prototype to a few kids in the target age range, informally.

Target test: this might be the most fun part of building your exhibit. It's the first, and possibly second and third, time that it's set up for museum visitors to try, so you and possibly other people can observe the results. Expect to go through this process several times. Important questions to consider include: Is the exhibit inviting to visitors? How long do visitors stay at the exhibit? Do several visitors use it together? Do visitors learn what you're trying to teach? Do visitors use the exhibit the way you intended? Can visitors figure out how to use the exhibit successfully? Does it appeal to the correct age bracket?

You may be able to interview visitors after they've tried your exhibit, or museum personnel may do so and report their results to you. This is probably the most valuable information you'll receive; kids can be brutally honest.

Professional evaluation: there are firms with expertise in evaluating educational programs such as museum exhibits. If your exhibit is being built as part of a National Science Foundation (NSF) grant, it likely will be evaluated by one of them. We've only had the opportunity to deal with one of these firms, Inverness Research Associates, but found their comments to be extremely helpful. If you have the chance, observe them at work—you'll learn a lot.

Revision, including code cleanup: like the prototype construction step, this is similar to what it would be like anywhere else. It's well worth taking the time to add extra comments and clean up any kludged code. Museum software tends to accumulate extra bits and pieces. If you need to modify your software sometime in the future, the extra time spent now may more than pay for itself later. At this stage, the nice versions of the non-computer parts of your exhibit are probably being built as well.

Deployment: again, this isn't much different from anywhere else, though you'll want to keep the turnkey and no-maintenance requirements of this sort of project in mind. If your software is shipping as part of a larger exhibition, remember that museum personnel will be extremely busy at this point, so make things as easy as possible for them. System deployment for *Tech City* ultimately consisted of setting up the systems—four targets, plus a spare—installing Linux on each and testing with the CD-ROMs we'd already prepared (see the Tips and Tricks section below).

Hardware Selection

Your software may have to run on an old PC that has been sitting around unused or on a latest-generation machine that has been donated specifically for the project. Or, if you're lucky—from the Linux driver perspective at least—you'll get a machine that is one generation old.

If you're able to specify the deliverable hardware, be conservative; try to look at this from the turnkey system viewpoint, as opposed to office desktop. Well-debugged hardware and drivers are probably going to be worth more to you than the latest-and-greatest devices, and saving a few degrees of cabinet temperature by using a somewhat slower processor and video board might prevent a failure six months from now in a faraway museum on a warm summer day. Don't be stingy with system cooling. If you can, specify good-quality fans, and make sure that whoever builds the exhibit kiosk, if there is one, provides for adequate airflow to and from the computer.

The *Tech City* software was deployed on donated Hewlett-Packard Vectra 400s, each with a 1GHz PIII processor, an i810 chipset with onboard video and a 20GB HD. We also added an Sound Blaster 16 PCI card to each machine, as *Sound Studio* requires either a full-duplex sound card or two cards without that capability. Each system was supplied with a 19" monitor, also donated by Hewlett-Packard.

Tool and Library Selection

Not knowing anything about your project, I can't recommend specific tools and libraries. I won't even try to recommend implementation languages to you. I can, however, suggest a few guidelines that worked for us.

Experienced developers probably have heard this already, but if an existing package is close to what you need to accomplish a particular task, think about modifying and using it. The *Traffic Jam* user interface is split across four windows—traffic display, density setting, control and other information. We used GTK+ because of its extensive theme capabilities and the icewm window manager for the same reason. We did, however, need to modify the latter slightly.

Carefully read the licenses for the code you want to borrow, of course. Respect the author's wishes, and don't create any liabilities for the museum. As with hardware selection, being conservative is good. If you really don't need the features available in the latest version of a tool or library, don't rush to install it—known bugs are easier to work around than unknown ones. Don't forget about maintenance and project lifetime; either you or someone else may have to modify this project in the future. A nonrelease version of a language or library might seem wonderful and stable now, but two years from now it might be quite different and somewhat incompatible. Even g++ changed in the four years from when I wrote the prototype Linux version of *Traffic Jam* until the final software was delivered.

Lastly, I'll go out on a limb by suggesting that you also be conservative with regard to selecting a Linux distribution. Latest and greatest may be ideal for your desktop, but again, reliability is far more important in the field. We chose Debian 3.0r0 for our final system deployment soon after it was released, but because Debian has a reputation as a conservative distribution, we felt comfortable with that decision.

Tips and Tricks

Here are several of the problems we encountered and how we solved them. The solutions may not be optimal, but they worked well for us.

Problem: How did we set the systems up as turnkey? They must power into the exhibit software automatically; no user intervention required. Museum staff also must be able to update software easily, as required.

Solution: We solved this by mounting the CD-ROM application directory over (on top of) the exhibit home directory, /home/techcity for example, and automatically logging on as that user at startup. If the appropriate CD-ROM isn't present—each deployment CD contains the software for only one exhibit—the console displays a message asking the user to put it in and then reboot. (Although not accessible to visitors, a keyboard is in the cabinet with each computer.) The reboot monitor watches a FIFO for either an R to reboot the system or a Q to quit, though we also considered using it in other ways. See Listing 1 for pseudo-code describing this process, Listing 2 for our autostart file and Listing 3 for a sample .xinitrc.

Listing 1. Pseudo-Code for Turnkey Application Startup

Listing 2. File 599xx-mytechcity, Inserted into /etc/rc2.d to Sequence Exhibit Software Startup

Problem: This type of application generally needs fine-tuning. How did we accomplish this?

Listing 3. *Sound Studio* Application .xinitrc File

Solution: Our first idea was to use a configuration file format based on the Windows .ini file. This would have worked for *Sound Studio* but not for *Traffic Jam*. The latter required, among other things, the defining of multiple vehicle types, which made XML's ability to represent easily multiple instances of the same class useful. My son coded a C library designed to run on top of xmllib2, which allows our software to access the various elements as a tree—based on the path to that element within the document, in other words.

Listing 4 shows a section of the DTD for the *Traffic Jam* configuration file, and Listing 5 contains the associated section of the file. Listing 6 is a section of the code used to load vehicle physics—notice how Cfg points to a C++ object wrapped around the library functions mentioned previously.

Listing 4. The Vehicle-Definition Section of the DTD for the *Traffic Jam* Configuration File

Listing 5. The Vehicle-Definition Section of the *Traffic Jam* Configuration File

Listing 6. Loading Vehicle Physics Constants from the XML Configuration File

Problem: How to deal with window manager security? Visitors shouldn't be able to exit the software, bring up other applications, move windows around and so on.

Solution: We found that when visitors had access to the keyboard during early testing, they were quite good at finding their way out of the applications. We fixed this with a combination of configuration file and code changes to icewm to disable pop-up menus, window moves and window resizes. Also, neither exhibit requires the use of a keyboard, so that's kept inside the kiosk during normal operation.

Conclusion

Educational software may not be your cup of tea, but if it is, try to find a science museum to help. They like seeing new faces, they can always use the help, and I can promise that you'll get back more than you give.

I'd like to thank the staff and volunteers at the Sciencenter, Ithaca, New York, for running a terrific museum and encouraging people to think outside the box; check out their web site at www.sciencenter.org.

Credits

Photographs courtesy of the Sciencenter, Ithaca, New York.

email: lkaplan@dreamscape.com

Len Kaplan has been programming since small computers were the size of refrigerators. In addition to Linux, he's fascinated by embedded systems and enjoys model railroading in his spare time. Len can be reached at lkaplan@nlzero.com.

Archive Index Issue Table of Contents

Advanced search

The Driver Model Core, Part I

Greg Kroah-Hartman

Issue #110, June 2003

The 2.5 kernel implements a unified device driver model that will make driver development for 2.6 easier.

In the 2.5 Linux kernel development series, a unified device driver model framework was created by Pat Mochel. This framework consists of a number of common structures and functions all device driver subsystems have been converted to use. It also consists of some generic structures that are starting to be used outside of the driver code by other parts of the kernel. This article discusses parts of the driver model and provides an example of how to convert a specific device driver subsystem to the driver model.

Buses, Devices and Classes

The driver framework breaks all things down into buses, devices and classes. Using these primitives, it controls how drivers are matched up with physical and virtual devices, and it shows the user how all of these things are interconnected.

A bus can be described as something with devices connected to it. Examples of buses are PCI, USB, i2c, PCMCIA and SCSI. Usually only one bus driver controls the activity on a bus, and it provides a type of bridge from the bus it is on to the bus it controls.

An example of a bridge is a USB controller that lives on the PCI bus. It talks to the PCI bus as a PCI device and looks to the kernel as a PCI driver. But it controls all access to that specific USB bus, talking to the different USB devices plugged in to it.

Buses are represented in the kernel with the struct bus_type definition, found in include/linux/device.h. All buses in the system are shown to the user in subdirectories of the sysfs directory /sys/bus/.

Sidebar: Sysfs

Devices are physical or virtual devices that reside on a bus. They are represented by the struct device definition and are created by the bus when the bus sees they are present in the system. Usually only one driver controls a specific device at one time. They can be seen in the /sys/devices directory as a giant tree of all devices in the system or in the /sys/bus/*BUS_TYPE*/devices/ directory for a specific type of device.

Devices also have drivers assigned to them that control how to talk to the device across a specific bus. Some drivers know how to talk to multiple buses, such as the Tulip network driver, which can talk to PCI and ISA Tulip devices. All drivers are represented by the struct device_driver definition. They can be seen in sysfs at /sys/bus/*BUS_TYPE*/drivers/. Drivers register with a specific bus and export a list of different types of devices they can support. The bus matches the devices and drivers based on this list of exported devices. The list also is exported to user space so the /sbin/hotplug tools can be used to match drivers to devices that do not have drivers already loaded. See my article, “Hot Plug”, in the April 2002 issue of *Linux Journal* for more information on this interface and how it works [also available at www.linuxjournal.com/article/5604].

Classes here do not take the general object-oriented definition but, rather, are things that provide a function to the user. They are not bus- or device-specific things but functionally look to the user as the same type of device. Examples of classes are audio devices; pointing devices, such as mice and touchpads; keyboards; joysticks; IDE disks; and tty devices. The kernel always has had these kinds of devices, and they traditionally have been grouped together by major/minor number range, so the user can access them easily. Classes are represented in the kernel with the struct device_class definition, and they can be seen as subdirectories of the sysfs directory /sys/class/.

For a description of the whole driver model, along with an introduction to the structures below the driver model that do all of the real work, see the thorough document at www.kernel.org/pub/linux/kernel/people/mochel/doc/lca/driver-model-lca2003.tar.gz. It was written by Pat Mochel for the 2003 Linux.Conf.Au conference.

Theory in Action

All of the above descriptions sound great on paper, but how does the driver model actually affect the kernel code? To show this, let us walk through how the i2c driver subsystem was modified to support this driver model.

The i2c code has lived outside of the main kernel tree for a long time, and it was offered as a patch for the 2.0, 2.2 and 2.4 kernels. It also was the subject of “Using the i2c Bus”, by Simon G. Vogl, one of the main authors of the code [*LJ*, March 1997, www.linuxjournal.com/article/1342]. In the 2.4 development cycle, a number of the i2c core files and a few i2c bus drivers were accepted into the main kernel. In the 2.5 development cycle, a few more drivers were added; hopefully, all of them eventually will migrate into the main tree. For a good description of the i2c code, what devices it supports and how to use it, see the main development site at secure.netroedge.com/~lm78/index.html.

When loaded, the i2c bus drivers, which talk to the i2c controller chips, export a number of files in the /proc/bus directory. When an i2c device driver is loaded and bound to an i2c device, it exports files and directories in the /proc/sys/dev/sensors directory. By moving the representation of the devices and buses to the kernel driver core, all of these separate files can be shown in their proper places in /sys.

The i2c Bus

The main i2c bus subsystem needs to be declared in the kernel and registered with the driver core. To accomplish this, the following code was added to drivers/i2c/i2c-core.c:

```
static int i2c_device_match(struct device *dev,
                          struct device_driver *drv)
{
    return 1;
}
struct bus_type i2c_bus_type = {
    .name =    "i2c",
    .match =   i2c_device_match,
};
```

The name field says what the bus should be called, and the match field points to our match function. Right now, the match function is left alone, always returning 1 whenever the driver core wants to try to match a driver with a device. This logic will be modified at a later time.

Then, in the i2c core startup code, the i2c_bus_type is registered with a call to:

```
bus_register(&i2c_bus_type);
```

When the i2c core is shut down, a call is added to unregister this bus:

```
bus_unregister(&i2c_bus_type);
```

When the above code runs, the following tree is created in sysfs:

```
$ tree /sys/bus/i2c/
/sys/bus/i2c/
|-- devices
'-- drivers
```

When the i2c core is removed from the system, the above directories are removed. This is all that is needed to create the i2c bus.

i2c Adapters

An i2c bus by itself is pretty boring. Now, the i2c bus adapter drivers need to be modified to register themselves with this bus. To do this, a struct device variable is added to the struct i2c_adapter structure:

```
struct i2c_adapter {
    .....
    struct device dev;
};
```

A to_i2c_adapter() macro is defined as:

```
#define to_i2c_adapter(d) container_of(d,
struct i2c_adapter, dev)
```

This macro is used by the i2c core to get a pointer to a real i2c_adapter structure whenever the driver core passes it a pointer to a struct device.

The struct device in the i2c_adapter is a whole variable declared within the structure, not merely a pointer. This is done so when the driver core passes a pointer to a struct device, the i2c code can use the to_i2c_adapter() macro to get a pointer to the real i2c_adapter structure.

Sidebar: container_of()

The individual struct i2c_driver variables are declared in the different i2c bus drivers. For example, in the i2c-piix4.c driver, there is a variable called piix4_adapter of type struct i2c_driver. This variable is passed to the i2c core in the i2c_add_adapter() function, when a PIIX4 adapter is seen by the i2c-piix4 driver.

In the i2c-piix4.c driver, before i2c_add_adapter() is called, a pointer to the parent device of the PIIX4 adapter needs to be saved within the i2c_driver structure. This is done with a single line of code:

```
piix4_adapter.dev.parent = &dev->dev;
```

dev is a pointer to the struct pci_dev that is passed to the i2c-piix4 driver's PCI probe function; the PIIX4 is a PCI-based device.

To link the i2c_driver variable to the sysfs tree, the following lines of code are added to the i2c_add_adapter() function:

```
/* add the adapter to the driver core.
 * The parent pointer should already
 * have been set up.
 */
sprintf(adap->dev.bus_id, "i2c-%d", i);
strcpy(adap->dev.name, "i2c controller");
device_register(&adap->dev);
```

With this code, when the PIIX4 device is detected by the driver, an i2c bus tree is created and linked to the controlling PCI device:

```
$ tree /sys/devices/pci0/00:07.3/i2c-0
/sys/devices/pci0/00:07.3/i2c-0
|-- name
`-- power
```

When the i2c-piix4 driver is unloaded, the i2c_del_adapter() function is called. The following line of code is added to clean up the i2c bus device:

```
/* clean up the sysfs representation */
device_unregister(&adap->dev);
```

i2c Drivers

The i2c bus has a number of different drivers that control access to a wide range of i2c devices that live on the i2c bus. These drivers are declared with a struct i2c_driver structure. Within this structure, a struct device_driver variable is added to allow these drivers to be registered with the driver core:

```
struct i2c_driver {
    .....
    struct device_driver driver;
};
```

And, a to_i2c_driver() macro is defined as:

```
#define to_i2c_driver(d) container_of(d, struct
i2c_driver, driver)
```

An i2c driver registers itself with the i2c core in a call to i2c_add_driver(). To add driver core support for i2c drivers, the following lines of code are added to this function:

```
/* add the driver to the list of
 *i2c drivers in the driver core */
driver->driver.name = driver->name;
driver->driver.bus = &i2c_bus_type;
driver->driver.probe = i2c_device_probe;
driver->driver.remove = i2c_device_remove;

retval = driver_register(&driver->driver);
if (retval)
    return retval;
```

This sets up the driver core structure to have the same name as the driver and a bus type of i2c_bus_type; the probe and remove functions are set to local i2c functions. For now, these functions are declared as:

```
int i2c_device_probe(struct device *dev)
{
    return -ENODEV;
}

int i2c_device_remove(struct device *dev)
{
    return 0;
}
```

because no i2c device support has been added yet. These functions will be called when an i2c device is added or removed from the driver core, but that will be described in the next column.

When the i2c_add_driver() is called, the driver is registered with the i2c_bus_type, and it shows up in sysfs as:

```
$ tree /sys/bus/i2c/
/sys/bus/i2c/
|-- devices
`-- drivers
    |-- EEPROM READER
    `-- W83781D sensors
```

To remove an i2c driver from the system, the i2c_del_driver() function is called. In order to remove the i2c driver from the driver core that was registered with the call to driver_register, the following line of code is added to this function:

```
driver_unregister(&driver->driver);
```

Conclusion

We have covered the basics of the new driver core, and to help understand how this driver model affects different subsystems, we covered the changes needed to convert the i2c core to support the kernel core bus and driver model. In the next Driving Me Nuts column, we will cover how to add i2c device support and how the probe() and remove() functions should look.

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM, doing various Linux kernel-related things and can be reached at greg@kroah.com.

Archive Index Issue Table of Contents

Advanced search

Memory Leak Detection in C++

Cal Erickson

Issue #110, June 2003

Don't put off fixing memory leaks. Make one or more of these convenient tools a part of your development process.

An earlier article [“Memory Leak Detection in Embedded Systems”, *LJ*, September 2002, available at www.linuxjournal.com/article/6059] discussed the detection of memory leaks when using C as the programming language. This article discusses the problem of detecting memory leaks in C++ programs. The tools discussed here detect application program errors, not kernel memory leaks. All of these tools have been used with the MontaVista Linux Professional Edition 2.1 and 3.0 products, and one of them, dmalloc, ships with MontaVista Linux.

When developing application programs for embedded systems, designers and programmers must take great care with using system memory resources. Unlike workstations, embedded systems have a finite memory source. Typically, no swap area is available to idle programs. When the system uses up all of its resources, nothing is left to do but panic and start over or kill some programs to make room for the needed resources. Therefore, it is important to write programs that do not leak memory. Many tools aid programmers in finding these resource leaks. All of the tools discussed here come with their own test programs.

One method of testing, which I have seen used successfully by application developers, involves using a workstation to develop prototype code and debugging as much as possible on it. Using memory leak tools in this manner is strongly advised. By debugging on a workstation, the application programmer can be assured that the transition to the target processor will be easier. A major reason for using workstations is they are cheap, and everybody involved has one. Targets, on the other hand, are usually few and in great demand.

Most memory leak detection programs are available as full source. They typically have been built on an x86-based platform. Running them on non-x86 targets requires some porting. This porting effort could be as simple as a recompile, link and run, or it could require changing some assembler code from one platform to another. Some of the tools come with hints and suggestions for use in cross-compiling environments.

dmalloc

The author of dmalloc, a tool I covered in detail in the September 2002 article, states that his knowledge of C++ is limited, and thus the C++ detection of memory leaks also is limited. In order to use dmalloc with C++ and threads, it has been necessary to link the application as static.

ccmalloc

The ccmalloc tool is a memory profiler with a simple usage model that supports dynamically linked libraries but not dlopen. It detects memory leaks, multiple de-allocation of the same data, underwrites and overwrites and writes to already de-allocated data. It displays allocation and de-allocation statistics. It is applicable to optimized and stripped code and supports C++. It also provides file and line number information for the whole call chain, not only for the immediate caller of malloc/free, and it supports C++. No recompilation is needed to use ccmalloc; simply link it with **-lccmalloc -ldl** or **ccmalloc.o -ldl**. ccmalloc provides efficient representation of call chains, customizable printing of call chains, selective printing of call chains, a compressed log file and a startup file called .ccmalloc. The major documentation is found in a file named ccmalloc.cfg. The test files included with the program provide more documentation. nm and gdb are required to get information about symbols and gzip or to compress log files.

NJAMD

NJAMD is, as the author states, “not just another malloc debugger”. As with most memory allocation debuggers, the standard allocation functions are replaced with new ones that perform various checks as memory is used. Specifically, it looks for dynamic buffer over/underflows and detects memory reuse after it is freed. The library built for NJAMD can be LD_PRELOADed, or it can be linked to the program. It creates a large memory buffer on the first memory allocation, 20MB, and it then carves this up as the program needs memory.

NJAMD can be used alone, with a front end or from within gdb. It has a utility that allows postmortem heap analysis. Another feature allows the application being debugged to skip recompilation; simply preload the library. NJAMD also is capable of tracing leaks in library functions that wrap malloc and free, GUI widget allocators and C++ new and delete. Often a memory leak is not discovered immediately but lurks, waiting to strike at the most visible moment. Tracking this down can take a long time. NJAMD has many environment variables that allow setting varying levels of detection. As with most debugging tools, performance can be an issue with NJAMD, so the tool should be used only during development. Deploying with the tool enabled can result in slower systems.

YAMD

YAMD (yet another memory debugger) is another package for trapping the boundaries of allocated blocks of memory. It does this by using the paging mechanism of the processor. Read and write out-of-bound conditions are detected. The detection of the error occurs on the instruction that caused it to happen rather than later, when other accesses occur. The traps are logged with the filename and line number with trace-back information. The trace back is useful because most memory allocation is done through a limited number of routines.

The library emulates the malloc and free calls. Doing this catches many indirect malloc calls, such as those made by strdup. It also catches new and delete actions. If the new and delete operators are overloaded, however, they cannot be caught.

YAMD, like other programs of its type, needs a large amount of virtual memory or swap available to perform its magic. On an embedded system, though, this is typically not available. The earlier suggestion to use this tool on a workstation to do prototype debugging is encouraged here as well. When this debug is done, moving the application to the target can proceed with confidence that most, if not all, memory leaks have been found.

YAMD provides a script, run-yamd, that is used to make the program execute easily. It offers several options to try to recover from certain conditions. A log file can be created when the program being checked performs a core dump. A debugger can be used to debug YAMD-controlled programs. However, problems can arise using a debugger when YAMD is preloaded rather than statically linked with the program.

Valgrind

Valgrind is a relatively new open-source memory debugger for x86-GNU Linux systems. It has more capabilities than earlier tools, but it runs only on x86 hosts. When a program is run under the control of Valgrind, all read and writes to memory, as well as calls to malloc, free, new and delete, are checked.

Valgrind can detect uninitialized memory, memory leaks, passing of uninitialized or unaddressable memory, some misuse of POSIX threads and mismatched use of malloc/free and new/delete actions.

Valgrind also can be used with gdb to catch errors and allow the programmer to use gdb at the point of error. When doing this, the programmer can look for the source of the problem and fix it much sooner. In some cases, a patch can be made and debugging can continue. Valgrind was designed to work on large as well as small applications, including KDE 3, Mozilla, OpenOffice and others.

One feature of Valgrind is its ability to provide details about cache profiling. It can create a detailed simulation of the CPU's L1-D, L1-I and unified L2 cache, and it calculates a cache hit count for every line of the program being traced. Valgrind has a well-written HOWTO with plenty of examples. Its web site contains a lot of information and is easily traversed. Many different combinations of options are available, and it is left to users to determine their favorite combinations.

Valgrind's error display contains the process ID for the program being examined, followed by the description of the error. Addresses are displayed along with line numbers and source filenames. A complete backtrace also is displayed. Valgrind reads a startup file, which can contain instructions to suppress certain error-checking messages. This allows you to focus more on the code at hand rather than pre-existing libraries that cannot be changed.

Valgrind does its checking by running the application in a simulated processor environment. It forces the dynamic linker/loader to load the simulator first, then loads the program and its libraries into the simulator. All the data is collected while the program is running. When the program terminates, all the log data is either displayed or written to log files.

mpatrol

The mpatrol library can be linked with your program to trace and track memory allocations. It was written and runs on several different operating system platforms. One distinct advantage of the library is it has been ported to many different target processors, including MIPS, PowerPC, x86 and by some MontaVista customers, to StrongARM targets.

mpatrol is highly configurable; instead of using the heap, it can be set to allocate memory from a fixed-size static array. It can be built as a static, shared or threadsafe library. It also can be one large object file so it can be linked to the application instead of contained in a library. This functionality provides a great deal of flexibility for the end user.

The code it creates contains replacements for 44 different memory allocation and string functions. Hooks are provided so these routines can be called from within gdb. This allows for debugging of programs that use mpatrol.

Library settings and heap usage can be displayed periodically as the program runs. All the statistics gathered during runtime are displayed at program termination. The program has built-in defaults that can be overridden by environment variables. By changing these environment variables at runtime, it becomes unnecessary to rebuild the library. Tuning of the various tests can be done dynamically. All logging is done to files in the current working directory; these can be overridden to go to stdout and stderr or to other files.

As the program is running, call stack trace-back information can be gathered and logged. If the program and associated libraries are built with debug information about symbols and line numbers, this information can be displayed in the log file.

If at some point the programmer wants to simulate a stress test on a smaller memory footprint, mpatrol can be instructed to limit the memory footprint. This allows for testing conditions that may not be readily available in the lab environment. Stress testing in emulating a customer environment or setting up a harsh test harness is made easier with this feature. In addition, the test program can be made to fail a random set of memory allocations to test error-recovery routines. This ability can be useful for exception handling in C++. Snapshots of the heap can be taken to allow the measuring of high and low watermarks of memory use.

Insure++

The Insure++ product by Parasoft is not GPLed or free software, but it is a good tool for memory leak detection and code coverage, very similar to mpatrol. Insure++ does do more than mpatrol in the area of code coverage and provides tools that collect and display data. Trial copies of the software can be downloaded and tried for a specified time period on non-Linux workstations.

The product installs easily under Linux but is node-locked to the computer on which it is installed. Insure++ comes with a comprehensive set of documentation and several options. The code coverage tool is separate but comes with the initial package.

Insure++ provides a lot of information about the problems it finds. To use Insure++, it is necessary to compile it with the Insure++ front end, which passes it to the normal compiler. This front end instruments the code to use the Insure++ library routines. During the compiler phase, illegal typecasts are detected as well as incorrect parameter passing. Obvious memory corruption errors are reported. During runtime, errors are reported to stderr but can be displayed by a graphical tool. When building an application, either the command line or makefiles can be used, facilitating the building of projects and large applications.

Execution of the program is simple. Insure++ does not require any special commands to execute; the program is run as if it were a normal program. All the debug and error-trapping code is contained in the Insure++ libraries that were linked with the program.

An add-on tool, called Inuse, displays in real time how the program uses memory. It can give an accurate picture of how memory is used, how fragmented it gets and subtle leaks that seem small but could add up over time. I had an experience with a client who found that a particular C++ class was leaking a small amount of memory that, on a workstation, was seen to be quite small. For an embedded system that was expected to be running for months and possibly years, the leak could become quite large. With this tool, the leak was easily traced, found and fixed. Other available tools did not catch this leak.

Code coverage is analyzed by another tool, TCA. As the program is run with Insure++ turned on, data can be collected that, when analyzed by TCA, paints an accurate picture of what code was executed. TCA has a GUI that enhances the display of code coverage.

Resources

Cal Erickson (cal_erickson@mvista.com) currently works for MontaVista Software as a senior Linux consultant. Prior to joining MontaVista, he was a senior support engineer at Mentor Graphics Embedded Software Division. Cal has been in the computing industry for over 30 years, with experience at computer manufacturers and end-user development environments.

Customizing Plone

Reuven M. Lerner

Issue #110, June 2003

Using CMF controls, custom skins and other Zope-based tools, Plone can look and feel however you want.

The market for content management systems (CMS) continues to grow by leaps and bounds, which should come as no surprise. Several years ago, a CEO might have wondered whether to create a web site for his or her company. Nowadays, the question is not whether to put up a site, but how to manage the people who run it and organize the information it contains. A good CMS makes it easy to handle all of this, by taking care of users, groups, permissions and scheduled publication.

But as anyone experienced with mission-critical software knows, software rarely (if ever) handles everything you need out of the box. Companies such as Vignette and Documentum, which sell and service their own content management systems, use this to their advantage and demand consulting and support fees from their customers. CMS customers would prefer to put as much power as possible in their own hands, both to avoid paying consulting fees and to have a greater degree of freedom on a day-to-day basis.

It should come as no surprise that open-source content management solutions allow and even encourage users to modify their own CMS software. But all too often, modifying a system means changing the source code, which not everyone is prepared or able to do. Writing a CMS that is simultaneously powerful, flexible and simple for nonprogrammers to customize has turned out to be a difficult and challenging task, one that is likely to occupy the time of many CMS vendors for years to come.

An increasingly popular open-source CMS, and one that makes it easy to customize a site's look and feel without too much programming, is Plone. Plone does not exist in a vacuum but is built on top of Zope's Content Management Framework (CMF), a set of APIs for creating content management systems. As we saw last month, Plone makes it easy to create a web site that has a number of advanced features built in, including an event calendar, a news archive and a search engine.

But what happens when you want to change things? What if you don't like Plone's default look and feel? Luckily, Plone was designed to be modified in a number of ways and at a number of different levels. This month, we look at some of the ways in which Plone can be modified. Along the way, we learn quite a bit about Zope's CMF, which serves as an excellent segue to next month's CMF tour.

Basic Changes

One of the most basic ways you can customize Plone is by modifying the look and feel. To do this, you need to log in to Plone as a site manager. Assuming you configured Plone in the standard default way, there are two ways to accomplish this. The simpler one is to log in to Plone as a site administrator, giving the user name and password you would use with the web-based Zope management interface (ZMI). Plone normally inherits user names, passwords and roles from its surrounding environment, so you can log in to Plone with those credentials.

Unfortunately, logging in to Plone means you aren't recognized as being logged in by the rest of the Zope site. To avoid this problem and to be able to manage both Zope and Plone, log in to your site at its /manage URL. For example, if your site is located on www.example.com, you can try to log in as the administrator by going to www.example.com/manage.

Once you have entered a Plone site as a site manager, you should see a menubar across the screen, under the main menu of boxes and right over the “you are here” line. Click on the menu item named Plone setup.

Once inside Plone setup, you are asked to answer several questions, such as the name of your Plone site (or portal, as Plone refers to it) and the e-mail address from which system messages should be sent. One of the options allows you to choose a default look, with about a dozen such looks provided in the default installation. A change to the site's default look takes place immediately, allowing you to view and then change whatever look you have chosen.

As the name indicates, the color scheme you choose here is only the default. All users on the system can go into the My Preferences menu and change their own looks. Thus, even if you want your site to have the look of New Mozilla, a user with more conservative tastes can choose something else.

Using CMF Controls

Unfortunately, the majority of Plone cannot be configured from within Plone itself. Rather, you must use the Zope management interface to modify Plone as if it were a simple component of the CMF. This means using a number of CMF controls to change the default Plone look and feel.

I was able to get to these controls by pointing my browser to the folder above the place where I had created my Plone instance. That is, if I access my Plone site at www.example.com/atf, I get to the management interface at www.example.com/manage. The management interface shows all of the objects in the top-level folder, including my Plone instance. Clicking on the Plone object (atf, in our particular case) brings up a long list of objects, most of which begin with the word portal: portal_catalog, portal_calendar, portal_skins, portal_membership and portal_undo, among others. Objects with wrench icons are CMF tools, allowing you to modify part of your Plone instance. For example, the portal_actions tool allows us to modify the different box-like buttons that appear in various places within a Plone site. These include the buttons at the top of each page, such as news and advanced search, and the buttons that appear when a site administrator wants to edit content over the Web. Each action is controlled by seven fields:

- The name of the action that is displayed to the outside world within the box.
- A unique identifier.
- The action that should be taken when the user clicks on a box, which is expressed in TALES format (from the Zope Page Templates system for web templates), normally pointing to a URL.
- The (optional) conditions under which the button should be visible. For example, the Paste button should appear only if there is valid information in the current clipboard, a condition represented in TALES as folder/cb_dataValid.
- The permission a user must have in order to see this button. For example, if an action has View permission, then anyone authorized can see the action box and is able to click on it. By contrast, if an action has Modify folder content permission, then only the users authorized to modify content can see the action's button box.
- The category in which the button should be placed, such as portal_tabs (shown at the top of the screen) or object_tabs (at the top of the screen).
- Finally, we can show or hide actions by clicking and unclicking the check boxes.

Adding, deleting and modifying actions is quite easy. But what if we want to add, remove or shift around the portlets that appear on the right and left sides of a Plone site? These items are known as slots in Plone, and they are customized by modifying the properties of the Plone instance itself. That is, you must click on the Plone instance you created (atf in this case) and then on the properties tab at the top of the page.

The left_slots and right_slots properties determine what is displayed on the side. If you have recently installed a Plone instance, you immediately will notice that each slot contains more lines than slots displayed at the top of the screen. This is because a portlet is displayed only if there is something to display. If, for example, no current events are defined, Plone does not display your events portlet at all. The portlet named in the third line of left_slots thus might appear first, second or third, depending on whether the first and second contain any current content.

On my own site, I was able to move the event listing to the left side and the calendar to the right side (and remove the login portlet altogether), simply by modifying the definitions of the left_side and right_side properties. Making these sorts of changes is both easy and quick, and they allow you to include only the functions you want on your site.

Finally, click on the portal properties link that ZMI displays within your Plone instance. This has a Plone icon rather than a wrench icon, indicating that this tool is specific to Plone and not generally available in the CMF. Clicking on this brings up a list of four different property lists (form_properties, navigation_properties, navtree_properties and site_properties), each of which allows us to change some of the properties on our Plone site.

If you click on site_properties, the list might seem strangely familiar. That's because site_properties lists many of the same settings we saw earlier. Plone itself exposes only the most common and necessary settings; more complex and advanced settings are available through the ZMI. It doesn't really matter whether you change the date format, for example, from the ZMI or the Plone settings page; in either case, the site immediately changes to reflect the new value.

Custom Skins

We can modify quite a bit of our Plone site by changing property definitions and using the ZMI. But if you really want to change your Plone site, you need to modify the page templates (ZPTs) that come with the system. This is easier said than done. The default ZPTs are stored in the filesystem, in such directories as $ZOPE_ROOT/lib/python/Products/CMFDefault/skins (for CMF content) and $ZOPE_ROOT/lib/python/Products/CMFPlone/skins/ (for Plone content). Modifying the skins within these directories affects all Plone instances, which is not what you want.

Plone takes this possibility into account and allows you to copy one or more ZPTs into Zope's object database (ZODB), where you can edit it as you would any other ZPT. For example, from within the ZMI, enter the portal_skins tool and then the plone_templates folder within portal_skins. plone_templates looks like a normal Zope folder (aside from the different icon), but it reflects the contents of files on disk rather than those within ZODB. plone_templates contains the ZPTs for most of the pages you see within Plone. The ui_slots folder within plone_templates contains ZPTs that determine how the portlets look.

If you want to modify the header that appears at the top of each page within your Plone site, you can click on the header icon. This brings you to a page that lets you view, but not modify, the header page template. In order to modify the header, you must export it to the custom folder, which exists only within ZODB. Click on Customize, and you can see that the URL has hung within the ZMI, putting you now within portal_skins/custom rather than within portal_skins/ plone_templates. This custom folder is the central repository for all customized templates, and you can edit them as you would any other ZPT on the system. Because the custom folder is specific to each instance of Plone, you can be sure that any changes you make affect only what you are working on.

If nothing else, it is worth looking through the header and footer page templates to see the great deal of customization work that was done, largely in JavaScript, to make sure that Plone would work on different browsers. Given that every browser has somewhat different support for CSS and JavaScript, it is rather impressive to see the great deal of effort the Plone authors put into keeping things as level as possible.

Of course, this means you might be in for a surprise or two. My father, who used Netscape 4 until quite recently, complimented me on my new site and on the fact that it chastised him for not using a more modern browser. Because I have long been using the latest versions of Mozilla and Galeon, I hadn't ever seen this message; it never occurred to me that one would appear. The Web would be a better place if every application were so clever and conscientious about checking cross-platform compatibility.

Conclusion

Plone is probably the best-known and most popular application written with Zope's CMF, one that is powerful and easy to customize. Between Plone-specific customization screens, changes that we can make with the ZMI and modifications to the page templates by importing them into the custom folder in ZODB, we can change things in a great many ways. We also can add new custom skins to Plone, contributing to the already interesting and varied options that come with the distribution.

Of course, Plone is only one application built using Zope's CMF. Next month, we will peel away another abstraction layer, looking at the CMF directly and seeing what sorts of applications we can create with it. As you will see, there is good reason why the CMF is attracting a great deal of attention in the Zope community, as well as from Zope Corporation itself.

Resources

Reuven M. Lerner (reuven@lerner.co.il) is a consultant specializing in open-source web/database technologies. He and his wife, Shira, recently celebrated the birth of their second daughter, Shikma Bruria. Reuven's book Core Perl was published by Prentice Hall in early 2002, and a second book about open-source web technologies will be published by Apress in 2003.

Using Firewall Builder, Part II

Mick Bauer

Issue #110, June 2003

Configure bastion host and firewall iptables policies so you can see exactly what the security policy is.

Last month we used Firewall Builder to create a set of reusable objects for iptables policies. In this month's column, I show you how to use Firewall Builder to create two such rule sets: one for a bastion host that needs to defend itself and another for a firewall that needs to defend entire networks.

Local Rules on a Bastion Host

Let's consider the bastion host scenario first. A common misconception about Netfilter/iptables, and about packet filtering in general, is that packet inspection is strictly a function of firewalls. In-depth defense, however, dictates that it's foolish to put all your security eggs in one basket. Although you must use a carefully configured and monitored firewall to protect all your internet-connected hosts, those hosts also should be able to defend themselves, especially the bastion hosts on which you host publicly accessible services, such as FTP and WWW.

If, for example, your public web server runs Linux 2.4, it follows that you should configure its local Netfilter rules to provide an extra level of defense in case a clever attacker subverts or otherwise gets around your enterprise firewall. If your server runs a pre-2.4 kernel, you need to use ipchains rather than Netfilter/iptables. You also need to find a contributed ipchains compiler plugin for Firewall Builder to build your scripts.

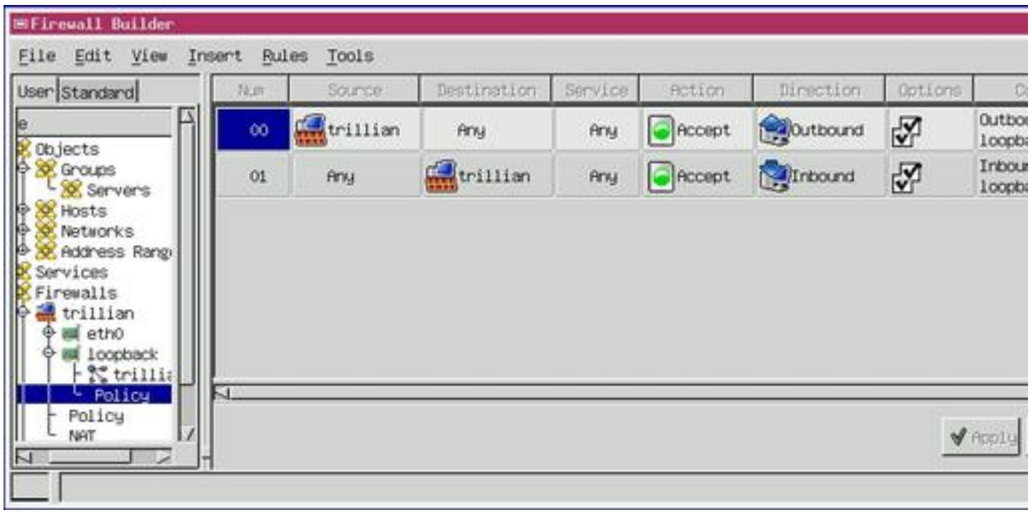
Loopback Rules

Step one for creating any firewall rule base, even for a bastion host, is to give free rein to the local loopback interface. Loopback is used for certain transactions between local processes and dæmons. Without loopback-allowing rules, things like name-service caching and SSH port forwarding break when you run the iptables script.

Suppose you've got a web server to harden, named Trillian. You've installed Firewall Builder on your administrative workstation; remember, we avoid running the X Window System and therefore X-based applications on bastion hosts. You've subsequently created some objects that describe hosts, networks and groups in your environment, plus a firewall object for Trillian, complete with a loopback-interface definition. In other words, you've done the things I described in last month's column.

You need two rules for Trillian's loopback interface: one that allows all traffic leaving the loopback interface and one that allows everything coming in to it. Follow these steps to create two such rules (Figure 1):

1. Beneath and to the right of your firewall's loopback interface sub-object, on the left-hand side of the Firewall Builder screen (in Figure 1, this is named loopback), select the loopback interface's policy, which should be empty.
2. In the Rules menu, select Append rule at the bottom. A blank rule appears in the right-hand half of the window.
3. Drag the firewall icon next to the name Trillian into the blank rule's Source field. Be sure to wait until the cursor changes into a plus (+) before releasing the mouse button.
4. Right-click in the new rule's Action field and select Accept from the menu.
5. Right-click in the rule's Direction field and select Outbound.
6. Right-click on the paper and pencil icon in the rule's Options field and select Turn logging OFF.
7. Right-click again in the rule's Options field and select Modify options. In the resulting window, check the box near the bottom of the window, which disables stateful inspection. We don't need to waste CPU overhead on state tracking for loopback traffic.
8. Optionally, right-click in the new rule's Comment field and select Edit Comment if you wish to write a brief reminder of the rule's purpose, perhaps “allow loopback outbound”.



Num	Source	Destination	Service	Action	Direction	Options	C
00	trillian	Any	Any	Accept	Outbound	✓	Outbo loopb
01	Any	trillian	Any	Accept	Inbound	✓	Inbou loopb

Figure 1. Loopback Interface Rules

To create the second rule in Figure 1, repeat steps 2 through 8. In step 3, however, drag Trillian's icon into the new rule's Destination field rather than its source. In step 5, set the direction to Inbound.

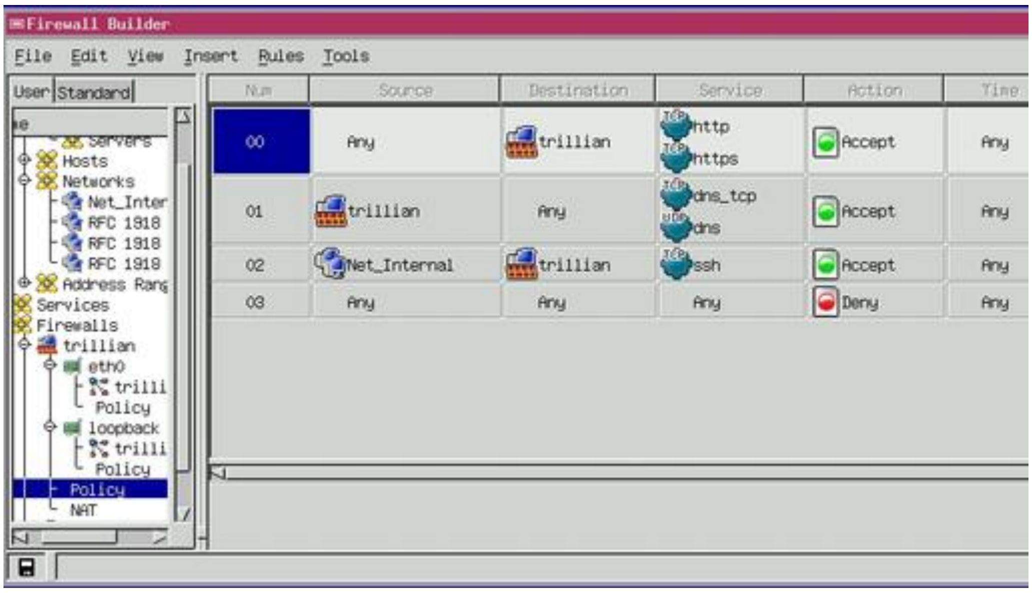
How, you may ask, do these rules work? First, you should understand that they apply only to the loopback interface. It's possible to create rules specific to any interface, rules that are parsed before your firewall's global policy. Although we used Trillian as the source and destination, respectively, of our two loopback rules, this doesn't mean that the rules match packets with particular IP addresses, that is, Trillian's. They'll match any packets leaving or entering the loopback interface.

This leads me to my last point about loopback rules. It may seem counterintuitive to use two rules referencing the firewall object rather than one rule that says any source to any destination should be accepted. But in my own tests, the single-rule approach caused Firewall Builder to write its loopback rules for the FORWARD chain rather than for INPUT and OUTPUT, which counterproductively killed loopback on my test system. Changing to separate loopback in and loopback out rules fixed the problem. Don't worry; this is the only time I've seen Firewall Builder choose the wrong chain for its rules. At that, it did so only for single-homed hosts, not multi-interfaced firewalls.

Bastion Host Policy

Once your bastion host's loopback needs have been attended to, turn your attention to its global policy. This requires a little thought. You want Netfilter to provide a meaningful amount of protection but not at the expense of desired functionality.

Our example host, Trillian, is a web server, so we want to allow other hosts to access it with HTTP and HTTPS. We also want to allow Trillian to perform DNS lookups for coherent logging. In addition, some sort of administrative connectivity should be allowed. The tool of choice for this purpose is SSH, so we'll also allow inbound SSH connections but only from our internal network. Figure 2 shows such a policy as defined for Trillian.



Num	Source	Destination	Service	Action	Time
00	Any	trillian	http https	Accept	Any
01	trillian	Any	dns_tcp dns	Accept	Any
02	Net_Internal	trillian	ssh	Accept	Any
03	Any	Any	Any	Deny	Any

Figure 2. Policy for a Bastion Host

I'll spare you a blow-by-blow description of how I created every single rule, but several things are worth noting. First, in the object hierarchy on the left-hand side of the window, you can see that I had selected the global Policy object in the hierarchical level directly below Trillian, rather than either of the interface-specific policy objects.

I also referenced not only the Trillian object, but also a network object named Net_Internal, which is the example network object from last month's column. This object refers to an entire network's worth of IP addresses, 192.168.111.0 to be exact. Whereas rule 02 uses a single IP address (Trillian's) as its source IP address, rule 03 matches packets whose source IP address is any of the entire range, 192.168.111.1-192.168.111.254.

Another important tip for rule building is to get at Firewall Builder's handy prebuilt service objects, click the Standard tab on the left-hand side of the window. Be careful, though; if you do anything *besides* drag a service object (for example, dns_tcp) into your rules, the rules display on the right-hand part of the window is replaced with information about whatever you've selected.

In other words, if you're working on a policy, you can click on the Standard tab, click on the + (expand) and - (collapse) icons in the hierarchy window and click and drag service objects from it, all without changing the mode of the right-hand part of the window. But if you simply select a service object or category in the Standard hierarchy (by clicking on it once without dragging), that object's properties are displayed on the right. You have to go back to the User tab and reselect your firewall's global policy to display your rules again. You do not lose any data, but this can be inconvenient and unsettling if you aren't expecting it.

A more substantial observation is that in all of these rules, I left stateful inspection turned on. I skipped step 7 from our loopback-rules procedure. Normally, we want the kernel to keep state information on network transactions; this is why we can describe most transactions with a single bidirectional rule rather than with two unidirectional rules. For example, thanks to stateful inspection, whenever a transaction matches rule 02 from Figure 2, which allows inbound SSH traffic from hosts on the internal network, Trillian's kernel matches not only those inbound SSH packets, but also the SSH packets that Trillian sends back out in reply. Had I turned off stateful inspection for rule 02, I'd need another rule allowing all packets originating from TCP port 22 on Trillian to accommodate those replies.

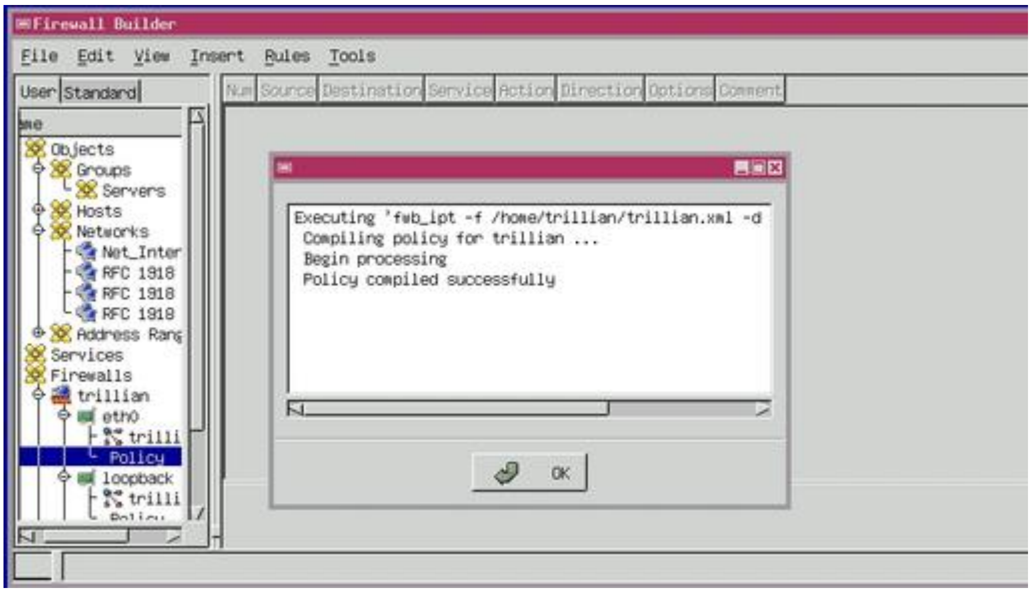
Finally, all rules but the last one have logging turned off, as described in step 6 of our loopback-rules procedure. Most people don't find it a useful or justifiable use of disk space or I/O overhead to log every packet their firewall rules process. Personally, I tend to focus on dropped packets and forego logging on allow rules. Thus, the sample rules in Figure 2 end with a cleanup rule at the bottom that explicitly drops any packet not matching the other rules or the rules in any interface-specific policies such as the loopback policy.

This rule's sole purpose in life is for logging. Firewall Builder automatically sets the default policy for all my iptables chains to DROP, but these dropped-by-default packets aren't logged unless you tell Netfilter to do so.

An experimental dropped-table patch is available for Netfilter that allows automatic logging of all dropped packets, but I recommend you wait for this code to stabilize before going out of your way to compile it into your kernel. If you can't wait for some reason, you can access this feature from Firewall Builder by selecting your firewall object, clicking its Firewall Properties tab and checking the box next to Log all dropped packets. For more information on the dropped-table patch, see www.netfilter.org/documentation/pomlist/pom-summary.html.

Compiling and Installing the Policy

After you've finished your firewall policy, you need to convert it to an actual iptables script. To do so, first make sure that on the screen's left-hand hierarchy view your firewall's object, its global policy or one of its interface policies is selected—it doesn't matter which. Then, pull down the Rules menu and select Compile. Figure 3 shows the result.



```
Executing 'fwb_ipt -f /home/trillian/trillian.xml -d
 Compiling policy for trillian ...
 Begin processing
 Policy compiled successfully
```

Figure 3. Compiling the Policy

Upon successful policy compiling, Firewall Builder writes a file whose name consists of the name of the firewall object whose policy you compiled and the suffix .fw. The example script we produced, trillian.fw, is shown in Listing 1. Listing 1 has been modified slightly due to space requirements, and some housekeeping material has been removed. All of the actual rules mentioned in the article are present.

Listing 1. trillian.fw

This script now can be copied over manually to Trillian and run as is, or it can be converted manually into a startup script appropriate to Trillian's Linux distribution, for example, a standard Red Hat 7.3 startup script. Easier still, you can copy it over automatically and activate it with a Firewall Builder installer script such as fwb_install, available under contrib at sourceforge.net/project/showfiles.php?group_id=5314.

The latter in particular is an elegant and simple way to copy and activate your firewall scripts securely; fwb_install uses **scp** to copy the script to /etc/firewall on the remote host and then **ssh** to execute the script remotely. If you've downloaded fwb_install somewhere on your Firewall Builder system, you can configure Firewall Builder to use it from within each firewall object's Compile/Install properties.

Be sure to tweak fwb_install manually to match your system settings and to set up SSH keys for fwb_install to use. Once you've set this up, all you need to do to install your policies after compiling them is pull down the Rules menu and select Install.

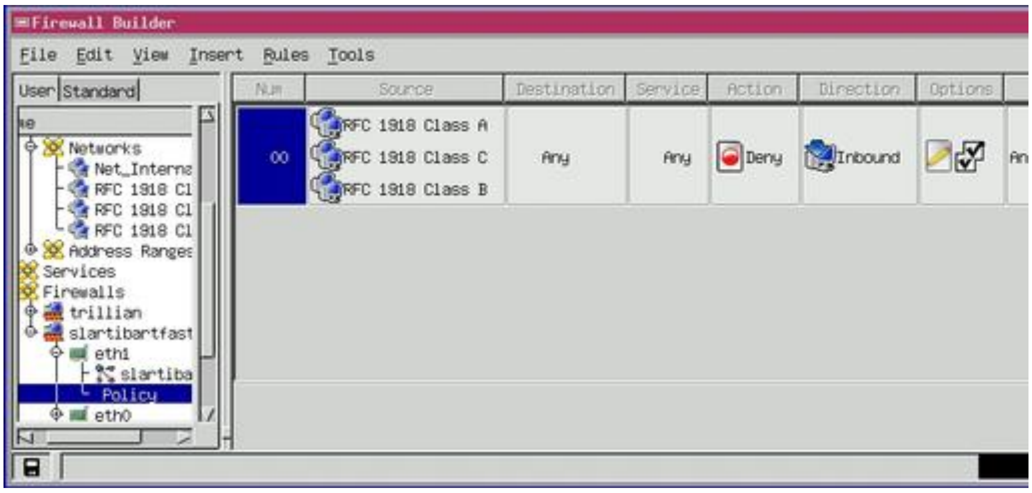
As handy as fwb_install is, you'll want a startup script in place on your target system that also executes the firewall script on startup and after any reboot. Otherwise, the system is vulnerable between each startup or reboot and the next time you execute Install from within Firewall Builder. It's easy to copy and adapt existing scripts in your system's /etc/init.d directory.

Policy for a Real Firewall

I've devoted most of this column to the bastion host example, but building a policy for a multihomed (multi-interfaced) network firewall is quite similar. Create loopback policies, create anti-spoofing policies for the other interfaces, create a global policy, compile the policy and install it.

The big differences have to do with the fact that a firewall, unlike a server, has multiple network interfaces. Because a single-interfaced system receives all packets except loopback at one physical point, it can't distinguish spoofed packets from legitimate packets; it must take each packet's source-IP address at face value. But a multihomed system can distinguish easily between packets that truly originate from local networks and packets that arrive from the Internet but have forged source-IP addresses matching a local or trusted network.

For instance, our example internal network is numbered 192.168.111.0 (subnet mask 255.255.255.0). If we have a firewall named Slartibartfast between this network and the rest of the world, we can use anti-spoofing rules to tell Slartibartfast to drop any packet immediately from any interface other than the one facing our internal network, if that packet has a source IP beginning with 192.168.111. Such a packet is obviously spoofed. Figure 4 shows Slartibartfast's anti-spoofing rule.



Num	Source	Destination	Service	Action	Direction	Options
00	RFC 1918 Class A RFC 1918 Class C RFC 1918 Class B	Any	Any	Deny	Inbound	

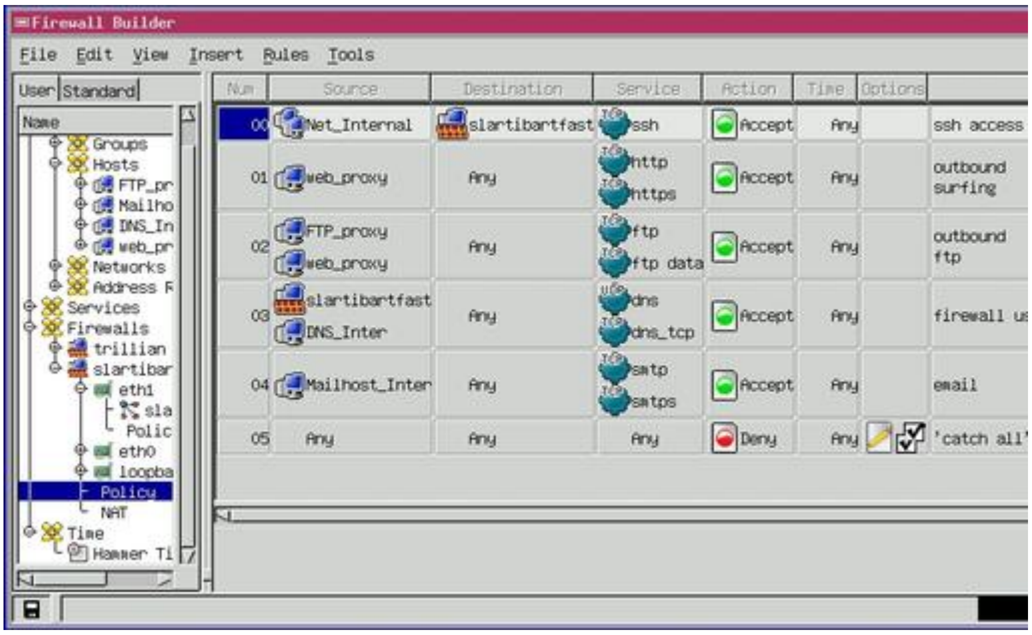
Figure 4. Anti-Spoofing Firewall Rules

Before I made this rule, I created several network objects that refer to the reserved IP address spaces defined in RFC 1918, “Address Allocation for Private Internets”. RFC 1918 address spaces are for use only within an organization and can't be routed over the Internet, so any internet firewall should consider inbound packets bearing such addresses to be spoofed, which is precisely what the rule in Figure 4 does. Because my RFC 1918 Class C object expands to 192.168.0.0, subnet mask 255.255.0.0 and my internal network address is 192.168.111.0 (part of RFC 1918 address space), it wasn't necessary to include my Net_Internal object in this rule.

By the way, if you're not familiar with RFC 1918, my RFC 1918 Class A object refers to 10.0.0.0, subnet mask 255.0.0.0, and RFC 1918 Class B is 172.16.0.0, subnet mask 255.240.0.0.

Global Rules

Figure 5 shows Slartibartfast's global policy; because this article is already too long I won't explain it in-depth. But the whole point of Firewall Builder is to display firewall rules in an easy-to-read format, so Figure 5 should be self-explanatory.



Num	Source	Destination	Service	Action	Time	Options	
00	Net_Internal	slartibartfast	ssh	Accept	Any		ssh access
01	web_proxy	Any	http https	Accept	Any		outbound surfing
02	FTP_proxy web_proxy	Any	ftp ftp data	Accept	Any		outbound ftp
03	slartibartfast DNS_Inter	Any	dns dns_tcp	Accept	Any		firewall us
04	Mailhost_Inter	Any	smtp smtps	Accept	Any		email
05	Any	Any	Any	Deny	Any	✓	'catch all'

Figure 5. Global Policy for a Network Firewall

Speaking of self-explanatory, did I mention that all rules, whether loopback, anti-spoofing or global, can be generated quickly and automatically using Firewall Builder's policy druid? You can run it by selecting a firewall object, pulling down the Rules menu and selecting Help me build a firewall policy.

Don't get too irked at me for not mentioning this until after making you slog through all my instructions on building policies the hard way. Firewall rules are too important to trust entirely to druids. Hopefully, you now can understand and tweak or even correct the rules Firewall Builder generates for you. Regardless of how you build your policies, I hope you find Firewall Builder as useful as I have.

email: mick@visi.com

Mick Bauer, CISSP, is *Linux Journal*'s security editor and an IS security consultant for Upstream Solutions LLC in Minneapolis, Minnesota. Mick spends his copious free time chasing little kids (strictly his own) and playing music, sometimes simultaneously. Mick is author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

Archive Index Issue Table of Contents

Advanced search

Click-N-Run: an Easier Future for Customers?

Doc Searls

Issue #110, June 2003

The remaining barriers to computer ease of use aren't technical but results of the way vendors distirbute their software.

In February 2003, I took the train from my home in Santa Barbara, California to the Desktop Linux Summit in San Diego, California. I paid around $50 for a round-trip business-class seat, which was cheaper than gas for a car, cushier than a first-class airplane seat and equipped with AC power for my laptop. Most of the trip was through paradise: emerald-green farmland, red-rock canyons and suburbs bounded by enormous mountains capped in snow.

But the most interesting part of the trip, at least for me, was the huge industrial district that starts in the San Fernando Valley, follows the concrete trough of the Los Angeles river through downtown and then spreads across several hundred square miles of Los Angeles and Orange county flatlands. Sliding past the window are endless yards of lumber, fabricated metal and piping of all sizes. To me, it's a living model of the computer industry's future—one in which commodities are considered good and necessary things.

Throughout its short history, the computer industry has treated commodity software as undesirable stuff with low margins or no margin at all. Yet, mature industries thrive on commodities. Lumber and mining companies, metal fabricators and blow-molded plastic manufacturers are all in commodity businesses. When the software industry grows up, it will come to a new understanding of commodity value, especially of the commodity we call free software—that's not “free” as in beer or speech, but free as in limestone, wood and silicon. Those are all elemental substances, freely produced by nature. In a similar manner, free software is produced by human nature.

It's hard to see the economic value of free software in an industry still dominated by a giant mutant company that leverages its monopoly position to extract 85% margins from customers who have little choice in the matter.

I believe I witnessed the dawn of a mature software industry at the Desktop Linux Summit in San Diego.

I wasn't quite sure what to expect before I got there. The show's planning was beset by problems. There was a communication breakdown of some kind between groups setting up the show. Some speakers and companies dropped out. The one thing everybody agreed on was that the show was less about Linux than Lindows. And after I got a chance to see what Lindows.com was up to, I decided that was a fine thing.

If the Lindows folks succeed at their mission, which I think they have a very good chance of doing, Linux on the Desktop (LOTD) will finally become a reality and not merely a nearly empty hole between servers and embedded devices, where Linux is clearly well on the road to World Domination.

While Dell, Gateway, HP and IBM all sit on their hands and wait for the market to scream at them to get serious about Linux desktops, Lindows.com does the hard work of actually making the market—not only with a cool new distribution, but with a business model that creates a win-win market for both free and proprietary software.

Lindows.com is the creation of Michael Robertson, the founder and former president and CEO of MP3.com. Robertson easily could have retired to a life of leisure and portfolio management after selling MP3.com to Vivendi. Instead, he decided to do something no venture capitalist would ever fund: compete straight up with Microsoft in the operating system business.

Where others look at Microsoft's success as a problem, Robertson sees it as pure opportunity. He's kind of twisted that way. Here's what he told me when I interviewed him at the show:

> Right now we're watching hardware vendors duke it out over 7% gross margins. Terrible business. Meanwhile, Microsoft is taking all the high-margin software profits for themselves. So you have this cutthroat business making the razor handles, and Microsoft taking all the profits making the blades.
>
> In the future that's going to change. The hardware companies will partner with companies like Lindows.com, which are interested in making the relationship work for both sides. They'll say, “Hey, I'm willing to invest in factories and companies that will put together PCs and market and sell and support them. But I want partners who will get me more than just margins on the raw hardware. Any razor blades you're selling to your customers—virus protection, web filtering, e-mail service, whatever—I want a piece of that.” And we're here to give that to them.

> I'm not just talking “add an SKU to your on-line web store.” We're already doing that with Walmart.com. I'm talking about really marketing in a big way, one that relieves these big hardware companies of their painful situation. They're all selling a commodity, which means the leanest and meanest low-cost provider will win. That's Dell. The other guys—the losers of today—are going to say, “We need to change the model a little bit, by looking for a better deal from partners with a better model for selling what runs on their OS.”

That better model is called Click-N-Run, which is built into LindowsOS, which is built on Debian GNU/Linux and KDE.

Think of LindowsOS as the way you would set up a computer for maximal user convenience, if you didn't have legacy software licensing issues to worry about. Robertson's goal is ease of use that's plainly superior to Windows, especially when it comes to the hard part: adding new software. That's what Click-N-Run is all about. You want the drivers for your new HP printer? Click on the download link and LindowsOS runs the installer, which brings up the Konqueror browser. After that, it's a quick direction-following exercise. Soon you're printing on your LaserJet; and thanks to the Debian dependency model, adding the printer doesn't break something else. Want KStars, a desktop planetarium? No sweat. Click on it, and it's yours. GIMP? Sure. Click and run.

But those are all free software. Click-N-Run mixes proprietary software into the same aisles. If you want *Marble Blast*, that'll be $9.95 US. There's also an annual subscription fee. What that fee buys is something you've never had with Microsoft or Apple: a real relationship with your OS supplier.

The old software industry model was all about manufacturing. You make a product, release it to the supply chain and measure success by quarterly sales results. If you have any kind of a relationship with the final customer, it's remote and indirect. If you seek a real relationship, it isn't always a mutual or trusting one. Mostly you want to make sure the customer isn't using a “pirated” copy of your product.

With Click-N-Run, the relationship goes both ways. And if it's a trusting relationship, Lindows.com can intermediate with the makers of the free and proprietary software it supplies. Rather than saying “How did version 1.04 do with high-income East Coast customers?” they can say “How many people download software in the games category?” Or, “How many are downloading GIMP plugins?” If 10,000 people download AbiWord and it becomes clear that a significant number of them want AbiWord to add a feature, maybe Lindows.com or one of its partners will fund development of that feature, even though they won't make money on the program itself.

My point is with Click-N-Run, Lindows.com has the means in place to become the most customer-responsive software company in the world, to do it in a way that finally makes software easy for the customer and helps everybody win.

How close have they come to some ideal here? Lindows.com also introduced a new $799 US laptop at the show that only weighs 2.9 pounds. One of the attendees called it “a sysadmin's PDA”, which it is. But my six-year-old kid also fell in love with it. He's a reader now and figured out Click-N-Run in about ten seconds. Then he downloaded a bunch of games and played them on the train all the way home. We have three platforms running in our household. Guess which one the kid likes best now?

Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal*.

Re-energizing the Stunted PC Revolution

Michael L. Robertson

Issue #110, June 2003

Getting Linux into stores is a dog fight every step of the way.

We're in the midst of a stunted PC revolution. I'm sure some would challenge this characterization given the enormous impact the PC has had on how we communicate, transact and entertain. But as I look around, I wonder why I don't see more computers in more places. Why doesn't every student have a computer? Why don't five-year-olds get their own computers the same as they get bicycles? Why doesn't every hotel room, conference room, school room and bedroom have a computer? Why do only 70% of Americans have access to PCs? The big answer, of course, is cost. PCs are too expensive.

For many consumers, the most expensive component of desktop computing is not the cost of the machine, but the price of the software. The average PC sells for $700 US and is dropping by more than a hundred dollars each year—and that's the average. Many people pay much less. But software has not shown the radical price decreases that hardware has, in spite of the near-zero reproduction costs of software. The lack of meaningful competition has allowed one company to operate with 85% gross margins, which is great for their shareholders but raises the cost of computing by billions of dollars and prevents the full benefit of the PC from reaching all corners of our society. Antitrust litigation and government programs have attempted to throttle the costs and close the digital divide with little results.

Desktop Linux is going to be the force that makes software affordable and energizes the PC business into the next wave, which will make the last wave—the proliferation of the graphical user interface—seem tame. Software prices will decline radically. Instead of eating up the majority of a computing budget, software should consume a much smaller portion, about what you'd pay for a hard disk and RAM.

Several challenges remain to be met before desktop Linux can have a significant impact on software costs, and most of these challenges are not technical. When I have a chance to put users on well-configured LindowsOS machines with Netscape, StarOffice and KDE, their typical reaction is one of shock. They're shocked that Linux can perform ably on the desktop. There's still some work to be done to bring together an easy-to-use OS and wide-ranging applications that together make a capable end-user experience. But the LindowsOS Click-N-Run system, I believe, is filling that void.

The nontechnical areas are where Linux needs to see growth before we'll witness widespread consumer adoption. And the challenge here is that the entrenched monopolist's war chest ensures a dog fight as every step of the essential retail ecosystem is built. It's challenging to recruit OEMs to build Linux computers when their number-one profit determiner is their existing economic relationship with the aforementioned monopolist. More than one of the top ten OEM and chip companies have said they cannot do desktop Linux until they "clear" it first, and they don't mean clear it with their internal management. Once you've got the hardware vendors on board, securing a retailer to carry the product is the next critical step. Many have questions about demand, sales training and customer support, but those questions are answered favorably once the cash register starts ringing.

If on-line sales of desktop Linux at outlets like Walmart.com and Tigerdirect are any indication, there's strong pent-up demand, which means the early retail adopters especially will see strong customer sales. The Brick, Canada's largest electronics retailer, has begun stocking computers with Linux pre-installed in more than 50 of their outlets. This is a dramatic development because it's the first time we've seen choice on computer store shelves in 15 years.

With the retail ecosystem required to bring desktop Linux to the masses showing promising developments, the final hurdle is education. Most PC consumers have known only Microsoft. They speak only Microsoft's language of computing. Microsoft even has attempted to rewrite history by positioning themselves as the innovator and owner of even basic terms like "windows". What's undeniable is that Microsoft file formats and protocols are the *de facto* standards, and all desktop products must interact with them seamlessly. Desktop Linux products today do a surprisingly great job of this, but consumers don't know about it. It's here where much work remains to be done.

Conferences, evangelists, training classes, stocked Linux aisles in stores, "Lin" labeling alongside "Mac" and "Win" on peripherals and much more are all needed to accelerate the education process. Ultimately, the cost advantage of Linux-based desktops will drive adoption in business, homes and schools, but education will dictate the growth rate. I look forward to the day when we see consumers benefiting from affordable Linux software, because we will see computers positively affecting our lives far beyond where we're at today. It's coming, but if you encourage a friend to try desktop Linux, we'll get there much sooner.

email: cheryl@lindows.com

Michael L. Robertson is the founder and chief executive officer of Lindows.com.

Archive Index Issue Table of Contents

Advanced search

The Sharp Zaurus SL-C700

Guylhem Aznar

Issue #110, June 2003

The audiophile will enjoy the excellent sound output; the typist will find the keyboard pleasurable for a device so tiny; and the graphics fan will be seduced by the crystal clear screen.

After the success of the Zaurus SL-5500 and the SL-5600, which was a revamped SL-5500 featuring a faster CPU, a better battery and a microphone, many people expected the next Zaurus would be as innovative as the SL-5500 was when it was first introduced. The Zaurus SL-C700 seems to satisfy this expectation.

The SL-C700 device is smaller than the previous ones and feels much more polished. When I received it, my fiancée immediately wanted to play with it, though she hadn't found the older ones very attractive. The SL-C700, however, comes with some disadvantages, which easily are explained by the lack of official support for it from Sharp outside Japan. Its range of use is quite impaired by the lack of support.

Specifications

Featuring an Intel XScale PXA 250 400MHz processor, 64MB of Flash, 32MB of RAM, a 65,536 color 640 × 480 VGA screen, an IrDA port, a USB port, both a CompactFlash and a MMC-SD port, a stereo audio jack, a jogdial-like wheel, a comfortable keyboard and a screen that can move between the traditional computer-style landscape mode and the traditional PDA-style language mode, this is the most-advanced PDA currently available.

The audiophile will enjoy the excellent sound output; the typist will find the keyboard pleasurable for a device so tiny; and the graphics fan will be seduced by the crystal-clear screen. Personally, I was much more impressed by the soft and luxurious feeling of the gray case. Although the earlier devices felt a bit like plastic toys, this one proclaims its style with its appearance. The inclusion of

LEDs on the left side of the hinge for quick access to battery charge and e-mail status, plus a convenient wheel with OK and Cancel buttons on the side of the machine, showcase the user-friendly design.

One missing feature may be an included microphone. Due to the poor quality of typical PDA microphones and the availability of hands-free earbuds and microphones, this is not a real issue. However, Sharp has listened to customer feedback and did not forget to include an internal speaker. It may not seem natural to include a speaker without also providing a microphone, but its presence allows the SL-C700 to communicate directly to its user through customized tones when an e-mail is received, when the device is powered on or for alarms.

Ordering

The SL-C700 is officially available only in Japan. Dynamism.com, though, sells the device worldwide for $699 US, plus custom duties. Because it is a big success in Japan, where it is always back-ordered, expect long delays before receiving one. It took two months to have my order processed, and the FedEx parcel arrived after I was no longer expecting it.

Dynamism.com hired a coder well known in the Zaurus community to localize the whole device to English. After the default Japanese applications had been removed, I couldn't spot a single Japanese word except in the Favorites bookmarks. Even with full-English support, two dangerous keys featuring Japanese letters stand between Fn and space. If you press them, you are put into Japanese mode. The English fonts used prevent the application from displaying Japanese, though, so only little squares underlined in red by the Japanese on-the-fly spell checker appear. It took me some time to realize these keys had to be pressed again until an A appeared in the titlebar. When in Japanese input mode, the letter A is replaced by a Japanese character.

First Impressions

The keyboard on the SL-C700 has keys that look big and soft, but a disturbing beep accompanies every key press. The beep can be disabled with a simple click on the audio icon on the taskbar, which brings up the Audio Setting menu. Then, the keys emit soft clicks with each key press, much more discreet than the loud beep.

The quality of the SL-C700 screen is impressive; I had never seen such a screen before. The picture is so sharp you cannot see any individual pixels. The colors and the display are so bright you can use the device outdoors with the backlight on. The SL-5x00 series screens were impressive—much better than the IPAQ, for example—but could not compete with the most recent Sony Clié

LCD screens. The SL-C700 screen outperforms the competition and will bring disappointment to previously proud Clié owners.

It takes four minutes to boot the first time the SL-C700 is powered up. During boot up, the SL-C700 told me it needed power on a pop-up window. I found a matchbox-sized charger in the box, with a Japanese/American plug. With a plug adapter, it worked like a charm in the European 230V output, even if it was labeled as only 100V. Dynamism.com comments that no power adapter is needed. The small size of the charger is a big plus; it makes the short battery life, around three hours, less important because it can be carried everywhere.

Logging In

When the boot is finally over, the setup screen welcomes you. After calibrating the screen sensors with five clicks, you must enter the local date, time and time zone. New York is included by default, along with Tokyo and other important cities.

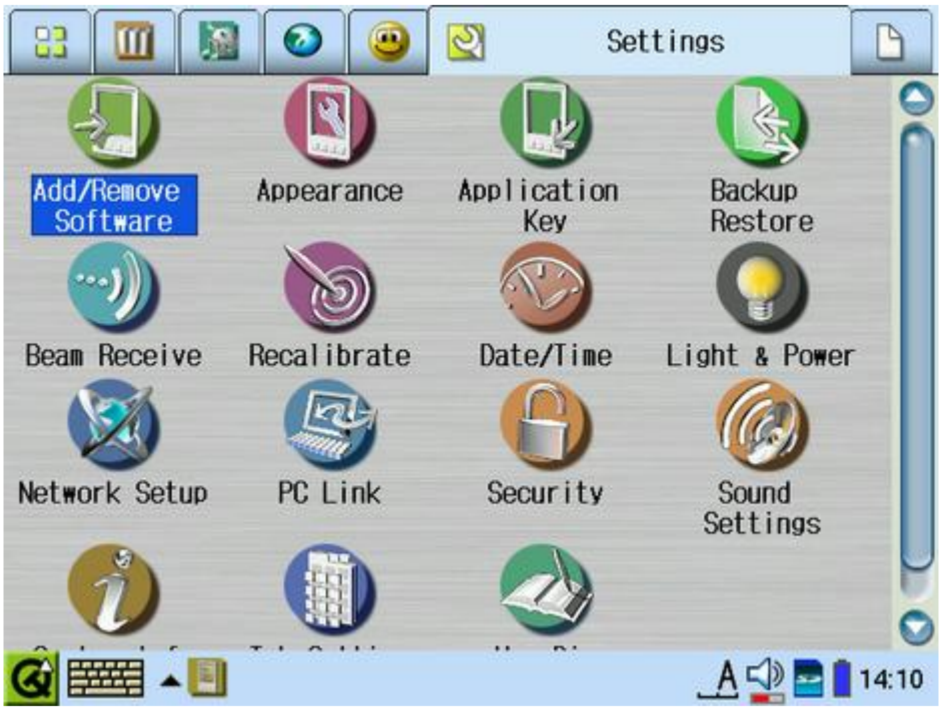
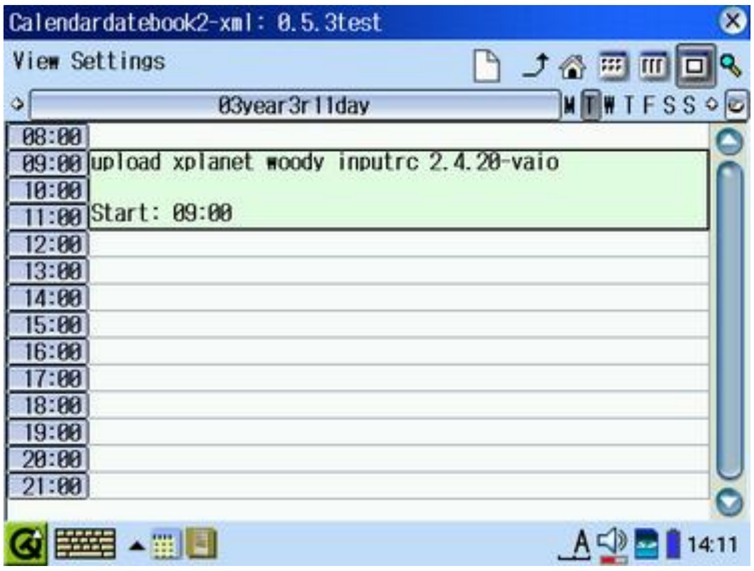


Figure 1. The launcher features big icons.

Next, the default launcher pops up. Many applications are installed by default. I tried the personal information manager (PIM) first. I was disappointed by the small font used in the datebook and address book until I realized this font is used in every other application. Comparing the screen in front of me to the Japanese user manual screenshots, I realized it was smaller than what it should have been. An e-mail to Dynamism.com technical support confirmed that the English localization of the device had the unexpected side effect of implementing a different font by default. It has been reported to Dynamism.com, so the devices now should ship with a normal font.

Another problem is the strange alphabetical ordering in the address book—A, Ka, Sa and so on. This is Japanese alphabetical order, which is not exactly suited for English speakers. Another problem is the lack of XML support for former SL-5x00 owners. Someone at Sharp must have had the clever idea of removing the industry-standard XML format in favor of an obscure binary data format, where data is stored in ~/Applications/dtm with strange names. Personally, I liked the old ~/Applications/Datebook and ~/Applications/Addressbook, with XML files that could be imported or exported into other applications easily.



```
08:00
09:00 upload xplanet woody inputrc 2.4.20-vaio
10:00
11:00 Start: 09:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
```

Figure 2. The datebook is not too handy.

I installed the original SL-5x00 Addressbook, which required only putting the addressbook.xml file in ~/Applications/Addressbook. For some reason, the SL-5x00 series datebook is not compatible with the SL-C700—it displays only a line instead of the meetings scheduled. So I decided to go with Korganiser embedded, which uses the same format of the award-winning desktop software. I simply put the addressbook.xml file and the calendar on a CompactFlash card and copied them to the Zaurus.

Even with my best efforts over several days of attempts augmented by forum support, I did not come close to having the Zaurus syncing to either Windows or Linux. The desktop setup is a mystical adventure I may try again when I have more time or when an English user manual is available.

The Office Suite

HancomWord and HancomSheet, the Zaurus' word processor and spreadsheet, were interesting on the SL-5x00. They have matured into fast, easy-to-use and professional software on the SL-C700, able to read and save Microsoft files without any problem. Although the import may be slow sometimes, having the documents available everywhere is a pleasure.

The big 640 × 480 screen of the SL-C700 provides a true interface to the user. The screen rotation between 640 × 480 landscape mode and 480 × 640 portrait mode is supported perfectly. It also enables an editing session to take advantage of the wider screen and the keyboard, as well as a quick visualization of the data using a taller display and a simple press of the wheel.

Getting on the Net

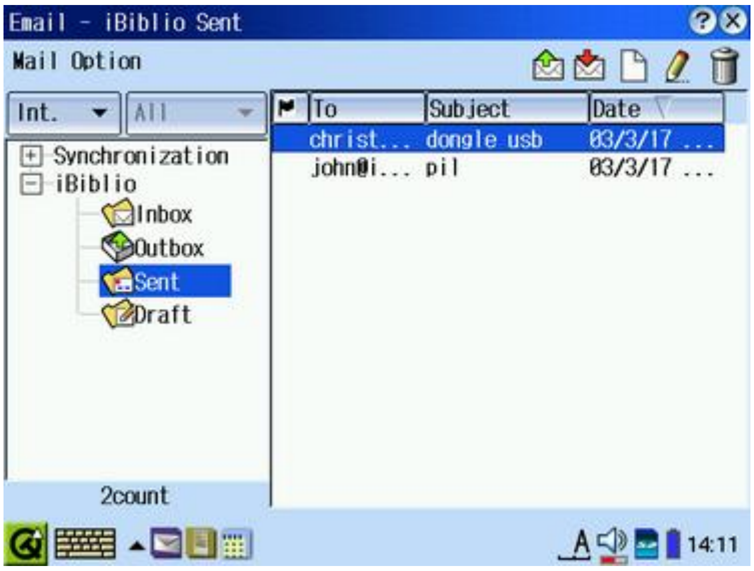
Configuring the internet connection is a kid's game. Being the lucky owner of a home wireless network, I had only to go to the network setting application, give a title to my connection, type my encryption key and select auto (dhcp) mode, a mode where IP addresses are assigned automatically. Plugging in a wireless CompactFlash card presented a globe-like icon with a big red cross on it. Clicking this icon showed me a list of available connections, where I chose the wireless connection I had set up and was connected. The list of connections means the SL-C700 is able to move from one network to another without any trouble—a necessary feature for a mobility device.

Both the Netfront browser and the Qtopia mail client are present on the SL-C700. They once again outperform their SL-5x00 equivalents. Although the Opera browser was not able to display every web site on the tiny SL-5x00 screen, Netfront is perfectly compatible with every web site I tried.

If the text is too small or too large, the Fn-3 and Fn-4 shortcuts allow the user to resize the display dynamically. The Fn-1 and Fn-2 shortcuts complete this customization with a brightness increase or decrease, which is useful when the ambient light changes.

Tabs allow multiple web sites to be opened at the same time. The browser is easy to use with self-evident menu items and icons and few configurable options. This ease is welcome as the only user manual is in Japanese.

The e-mail application also is easy to configure and use. I had to type only my POP server, user name, password and outgoing mail server to receive e-mail on the Zaurus. I immediately was able to reply off-line. Mails too big to be stored reasonably are not downloaded. It still is possible to retrieve them, but each message requires manual confirmation. Once again this is a useful feature to prevent the PDA memory from becoming full of junk mail and useless attachments.



	To	Subject	Date
	christ...	dongle usb	03/3/17 ...
	john@i...	pil	03/3/17 ...

Figure 3. Excellent e-mail support is a big plus for the SL-C700.

The Multimedia Suite

Both an audio player and a video player are present on the SL-C700. They are completed by a picture viewer and an audio recorder. The audio player plays MP3 files perfectly, keeping a playlist for the user. The sound output is excellent, and the user interface for the audio player is intuitive. A volume control and a randomize function allow the SL-C700 to be used as a digital jukebox. The audio player is completed by a clever display-off button that turns off the whole screen, saving battery usage to prolong the musical experience.

In the case of the picture viewer, a list of the digital pictures I had on my CompactFlash card was presented with thumbnails. I could then click on each thumbnail to see the pictures individually or start a slideshow to use the SL-C700 as a digital picture viewer. My fiancée was impressed by the quality of the pictures we took on New Year's Eve on such a small display. They were crisp and bright, with such beautiful colors it was hard to believe we were viewing them on a PDA.

I do not like carrying a laptop with me when I am traveling, which can be annoying when the on-flight movie is bad. But next time I take a long flight, I will not envy the big, portable DVD player the person sitting next to me has. Saving a DivX file to a 256MB CompactFlash disk, then watching it on my PDA, however, certainly will make my flight neighbor jealous. Thanks to the Doctor Z video player, it is possible to enjoy a high-quality playback without frame drop if the movie has been recoded with some settings regarding the supported DivX codec, the SL-C700 screen and supported frame rate.

Java

Another interesting feature of the SL-C700 is the complete Java support. I always am looking for more knowledge of the sky and galaxies, so sky maps are the first applications I install. I went to the Solun web site at www.piecafe.demon.co.uk and downloaded the Java version, designed for the SL-5500. It worked like a charm on the SL-C700, after some minor tweaks to use the 640 × 480 resolution. The high resolution allows comfortable use, even if the lack of memory quickly reminds you the SL-C700 is not perfect.

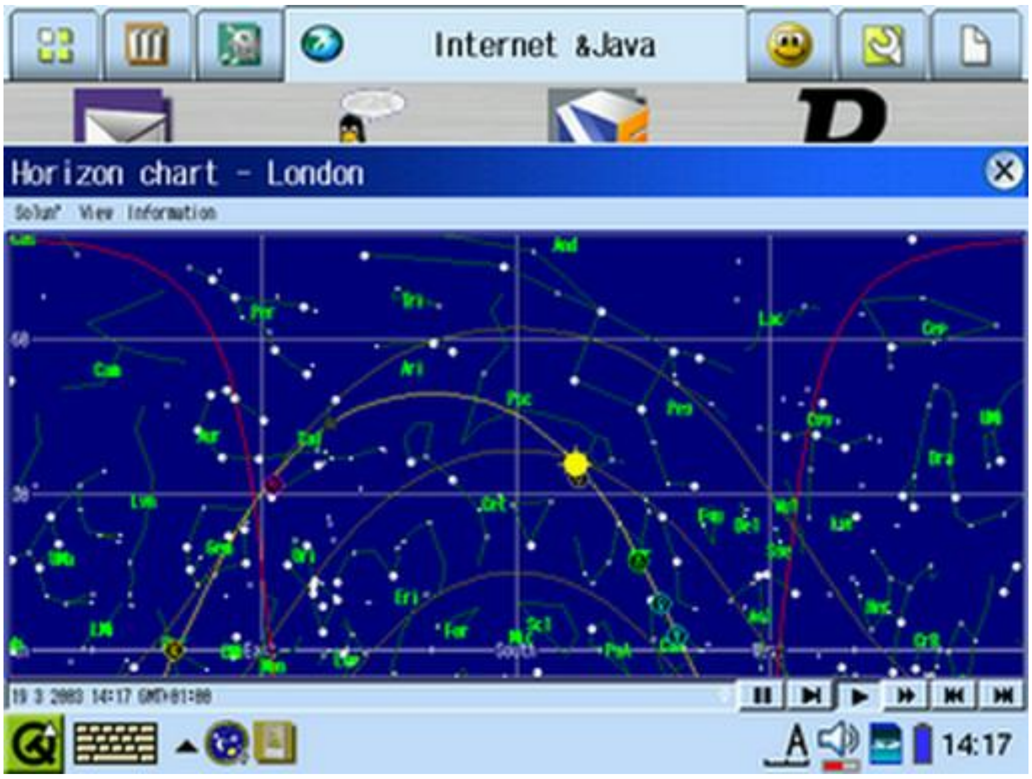


Figure 4. Java applications take advantage of the big screen.

Drawbacks

After the honeymoon with the new toy, I realized the SL-C700 was not without fault. The first problem is the resolution tweaking. Although using a different processor from the one the SL-5x00 uses, every SL-5x00 application should work fine on the SL-C700. However, some have been coded using fixed-screen sizes, which cause them not to scale well on a bigger display. To avoid this problem, Sharp introduced a low resolution (240 × 320) portrait mode to emulate the SL-5x00 screen better. It is active by default, however, and the transition between high resolution and low resolution takes four seconds, which seems like forever if you need the application at that moment.

The Doctor Z video player works fine and officially is SL-C700-compatible. The SL-C700 also comes with excellent e-mail support. I tried the free SL-5x00 applications, and most of them work fine even if the high resolution mode is selected. It takes only a long click on the application icon then selecting/ deselecting the run-in compatibility mode icon to give it a try.

Another annoying problem is the lack of memory. 64MB of Flash means 64MB of storage space, where 30MB are left for the user. Although 32MB may seem a lot for a PDA, Qtopia is memory hungry, which does not leave a lot for the user. When I run the audio player and a command line, only 600K are left. Even worse, only 4MB of memory are free after a clean boot up, which means there is little room left for the applications to run. I had many errors due to the lack of available memory when using Java applications. The device became sluggish until a screen suddenly appeared and asked me to stop some applications.

A serious problem my unit had was the SD port; for some reason, inserting an SD card resulted in a complete lock until the card was removed, when the traditional four-minute reboot takes over. I had to send it back to Dynamism.com.

I also could not find any cases or accessories for the SL-C700. In Japan, some accessories are starting to be made available, but it is hard to order them on-line when you do not speak Japanese.

I must admit that even with these problems, I miss my SL-C700. The default PIM suite is not really usable, but the multimedia suite, the office suite and the internet suite were more than what I needed to be happy with my PDA. Most of the problems I experienced are explained easily by the lack of official support in English. The lack of XML support and SL-5x00 backward compatibility in the PIM is a bigger issue, because it certainly cannot be solved by official support.

With some help found on the SL-C700 forums on externe.net/zaurus/forum, I will start using the SL-C700 on a daily basis when it comes back. It will completely replace my former PDA, the Zaurus SL-5500, when I can run a good PIM suite that syncs to the desktop computer.

Product Information

email: g@7un.org

Guylhem Aznar is the coordinator of the LDP (www.tldp.org). In real life, he is a consultant, a sixth-year medical student and is preparing a PhD in Computer Science. With the little time left, he enjoys playing with his Zaurus.

Archive Index Issue Table of Contents

Advanced search

SCO Linux 4

Steve R. Hastings

Issue #110, June 2003

SCO Linux 4 provides the bare bones of a high-availability clustering solution.

SCO Linux 4 is a server operating system intended for the same market segment as Red Hat Enterprise Linux. SCO Linux is based on UnitedLinux, a common base Linux distribution put together by four companies: SCO Group, SuSE, Conectiva and Turbolinux.

UnitedLinux has embraced all the major Linux standards, including the filesystem hierarchy standard (FHS), Linux standard base (LSB) and Open18N internationalization. On top of this stable foundation, each partner in UnitedLinux can ship extra features or customizations. For the most part, though, it should be easy to move from one version of UnitedLinux to another.

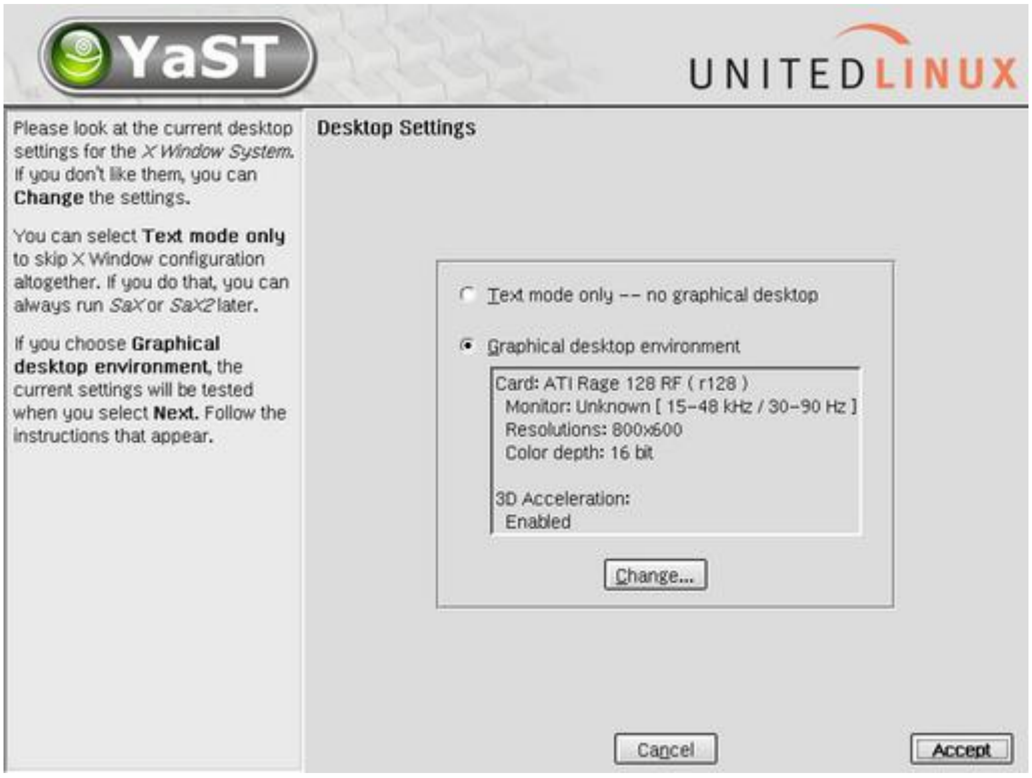
UnitedLinux clearly is intended to run on servers rather than on workstations. It uses the Linux 2.4.19 kernel with the O(1) scheduler patch applied, and it has server features enabled, including large memory, IPv6, logical volume management (LVM) and enterprise volume management system (EVMS).

Although UnitedLinux supports the major server architectures—x86, IA-64 or Itanium, AMD's x86-64 and IBM's zSeries, iSeries and pSeries—SCO Linux 4 currently supports only IA-32. SCO has promised IA-4 support in the near future. SuSE Linux Enterprise Server 8, also based on UnitedLinux, offers x86, IA-64 and IBM zSeries, iSeries and pSeries support.

Installing

SCO Linux 4 is distributed on three CDs. The first is the installer CD, and it also contains SCO-specific packages to install. The other two CDs are the standard CDs for UnitedLinux 1.0. They provide the common core of UnitedLinux, and any Linux distribution based on UnitedLinux 1.0 includes them.

The installer is a rebadged YaST2 installer from SuSE. It correctly detected most of the hardware in my test system and set it up with sensible defaults. It partitioned my disk and set up ReiserFS on the main partition.



Please look at the current desktop settings for the *X Window System*. If you don't like them, you can **Change** the settings.

You can select **Text mode only** to skip X Window configuration altogether. If you do that, you can always run *SaX* or *SaX2* later.

If you choose **Graphical desktop environment**, the current settings will be tested when you select **Next**. Follow the instructions that appear.

Desktop Settings

( ) Text mode only -- no graphical desktop

(● Graphical desktop environment)

```
Card: ATI Rage 128 RF ( r128 )
  Monitor: Unknown [ 15-48 kHz / 30-90 Hz ]
  Resolutions: 800x600
  Color depth: 16 bit

3D Acceleration:
  Enabled
```

Figure 1. YaST2 installer, rebadged for UnitedLinux. The SuSE lizard remains.

For the most part, the installer worked well and the installation went smoothly. There were a few rough edges, but nothing an experienced Linux user will have trouble handling. For example, when I tried the GNOME desktop, I received an error message because the hostname wasn't in /etc/hosts. When I tried the KDE desktop, an error message from the sound server came up because the driver module for the sound card was not loaded. Someone new to Linux probably will need to contact SCO support to sort out a few things.

The packages available for installation are mostly server-oriented. It is possible to install packages from SuSE if something you want is missing. To test this, I installed the SuSE 8.0 package for Stella, an emulator for the Atari 2600 game system. It worked perfectly, and now I can play old Atari games on my enterprise server.

Configuring

It's possible to configure a SCO Linux 4 system using SuSE's YaST2, either character-based or GUI, but the recommended and supported way is to use Webmin. Webmin is a web-based front end for system administration that makes many tasks as easy as clicking buttons on a web page. Also included is Webmin's companion system, Usermin, a similar system that lets users configure their own accounts. Webmin and Usermin can be run locally or from any computer that can access the web server on the SCO Linux 4 system.

Many services are included, but few are enabled by default. Using Webmin, it's easy to start up only the services you need or set them to start up on particular init levels.

Running

When you log in with KDE 3, you see a nice desktop, ready to go. Icons are set on your desktop for Webmin and Usermin, and the KDE panel has a selection of useful program launchers.

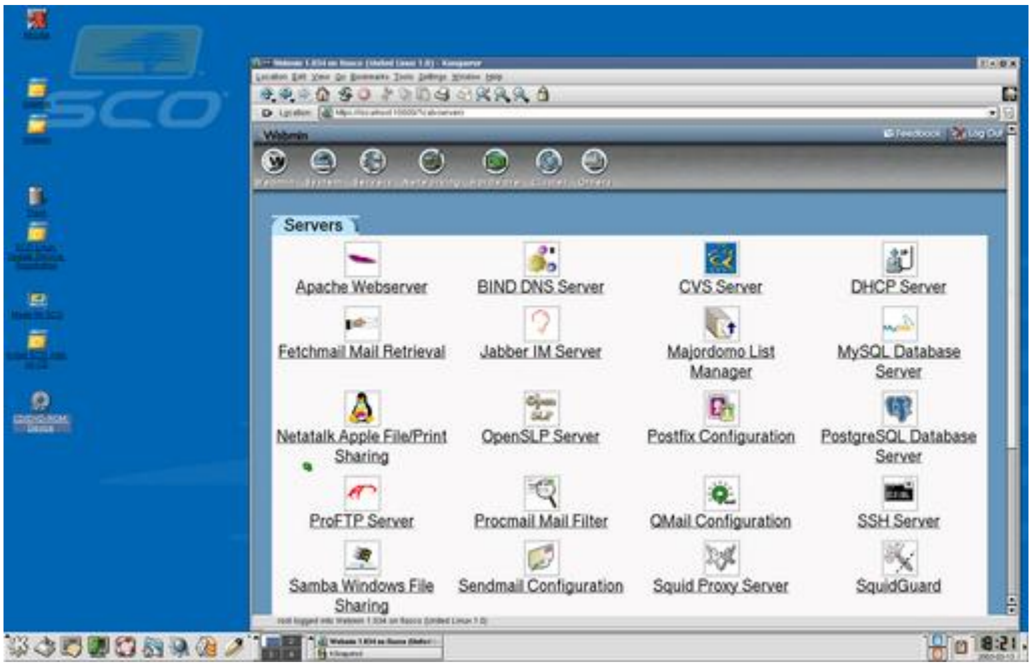
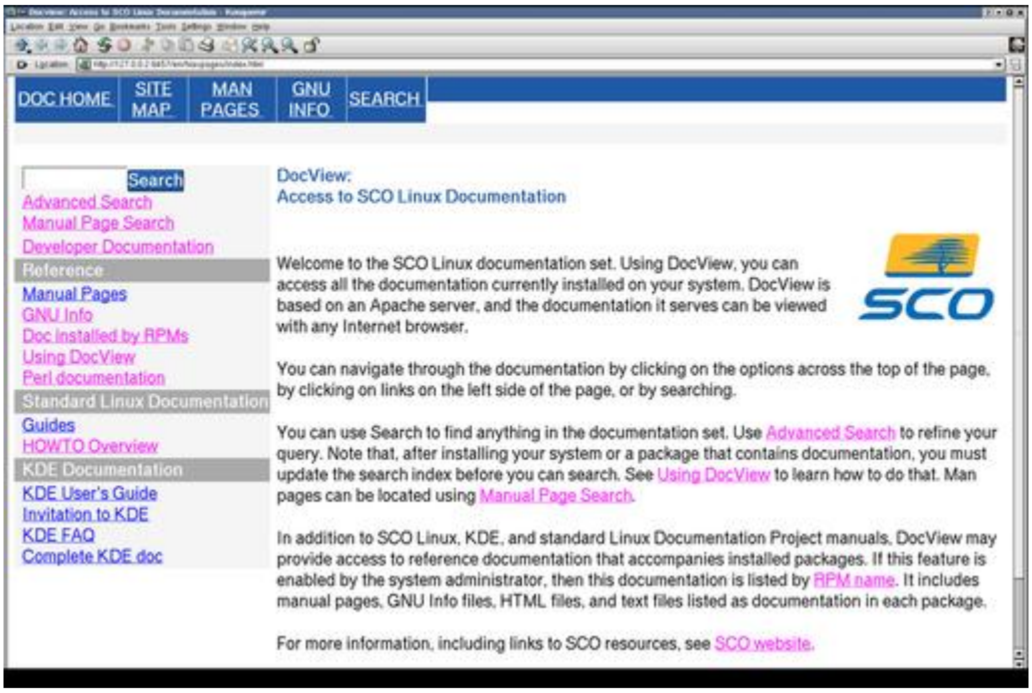


Figure 2. The KDE desktop, with a Webmin session in Konqueror.

Not so on the GNOME 2 desktop; it is barren, with few launchers to be found. UnitedLinux includes only the minimum core of GNOME 2.0—even GNOME Terminal is missing. The GNOME 2 support includes everything you need to run GNOME 2 applications under KDE or to launch KDE applications from the GNOME desktop. But if you actually want to work in the GNOME 2 environment, plan on spending some extra time installing missing pieces of GNOME and setting up a usable desktop.

DocView is a slick web interface that uses ht://Dig to put all the system documentation at your fingertips. You get man pages, GNU info pages, HOWTOs, Perl documentation and KDE documentation, all nicely browseable and searchable. GNOME, Python and other subjects are searchable but not included in the table of contents for browsing. To add them you would need to edit DocView files in /usr/lib or set up your own documentation pages.



DocView:
Access to SCO Linux Documentation

Welcome to the SCO Linux documentation set. Using DocView, you can access all the documentation currently installed on your system. DocView is based on an Apache server, and the documentation it serves can be viewed with any Internet browser.

You can navigate through the documentation by clicking on the options across the top of the page, by clicking on links on the left side of the page, or by searching.

You can use Search to find anything in the documentation set. Use Advanced Search to refine your query. Note that, after installing your system or a package that contains documentation, you must update the search index before you can search. See Using DocView to learn how to do that. Man pages can be located using Manual Page Search.

In addition to SCO Linux, KDE, and standard Linux Documentation Project manuals, DocView may provide access to reference documentation that accompanies installed packages. If this feature is enabled by the system administrator, then this documentation is listed by RPM name. It includes manual pages, GNU Info files, HTML files, and text files listed as documentation in each package.

For more information, including links to SCO resources, see SCO website.

Figure 3. DocView, a Slick Web Browser Interface to Documentation

Basic development tools are included, and they are up to date. GCC/g++ 3.2, Python 2.2.1, Perl 5.8.0 and gdb 5.2.1 are all included, but integrated development environments, such as KDevelop, Anjuta or IDLE, are not. However, Glade 1.1.1 for GNOME 2.x is included.

Services

Both sendmail 8.12.6 and postfix 1.1.11 are included. Apache 1.3.26 (rather than 2.x) is included with all the usual modules. Apache works right out of the box, serving up a “Hey, it worked!” page with a handful of useful links. Both Sun and IBM Java are present, along with JServe and Tomcat. Samba 2.2.5 and OpenLDAP 2.1.4 are included, along with pam_ldap and other support libraries. The Squid 2.4 caching proxy server also is provided. Tape backup is handled by Amanda 2.4.2, which allows you to use one server with a big tape drive to back up many systems.

SCO Linux 4 also provides the bare bones of a high-availability clustering solution: packages for Linux Virtual Server, DRBD, Heartbeat and Mon. Very little documentation about these services is available, either installed on your system or on the SCO web site. The SCO web site, under its Knowledge Center heading, has one technical article on Heartbeat that is helpful, but there is nothing on LVS, DRBD or Mon. The Red Hat web site contains much more documentation on high availability than this, and SCO Linux 4 has no equivalent of the Red Hat Cluster Manager.

SCO also includes their sysinfo tool. It's a script in /etc that looks all over the system, pulls out a lot of useful information and builds a web page with the results. Anyone supporting a large number of servers will like this feature.

Support

SCO makes support information available on their web site. Most of it requires you to log in before you can access it. Unfortunately, there isn't much information there yet, and what is there is organized in a confusing manner. For instance, a page for Security Advisories (www.sco.com/support/security) does not list any security advisories for SCO Linux 4. All information about SCO Linux 4 is in the Support Knowledge Center. At press time, there are 64 technical articles covering SCO Linux 4; 56 are notices of new packages (security patches or upgrades), four are bug information articles and four are general information articles. The general information articles include the Heartbeat article, an article on setting up Squid, how to allow root to log in remotely and where to get System V compatibility libraries.

The UnitedLinux installer can be automated with an XML options file. It also can import configuration files from Red Hat's Kickstart and convert them to the UnitedLinux XML format. There is no documentation about these features, however, either on the SCO web site or on the system. After I inquired about this, SCO technical support sent me a URL to a page on the SuSE web site that documents this information.

SCO's phone support was good. When I called SCO technical support, the people I spoke with were able to answer my questions quickly. They also followed up by e-mail to make sure my problems were solved. SCO has promised to support all releases of SCO Linux for a minimum of two years.

Updates are handled with the Advanced Package Tool (APT) system. APT originally was developed for the Debian GNU/Linux distribution, but Conectiva ported it to work with RPM packages. Once you have registered with SCO, you can update all packages on your system to the latest with one command, or you can update only specific packages. If you update a package that depends on other packages, APT updates the other packages as well, automatically. APT can get the packages from SCO's servers, using the Internet or from a Service Pack CD.

SCO's upgrade policy states you are not required to upgrade any packages if you don't want to do so. If you are calling support with a problem, though, they may tell you to upgrade some packages as the solution to the problem.

A Security Problem

During this review, SCO did not make an important security update available quickly, although other distributions did. On March 3, 2003, the CERT Coordination Center published a remote root compromise in sendmail, reported by Internet Security Systems. Red Hat and SuSE had patches available the same day as the CERT announcement, but SCO released their patch 11 days later. This left the phone support person in the unenviable position of having to tell me that no update was available. He did, however, send an RPM to me for testing. This RPM turned out to be the same one that was ultimately released a week later. It is surprising that SCO would take so long to release a major security patch to sendmail.

In the meantime, I downloaded the SuSE RPM for sendmail and tried it out; it installed with no problems. If the SCO phone support person had not been able to get the RPM to me early, I could have installed the SuSE RPM as a temporary fix.

Conclusion

SCO Linux 4 is licensed on a per-server basis, with four levels of support: base for $599, classic for $699, business for $1,249 and enterprise for $2,199. This price structure is similar to Red Hat Advanced Server's, but it is slightly less expensive at each price level. (At press time, Red Hat announced a lower server price point: Red Hat Enterprise Linux ES Base Edition is $349.)

UnitedLinux 1.0, upon which SCO Linux 4 is based, has some rough edges. SCO hasn't smoothed them out yet, and their available documentation isn't much better than you can get by surfing the Internet. On the other hand, with Webmin and APT, it should be easy to keep your servers humming once you do have them set up. Except for that one scary security situation, the tech support is good.

Product Information

Why 11 Days to Fix a Remote Root Vulnerability?

email: steve@hastings.org

Steve R. Hastings first used UNIX on actual paper teletypes. He enjoys bicycling with his wife, listening to music, petting his cat and making his Linux computers do new things.

Archive Index Issue Table of Contents

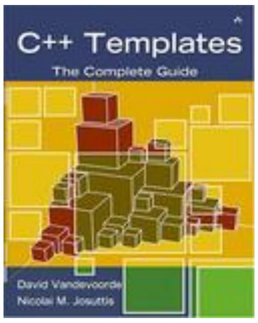
Advanced search

C++ Templates: The Complete Guide

Michael Baxter

Issue #110, June 2003

Book Review: *C++ Templates: The Complete Guide* by David Vandevoorde and Nicolai M. Josuttis



Boston, Pearson Education, Inc., 2003

ISBN: 0-201-73484-2

$54.99 US (hardcover)

This book is a user's manual for the future of C++ programming. C++ is going through a dramatic metamorphosis, moving far away from C-like pointer manipulation. With the introduction of the standard template library (STL), part of the C++ standard library, low-level coding for data structures and containers is no longer necessary. Instead, efficient generic tools already are available for your code.

One important construct underlying all the changes in C++ is templates, which introduce parameterized types for functions. This new book opens the door to understanding how templates work, with examples throughout.

Part I introduces template basics in seven chapters that extol the virtues of static polymorphism, which allows compile-time type checking that would not be possible at runtime. The authors explain the differences between function templates and class templates, how to use nontype parameters in templates and how to avoid numerous problem areas.

Part II explains templates in-depth and provides information about various kinds of arguments in template code, friend functions, the importance and control of names and how instantiation is done. It also covers how template arguments are deduced and how to decide between generic and specialized code.

The drama really unfolds in Part III. You even may catch yourself thinking, “Whoa, this is C++? I didn't know it could do all this.” The tone is set by a discussion of dynamic vs. static polymorphism. Dynamic polymorphism is the kind generally associated with C++--inheritance with dynamic dispatch at runtime. Templates are a form of static polymorphism, and compile time only seems to be limiting. But this turns out not to be the case. For example, C++ code can be constructed to moderate the *behavior* of classes by using templatized traits and policy classes. And, templates can be used *with* inheritance for applications such as parameterized virtuality, which combines the best of two forms of polymorphism.

Metaprogramming, normally associated with Lisp is one of the most powerful uses of templates. Templates actually can be used to construct program-writing programs, or code generators. After the unveiling of capabilities in Part III, Part IV offers a reprise on advanced applications. A highlight is being able to do type classification with templates. Templates allow you to use the type algebra in C++ directly for your own applications, a key to behavioral patterns.

Templates have been used to construct several useful C++ libraries, including the vast collection at boost.org. Both authors are experienced C++ programmers and are closely associated with the C++ Standards Committee. Vandevoorde is a cofounder with of the comp.lang.c++.moderated newsgroup, and Josuttis has written extensively on C++, including *The C++ Standard Library*.

—Michael Baxter

email: mab@cruzio.com

Letters

Various

Issue #110, June 2003

Readers sound off.

Happy Chinese New Year, Tux

Thought you might be interested in this picture I took across from the main train station in Taipei, Taiwan. It is a very large Linux penguin, dressed up for Chinese New Year. I just finished up with Apricot (the Asian networking conference), and Linux had a large presence in the IPv6 appliances on display, as well as the research being presented. The research lab I work in has many Linux systems. The software engineers do most of their work on Linux and FreeBSD.

—Van Emery

Debit on the Left, Credit on the Right

I just read the article “Linux for a Small Business” by Gary Maxwell [*LJ*, April 2003]. He states that when you pay for a book, you debit your cash account and credit your expense account. I believe it should be the other way around. In accounting, debit refers to the left side of a T-account and credit refers to the right side of a T-account. The words are of Latin derivation meaning left and right. They don't mean decrease or increase. Depending on the type of account, a debit can either increase it or decrease it. The same applies to credits. I think this is probably the source of most non-accountants' confusion about double-entry accounting.

—Kathy

Network Desktop Advice

Thanks for the great article listing rdesktop, Marcel [Cooking with Linux, *LJ*, April 2003]. In a heterogeneous environment, rdesktop works nicely for getting to Windows machines. I have also found tsclient (www.gnomepro.com/tsclient), a GNOME 2 front end to rdesktop that looks and acts exactly like the Windows Terminal Services client. For those of us who must work in this environment, it helps things a little bit.

—Jeremy

MST Helps Brazil's Poor

To Jon “maddog” Hall: I am Brazilian and would like to let you know that I agree with your response to Bruno Trevisan's letter about the “Landless Workers' Movement”, Movimento dos Trabalhadores Rurais Sem Terra (MST) [*LJ*, February 2003]. I sincerely thank you for bringing out the facts. MST is addressing the very basic needs of landless people in Brazil in a conscious, organized and effective way. MST has shown results—tons of results. The land issue in Brazil is a serious one, and Trevisan's statements show a total lack of respect toward people in need of basic things, people in poverty, people that die from hunger. Former President Cardoso never had his farm destroyed. The Brazilian army was always securing his properties because of the lack of response of local police. A few years ago, in the Brazilian state of Para, in the Amazon region, the state police cowardly murdered dozens of rural workers. It took a very long time to bring the police officers to trial. A few years ago, in the state of S. Paulo, the state police invaded a prison and murdered 111 prisoners. Again, it took a very long time to put the responsible ones on trial.

—Nuno Vasconcellos

“Wardialing” in 1979

The term wardialing was in use in 1979-80. It was used to describe either linear or random dialing of phone numbers and keeping tabs on the modem carrier detects (CDs) received. Wardialing also was used interchangeably to signify using those numbers, getting the codes from the long-distance carriers, or if you were lucky, a PBX connect to trunk calls and make party lines. Nothing too intelligent, just brute force. Not that I ever did any of that. I think the term war was that you were at “war” with the phone company doing this. Think of it as carpet bombing the telco switch. Eventually, you'll hit something. There were a lot of turf wars like they have now, and essentially, the biggest “list” wins. The movie *Wargames* (1983) had nothing to do with it whatsoever.

—Craig

GNOME 2 Drops Features of Version 1

I disagree strongly with your description of GNOME 2 as “an excellent choice for first-time and nontechnical users” [“The GNOME 2 Desktop Environment”, *LJ*, April 2003]. I used GNOME 1 and persuaded my wife to use it too, but when I installed Red Hat 8 with GNOME 2 we found things had gone backward.

Under GNOME 1, with the Sawfish window manager, I set up all sorts of keyboard shortcuts. This was reasonably easy. There was a decent keyboard shortcuts tool, which allowed you to set the context (global, window, title) and then set shortcuts like Alt-MOUSE3, etc., with a long list of commands to assign them to. Now under GNOME 2, I find a very primitive “keyboard shortcuts editor” that offers a much smaller number of predefined commands and does not allow you to specify the context. Why adopt a worse new window manager without offering the choice of keeping the old one?

We wanted to have some programs run on GNOME startup. I looked in the menus for something like “startup”. It took ages: I finally found the required functionality in a program that runs when you click on Extras-->Preferences-->Sessions. Well hidden!

My wife found GNOME 2 unusable and unfriendly. I found it shockingly weak on everyday functionality, compared with GNOME 1. Overall, the point of a desktop is to make the computer more usable, but I see no sign of any user-sensitivity in the GNOME desktop. It feels like a half-baked programming project, not a user-oriented functional tool.

—Dr Mark Alford

HTTP User-Agent in Mozilla

Gary Maxwell's article “Linux for a Small Business” [*LJ*, April 2003] is very useful for an average desktop user like me. I'd like to add a small correction to his statement that “Mozilla lacks a feature that Konqueror has: changeable user agents.” In Mozilla 1.3 and later, the user can change User-Agent by changing profiles.

—Hiroshi Iwatani

Ready to Make Movies

I would like to say thanks for some really inspiring and interesting stuff about movies in *Linux Journal*. The January 2003 issue about *Star Trek* was excellent. I would also like to give a special thanks to Robin Rowe, because he's actually the reason I'm switching to Linux. The biggest problem as a 3-D animator is that there are not enough 3-D applications for Linux. Currently, I'm using the free Blender 3-D application, and I'm very impressed.

—Jesper Christensen

Robin Rowe replies: Thank you, and you may want to try Wings. That's free, too. See www.linuxmovies.org for a list of more movie-related software.

Linux Training?

It would be great if you had a training section in the magazine highlighting where you can get free or paid training for the topics in that month's magazine.

—Mike Hjorleifsson

We Are Not Riffraff

In response to the letter “..and Loses Another” [Letters, *LJ*, April 2003] that accuses *LJ* of containing “apologies for terrorists and other assorted anti-American, third-world riffraff”, I don't believe the editorial team should worry about losing another loser, but should rejoice that *LJ* does not encourage such xenophobia. Linux provides an opportunity to rise above this type of nationalism. Open-source software can help create a more equitable sharing of knowledge and access to wealth, and this is a great thing. Linux is most definitely a multinational effort in the best traditions of freedom and democracy (a European invention). And yes, that operating system you use every day includes contributions from the third world too.

—Ian, a citizen of the world

Life without *LJ* Is Pain

How dare you! I had canceled my subscription a few months ago, then what do I find when I look at *Linux Journal* at my local Borders bookshop? I find interesting, technically unrepentant articles. I find excellent design and a good balance between news, discussion and facts. The cheek of it! I was shocked by the current issue [April 2003], which I was forced to buy—a GNUstep programming introduction, a brilliant inspiring GIMP tutorial, USB drivers, a kernel cryptography overview, CMS chitchat, teasing screenshots of GNOME 2, gossip on kernel patches for the SGI VISWS—and all on lovely glossy paper! You do realise the pain you're putting me through knowing that I'm not subscribing anymore! People I have spoken to were really inspired by your GIMP tutorial. They couldn't wait to get back to their mice and keyboards to try it out. I think that's the key really: to show people how powerful the tools they already have are. “I didn't know you could do X with this Y I have here” is a nice feeling!

—Tariq

Don't Try to Mimic Another OS

I don't get why everyone wants Linux GUIs to look like Microsoft Windows. I've supported Mac users and Windows users for years. It never fails to thoroughly confuse Mac users on Windows systems when something *almost* works like on their Mac, but not quite. Most users I've worked with had an easier transition from one platform to another when the two had very little semblance to each other.

—LT

Scribus Progress?

Are there any plans for an article or two on the state of desktop publishing software and production on Linux? I'm in the middle of a small press startup, and I'm hoping to be able to standardize on Linux. I've looked at software packages like TeX, LaTeX, LyX and Scribus. None of them seem to be quite ready for my needs, though Scribus is very close. I think you guys are probably in a prime position to write articles on this subject, being already in the publishing industry.

—Charles

Put maddog's Letter on the Web

I regularly pick up the *Linux Journal*, but in February I was unable to. Now having picked up the March issue and seeing two letters about maddog's “I believe” response and how inspirational it was, I would really like to see it. Is it possible to post it on the web site?

—Chris Bruner

Yes: www.linuxjournal.com/article/6770—Ed.

Freedom Threatens Some Companies

I read with some concern your recent article on Koha [*LJ*, February 2003]. I wondered what would happen to small- to medium-sized software companies that currently produce competing library systems. No doubt these companies have invested real money into developing these products and need some return on their investment in order to survive. This led me to wonder about the long-term effect of free software. Any software product that has to compete with the equivalent open-source product must surely struggle to survive. This means one open-source product will eventually dominate each market niche leaving consumers with no choice. Presumably a lot of open-source products are developed by charitable professional software developers during their free time. Will the pool of professional developers shrink as there is little work available other than charity work? Why should large organisations increase their profit margins by significantly reducing the cost of software essential to their organisation? Charity is fine for those who cannot afford to pay but shouldn't be exploited by those who can.

—John

Richard Stallman covered many of the issues you raise in the 1985 GNU Manifesto (www.gnu.org/gnu/manifesto.html), and they have been the subject of intense community discussion ever since. That page, and a web search for links to it, is a good way to catch up on the debate—Ed.

UPFRONT

Various

Issue #110, June 2003

LJ Index, diff -u and more.

upFront

AeroMail

A good choice for a spartan web-based mail agent is AeroMail. It has no frills, like address books, but it does allow you IMAP access to your mail server. It sports folders for storing and classifying mail. You can reply, forward and compose mail messages. All the important functions are present, but not a lot else. Perfect for road warriors who use a regular mail client when they return. Requires: web server; PHP, compiled with IMAP support; IMAP server.

—David A. Bandel

diff -u: What's New in Kernel Development

The **FUSE (filesystem in user space) Project** has reached version 1.0. One of the basic ideas of UNIX is to provide tools that perform fundamental operations in generic, interoperable ways. These tools typically work on streams of data stored in the filesystem as files. But some tools, including SSH, FTP and compression tools, interfere with this, because they either convert the files into a difficult-to-use state or they interact with systems located elsewhere on a network. Still other tools insist on providing all control internally to themselves, so none of the basic UNIX tools are of any use. FUSE allows the user to layer the appearance of a filesystem over any or all of these programs, so all operations can be done as file manipulations, taking advantage of the great wealth of basic tools available in UNIX.

The **virtual memory (VM)** subsystem in 2.4 now has some good documentation, courtesy of **Mel Gorman.** Once upon a time, **Linus Torvalds** dropped an entirely new VM subsystem into the Linux kernel, smack dab in the middle of a stable release series. This was tremendously controversial among developers, one reason being that Andrea Arcangeli, the author of the new code, provided virtually no documentation of any kind. Mel put in six months of work, much more than he originally thought would be necessary, resulting in a solid explanation of the workings of the entire VM, along with specific commentary on the source code itself.

The **Kernel Bug Database** has been documented extensively by its creator, **John Bradford.** Intended as an improvement over the current Bugzilla database used by a number of developers, the Kernel Bug Database rejects a generic approach, providing features based on the specific needs of the Linux kernel, such as the ability to search based on options in the .config file.

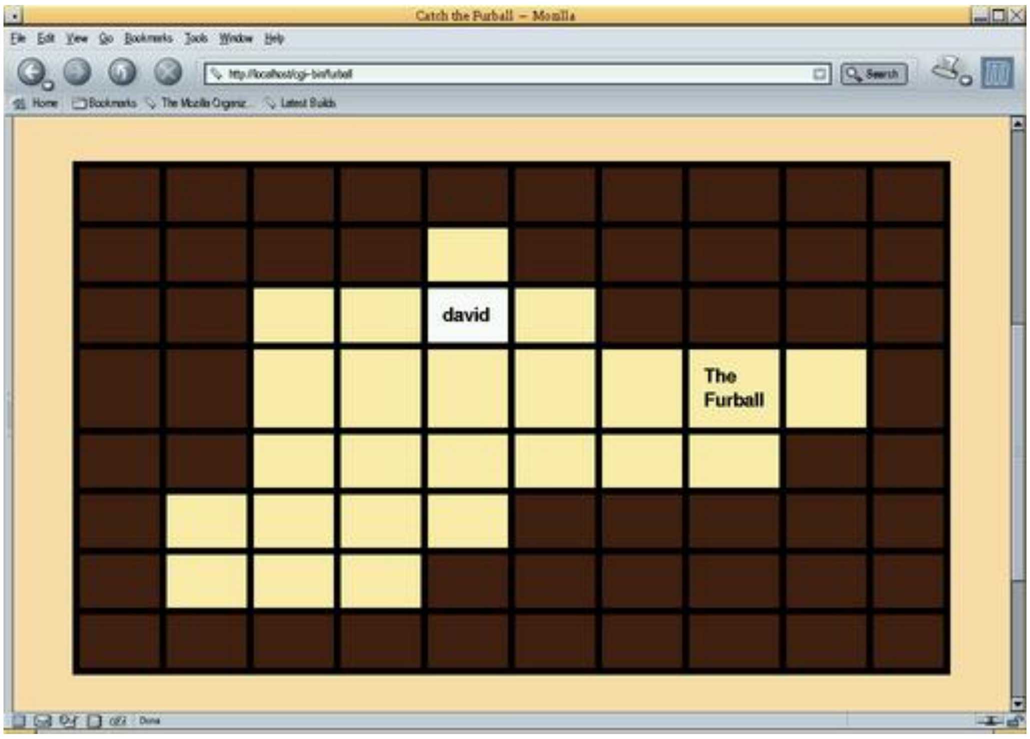
Dynamic kernel module support (DKMS) has come from a developer group at Dell. A GPLed project, it aims to allow device driver source code to reside anywhere on the filesystem, not only in the kernel source tree. This makes it easier for vendors to release new versions of their drivers and for users to recompile those drivers. As of March 2003, DKMS is 2.4-specific, and it doesn't take account of some of the massive reworkings appearing in 2.5, especially with the module code itself.

Several groups are working to implement **IPSec for IPv6**. The IPSec suite of protocols presents a framework for providing privacy and authentication support at the IP address layer, while IPv6 attempts to expand the number of available IP addresses. Although IPv6 is not yet in widespread use, it is important to continue to build the infrastructure to one day migrate away from the ailing IPv4 standard. **Kazunori Miyazawa, Kunihiro Ishiguro, Hideaki Yoshifuji** and **Mitsuru Kanda** recently joined forces to produce working IPv6 IPSec support in the 2.5 kernel tree.

—Zack Brown

Furball

Here's a fun game for the kids, but you can edit it and make it suitable for anyone. It's a game to learn about others, with questions and more. Think of it as a truth-or-dare game. Requires: web server, web browser, Python.



—David A. Bandel

LJ Index—June 2003

1. Forecasted global percentage increase in IT spending for 2003: 4
2. Server growth percentage forecast: on top of a 50%-plus growth rate in enterprise servers in 2002, a 40% growth for Linux servers in 2003.
3. Forecasted 2003 percentage growth rate for Linux in Asia: 24
4. Forecasted 2003 percentage growth rate for Microsoft Windows in Asia: 6
5. IBM's claimed Linux revenue, in billions of dollars: 1
6. HP's claimed Linux revenue, in billions of dollars: 2
7. Damages in billions of dollars sought by SCO in a lawsuit against IBM for disclosing trade secrets in SCO-licensed AIX source code: 1
8. Percentage cost-savings range experienced by Merrill Lynch since deploying Linux on IBM mainframes: 40-50
9. Projected minimum yearly savings in millions of dollars for Merrill Lynch by fully deploying its Linux-on-mainframe strategy: 100
10. Number of Linux boxes currently in production at Morgan Stanley: 400
11. Number of Linux boxes currently “in the pipeline” at Morgan Stanley: 300
12. Price/performance increase multiple Morgan Stanley experienced on six new four-way Linux boxes: 13
13. Percentage improvement in cost experienced by Lehman Brothers with Linux: 50
14. Percentage improvement in performance experienced by Lehman Brothers with Linux: 20

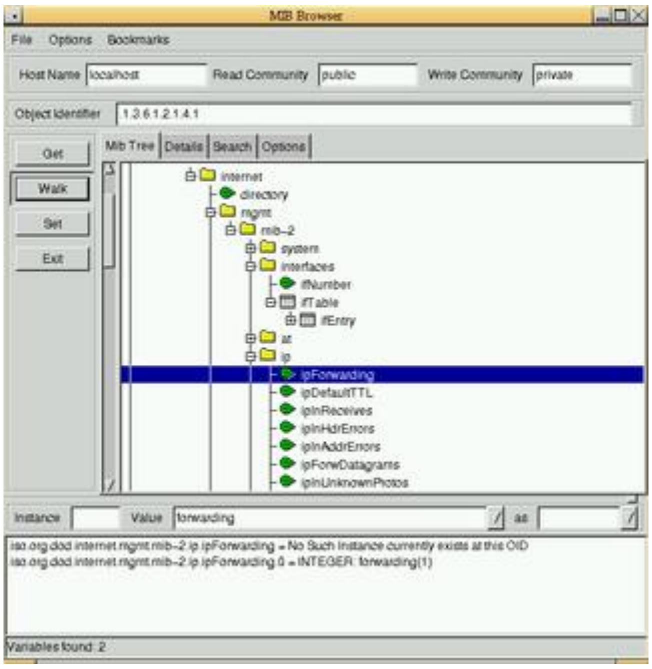
15. Percentage bandwidth capacity of ordinary phone wire to households currently utilized: 1
16. Percentage of IBM's servers currently sold that are Linux-driven: 15-20
17. Percentage annual growth in Linux users over the next few years, predicted by Sun CEO Scott McNealy: 30

Sources

1,2: Aberdeen Group3,4: International Data Corp., in *Economic Times*5,6: *eWeek*7: SCO8-14: Risk Waters Group15: Bob Frankston16: Jim Stallings, IBM17: Associated Press

Mbrowse: www.kill-9.org/mbrowse/index.html

This is a nice, simple-to-use and easy-to-install management information base (MIB) browser. Those of you who use Simple Network Management Protocol (SNMP) know how easy it can make life. The Details tab provides information about those MIBs you might not know a whole lot about or use often, so you can interpret the information or make changes using the browser. Requires: libgtk, libgdk, libgmodule, libglib, libdl, libXi, libXext, libX11, libm, libnetsnmp, libwrap, glibc, libcrypto, libnsl.



Host Name: localhost Read Community: public Write Community: private

Object Identifier: 1.3.6.1.2.1.4.1

Instance: Value: forwarding as:

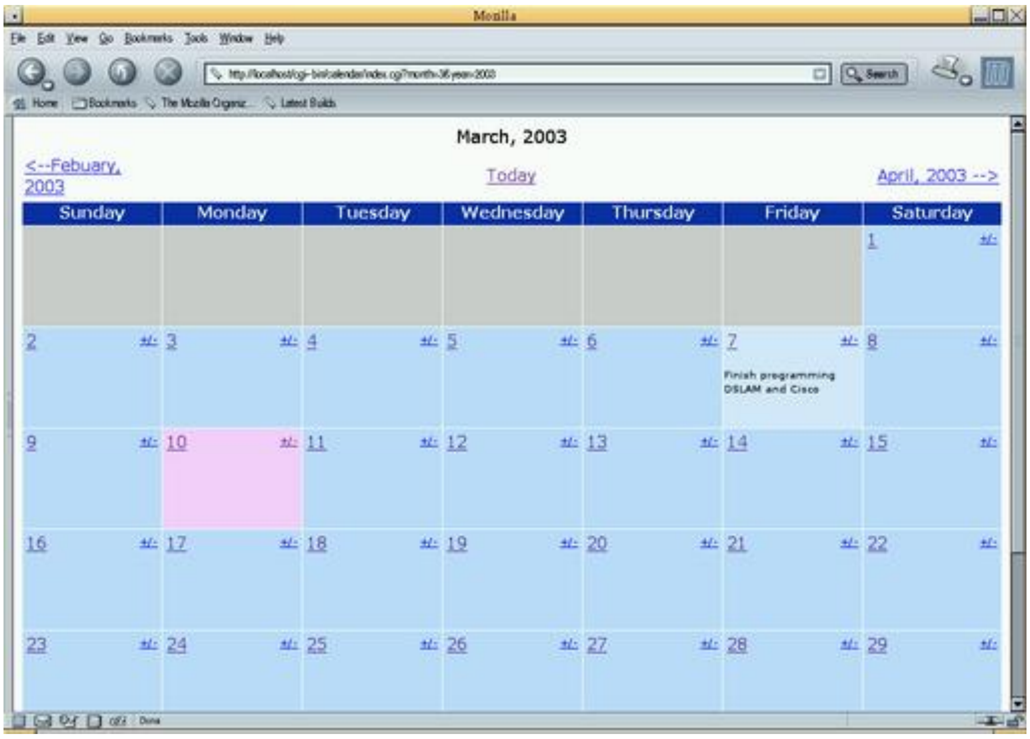
```
iso.org.dod.internet.mgmt.mib-2.ip.ipForwarding = No Such Instance currently exists at this OID
iso.org.dod.internet.mgmt.mib-2.ip.ipForwarding.0 = INTEGER: forwarding(1)
```

Variables found: 2

—David A. Bandel

My Calendar: fuzzymonkey.org/newfuzzy/software

If you need only one calendar—not one for everyone, but one for yourself or for the office or the Web—this program is extremely easy and quick to install. And as long as the protected/ directory is protected, you don't need to worry about someone changing your appointments. This calendar also e-mails you the next three days' appointments if you set up cron to run the e-mail script. Requires: web server, Perl, cal, pscal (optional).


Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost/cgi-bin/calendar/index.cgi?month=3&year=2003
Search
Home Bookmarks The Mozilla Organiz... Latest Builds
Done

March, 2003

<--Febuary, 2003 | Today | April, 2003 -->

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
						1 +/-
2 +/-	3 +/-	4 +/-	5 +/-	6 +/-	7 +/- Finish programming DSLAM and Cisco	8 +/-
9 +/-	10 +/-	11 +/-	12 +/-	13 +/-	14 +/-	15 +/-
16 +/-	17 +/-	18 +/-	19 +/-	20 +/-	21 +/-	22 +/-
23 +/-	24 +/-	25 +/-	26 +/-	27 +/-	28 +/-	29 +/-

—David A. Bandel

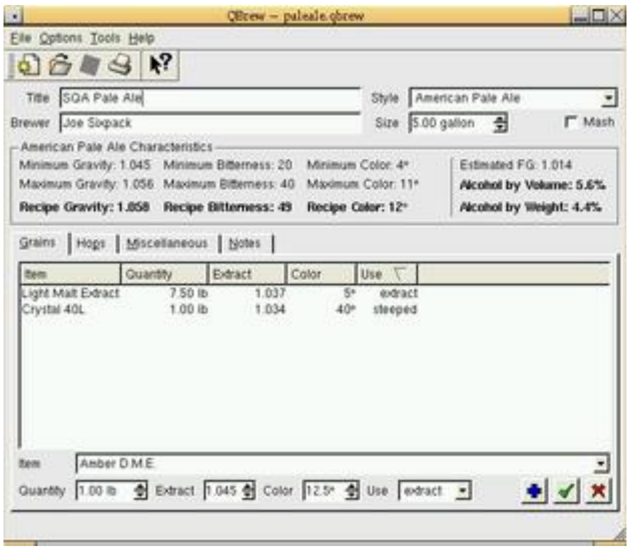
ps-watcher: ps-watcher.sourceforge.net

If you need something to keep an eye on your process table and perform some action based on said table, you need ps-watcher. Every day I used to have to look for errant Netscape processes and kill them. Well, ps-watcher can do this before the system comes to a crawl. You can base actions on percentage of CPU time used and other parameters. Simply define the parameter in the config file and set an action to take. You can log, kill, log and kill, and any of a number of other actions. Requires: Perl, Perl modules Sys::Syslog, File::Basename, Pod::Text, Config::IniFiles, Getopt::Long.

—David A. Bandel

QBrew: www.usermode.org/code.html

A virtual beer is great, but a real one is undeniably better. Well, why not go that one step further and brew your own to your own tastes? This program comes with a tutorial for brewing real, not virtual beer. When you find the best combination of ingredients for a really hearty ale, pass the recipe on. Requires: libSM, libICE, libXext, libX11, libqt-mt, libstdc++, libm, libgcc_s, glibc, libdl, libfontconfig, libaudio, libXt, libpng, libz, libGL, libXmu, libX render, libXft, libfreetype, libpthread, libexpat.



Title: SQA Pale Ale
Style: American Pale Ale
Brewer: Joe Sixpack
Size: 5.00 gallon
Mash

American Pale Ale Characteristics

Minimum Gravity: 1.045 Minimum Bitterness: 20 Minimum Color: 4°
Maximum Gravity: 1.056 Maximum Bitterness: 40 Maximum Color: 11°
Recipe Gravity: 1.058 **Recipe Bitterness: 49** **Recipe Color: 12°**

Estimated FG: 1.014
Alcohol by Volume: 5.6%
Alcohol by Weight: 4.4%

Item	Quantity	Extract	Color	Use
Light Malt Extract	7.50 lb	1.037	5°	extract
Crystal 40L	1.00 lb	1.034	40°	steeped

Item: Amber D.M.E.
Quantity: 1.00 lb Extract: 1.045 Color: 12.5° Use: extract

—David A. Bandel

They Said It

Alas, 2003 will not be the year of the enterprise Linux desktop; however, expect support from the large system vendors such as Dell, HP, IBM and Sun to be on the increase for desktop Linux (from practically nothing in 2002), as they realize that they can sell more Linux servers if there is a viable desktop Linux.

—Aberdeen Group

By 2007, we said one year ago, “No one will be fired for recommending Linux.” Shortening our own timeline by four years, we suggest that an IT buyer might already be fired today for failing to consider Linux. That's a small step but one of Neil Armstrong caliber.

—*eWeek*

Linux is a large component of our five-year computing strategy. We are investing and deploying it heavily in all areas of our Institutional Securities business. It's currently being used for mission-critical applications in our Equity and Fixed Income Divisions.

—Jeffrey M. Birnbaum, Morgan Stanley's global head of enterprise computing for the Institutional Securities business

We're no longer locked into a development platform. If we were going to port an application [to HP-UX], there would be some problems. Going with Linux, we can run the application on commodity hardware—IBM, HP or Dell—and take advantage of the benefits of the platform.

—Bridget O'Connor, Lehman Brothers

The other thing is continuing to enable all the platforms in the IBM family of products. When you do that, then no matter where the customer interacts with us, Linux is a part of this picture.

—Jim Stallings, IBM's Linux manager

Linux will not be very useful to ordinary people. It will be more useful to companies like ours.

—Scott McNealy, CEO, Sun Microsystems

Lethal Linux

Future Combat Systems (FCS, www.darpa.mil/fcs) is a $4 billion Defense Advanced Research Projects Agency (DARPA, www.darpa.mil) program with an immodest purpose that its web site puts this way:

> The FCS program will develop network-centric concepts for a multi-mission combat system that will be overwhelmingly lethal, strategically deployable, self-sustaining and highly survivable in combat through the use of an ensemble of manned and unmanned ground and air platforms.

The Lead System Integrator (LSI) selected for the Army's FCS is The Boeing Company's Science Applications International Corporation (SAIC, www.saic.com), which is expected to field its results in the year 2010. The selection announcement was made in March 2002. In November 2002, at a Boeing C4ISR (Command, Control, Computers, Communications, Intelligence, Surveillance and Reconnaissance) conference, a Boeing FAQ (www.boeing.com/defense-space/ic/fcs/bia/faq_c4isr_conf.html) addressed the operating system question:

> Q: COMPUTERS—What operating system will FCS use? Windows? VX Works? Lynxos? Linux? Other?
>
> A: FCS C4ISR has selected the Linux OS.

—Doc Searls

C++? Are You Crazy?

Don Marti

Issue #110, June 2003

Cool projects are using this big, complicated language. Maybe you should too.

About two years ago, programmer Dan Egnor posted to advogato.org with the question, “Why don't C++ and free software mix?” He pointed out that freedom-loving software developers tend to stay away from C++.

But, he added, although C++ is a big complicated language with “terrible pitfalls and simple misfeatures”, it is standardized and offers good flexibility and performance. And, he wrote, “its standard library includes the STL, which knocks the socks off anything available in the C world for power, flexibility and efficiency.”

Or, does C++, as many have argued, represent the worst of both worlds, an infertile middle ground between the simplicity and control of C and the almost-automatic everything of Perl and Python?

Today, though, it might be time for a second look at this much-maligned language. For two big reasons books on standard C++, templates and all, are on my to-read stack above the more tempting ones on the next great scripting languages. First, the tools are good. The C++ support in the GNU Compiler Collection (GCC) is being actively cleaned up, with binaries getting smaller and version 3.2 offering a stable application binary interface (ABI) that will help with deploying software written in C++.

Besides GCC, a lot of other good tools are available to help with C++, as Cal Erickson points out on page 34. Cal does embedded development and that article is in the Embedded section, but his development strategy—get as much as possible working on your workstation first—means that the article will be useful to any C++ programmer.

Second, new or newly freed C++ projects, such as the Xerces XML parser contributed by IBM, mean that your new C++ code can draw on a lot of already tested, supported functionality. See John Dubchak's article on page 50 for an example. As more corporations start sharing in-house code, corporate technical preferences such as C++ start to be more important on the outside.

If you're looking for a place to apply your software development skills, Len Kaplan has a great one—your local museum. On page 89 he covers the unique challenges and rewards of creating applications for museum exhibits. And, you might pick up some C++ and XML hints from that article, too.

Another school of thought favors doing object-oriented programming in C. For an example of how that is happening in the kernel, see Greg Kroah-Hartman's Driving Me Nuts column on page 28. Your brain is inside your head, so people won't see the stretch marks on it from reading his code.

But speaking of brains, don't worry. We couldn't let the development issue slip by without at least one regular C article, and you'll be happy to know that the performance-critical parts of interpreting the brain waves of the test subject on the cover are in C. Enjoy Sam Clanton's Matlab-to-C porting advice on page 56.

Don Marti is editor in chief of *Linux Journal*.

Advice, Gift of the Open Source Community

Heather Mead

Issue #110, June 2003

We sometimes may give newbies a hard time, but not everyone can know everything.

Last month we talked about the do-it-yourself tenet of the open-source philosophy. Closely linked to that one is this month's topic of discussion, advice. One of the best aspects of the community is the willingness to share experiences with one another, which can be especially gratifying if you are undertaking a new project. Without a doubt, one of the most popular article types on the *Linux Journal* web site—both most-read and most-submitted—is the how-to article.

Early in 2003, Jay Docherty began a series of articles detailing the steps to take when one decides to install Linux on a laptop. Indeed, if you want to run Linux on a Dell or Compaq or other major-vendor laptop, you're going to have to put it there yourself. His first article, “Advice for Buying a Linux-Compatible Laptop” (www.linuxjournal.com/article/6684), offers some points to consider when you're buying a new laptop on which you want to install Linux. In part 2, “Setting Up a Base Linux Install on a Laptop” (www.linuxjournal.com/article/6742), Jay explains how to install Debian Sid and compile a custom kernel for your installation.

Alternatively, if you want to have a full Linux workstation but can't and don't want to spend a bundle, take a look at Glenn Stone's article, “Roll Your Own $450 Linux Box” (www.linuxjournal.com/article/6668). From the case to the video card to the CD-RW drive, Glenn offers suggestions for quality inexpensive hardware that will build a budget version of the Ultimate Linux Box. Of course, our readers had comments and advice of their own, so be sure to catch their postings at the end of the article.

Our most popular article so far this year has been Greg Kroah-Hartman's “Time for Users to Start Testing 2.5” (www.linuxjournal.com/article/6740), in which

Greg asks you, our readers, to help him and the rest of the kernel team test the development kernel. The comments quickly blossomed with questions and advice on how to get 2.5 working, so look for more testing and bug reporting help on the site as 2.6 draws closer.

Our web site maintains all of its articles, new and old; so if you do a little exploring, you might come across an article outlining the process of “Setting Up a VPN Gateway” (www.linuxjournal.com/article/4772). Duncan Napier explains how to “install and run an IPSec-based VPN gateway with a firewall using a single bootable Linux diskette distribution”. If you'd like to set up a secure internet connection between your home system and your work LAN, this one should help you do exactly that.

If you have some project advice or experience you would like to share with others, perhaps saving them a few missteps along the way, send your ideas to info@linuxjournal.com. New articles are posted on the web site every day.

Heather Mead is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #110, June 2003

PRISMIQ MediaPlayer, Red Hat Enterprise Linux ES and WS, SnapGear PCI630 and more.

PRISMIQ MediaPlayer

The PRISMIQ MediaPlayer is a set-top device that plays and displays media files from home computers, connects the TV/entertainment center to the Internet and acts as a platform for broadband services. The PRISMIQ includes an NEC uPD61130 32-bit MIPS microprocessor with an integrated MPEG decoder, 16MB of Flash ROM and 64MB of SDRAM. Network interfaces include 10/100 Ethernet on an RJ45 jack and a cardbus/PCMCIA slot for wireless. It currently supports MPEG-1 and MPEG-2 video formats, the MP3 audio format and the JPEG, GIF and PNG graphic formats. Output interfaces include one S-video, one composite video, one S/PDIF and two RCA Audio (L/R Stereo). The MediaPlayer is built on Linux 2.4 and comes with software for web browsing and an optional wireless keyboard.

Contact PRISMIQ, 2121 South El Camino Real, 10th Floor, San Mateo, California 94403, 866-774-7647, sales@prizmiq.com, www.prismiq.com.

Red Hat Enterprise Linux ES and WS

Two new releases from Red Hat, both compatible with Red Hat Enterprise Linux AS, formerly Red Hat Advanced Server, assist company-wide Linux installations. Red Hat Enterprise Linux ES provides an OS for a range of entry-level and departmental duties, including network, file, print, mail, Web and custom or packaged business applications. It is designed for smaller systems with up to two CPUs and 4GB of main memory, and it comes in Basic and Standard Editions. Red Hat Enterprise Linux WS is an engineering desktop/workstation. It is designed for use in client/server deployments, software development environments and targeted ISV client applications. Also available in Basic and

Standard Editions, WS provides support for workstation/desktop systems with up to two CPUs.

Contact Red Hat Software, 2600 Meridian Parkway, Durham, North Carolina 27713, 888-733-4281, www.redhat.com.

SnapGear PCI630

The SnapGear PCI630 is a VPN firewall PCI card that offloads all firewall and VPN processing duties from the host computer to the card, allowing remote management, high security and simplified installation. An isolated, stateful firewall PCI device, the PCI630 provides onboard multi-VPN capabilities for secure access and communication in a NIC PCI footprint. For use on servers and desktops, the PCI630 includes 10/100 Ethernet connectivity, 4MB of Flash memory and 16MB of RAM. It supports authentication up to 2,048-bit for RSA key signatures, X.509 certificates in DER and PEM formats and multiple subnets, without third-party client software or per-user licensing restrictions.

Contact SnapGear, Inc., 7984 South Welby Park Drive #101, West Jordan, Utah 84088, 801-282-8492, contact@snapgear.com, www.snapgear.com.

5070 PC/104 CPU

Octagon Systems has released the 5070 PC/104 CPU, an integrated, PC-compatible, single-board computer (SBC) for thin-client and other network-enabled applications. Utilizing a low power 5x86 class processor, the 5070 can operate in temperatures from -40° to 85° C with little ventilation. It can be expanded using the PC/104 or ISA connectors. The 5070 includes two RS-232/422/485 serial ports, a 10/100 Base-T Ethernet port, two USB 1.1 ports, FDD, HDD, back-drive protected parallel and keyboard ports, CompactFlash and removable memory up to 2GB. SVGA CRTs and flat-panel displays are supported. Fast-boot Phoenix BIOS provides for operation in less than six seconds, and a boot image is stored in serial EPROM in case of CMOS battery depletion.

Contact Octagon Systems Corporation, 6510 West 91st Avenue, Westminster, Colorado 80031, 303-430-1500, sales1@octagonsystems.com, www.octagonsystems.com.

Visual SlickEdit 8.0

Version 8.0 of the Visual SlickEdit code editor is now available from SlickEdit, Inc. Visual SlickEdit 8.0 supplies a range of code editing tools that provide language and encoding capabilities for a range of languages and platforms. New for version 8.0 are directory-based projects, auto-updated distributed tagging, secure FTP (SFTP) and Section 508 accessibility for blind and vision-impaired developers. A new three-way merge interface extends the DIFFzilla file and directory tree differencing engine. Version 8.0 also provides increased support and capabilities for coders using JBuilder, Java, GNU C/C++ and CVS. A 30-day free trial is available on the web site.

Contact SlickEdit, Inc., 3000 Aerial Center Parkway, Suite 120, Morrisville, North Carolina 27560, 800-934-3348, www.slickedit.com.

Eventide VR778

The Eventide VR778 is a digital voice logging and archiving system, designed for use in health and public safety environments. Able to work as a standalone logger using a front-panel GUI or as a network server for PC workstations, the VR778 provides fault-tolerant features, such as dual hot-swap power supplies, fan assemblies and dual redundant hard disks. Anywhere from 8-160 analog and 16-96 digital record channels can be mixed in one VR778. Variable recording compression rates range from 13.3 to 16, 32 and 64Kbs. A mirrored RAID 1 system with dual-120GB hard disks (380GB RAID 5 optional) can record and store up to 19,800 channel hours at 13.3Kbs. DVD-RAM drives also are available.

Contact Eventide, Inc., 1 Alsan Way, Little Ferry, New Jersey 07643, 201-641-1200, www.eventide.com.

AMD Opteron Processors

AMD has announced that its eighth-generation enterprise class processor core, named Opteron, will use x86 64-bit technology. Providing high-level performance for both existing 32-bit x86 code and new 64-bit computing, Opteron is designed to support applications requiring large amounts of physical and virtual memory, such as high-performance servers, database management systems and CAD tools. Opteron also employs HyperTransport technology, a high-speed, point-to-point link for integrated circuits that reduces I/O bottlenecks, increases bandwidth and reduces latency. An integrated memory controller is used to reduce memory bottlenecks.

Contact AMD, PO Box 3453, Sunnyvale, California 94088, 408-749-4000, www.amd.com.