

# Re: compile+link Fujitsu Linux

---

*Source:* <http://coding.derkeiler.com/Archive/Cobol/comp.lang.cobol/2008-02/msg00009.html>

---

- *From:* Robert <no@xxxxxx>
  - *Date:* Thu, 31 Jan 2008 21:23:03 -0600
- 

On Thu, 31 Jan 2008 11:47:51 -0800 (PST), Richard <riplin@xxxxxxxxxxxxxx> wrote:

On Jan 31, 7:23 pm, Robert <n...@xxxxxx> wrote:

On Wed, 30 Jan 2008 20:46:28 -0800 (PST), Richard <rip...@xxxxxxxxxxxxxx> wrote:

On Jan 31, 4:38 pm, Robert <n...@xxxxxx> wrote:

```
when I try to execute I see
my first DISPLAY and then
it crashes:
BEGIN MYMAIN
cobol-rts.: HALT:
JMP0015I-U
[PID:0000763D
TID:002516C0] CANNOT
CALL
PROGRAM 'MY
SUB1'. ./MYMAIN:
undefined symbol:
MYSUB1 PGM=MYMAIN
Aborted
```

```
I am running on Red Hat
Enterprise, and Fujitsu
support say they will
only support:
* Red Hat Linux 7.2, Locale
C
* Red Hat Linux 7.3, Locale
C
* Red Hat Linux Advanced
Server 2.1, Locale C
```

Re: compile+link Fujitsu Linux

I am hoping that someone here has figured out how to compile and link on less ancient versions of Linux.

Try '-l MYSUB1' (lower case el) on the compilation of MYMAIN. As written, MYMAIN has no way of knowing the library name.

I think that you fail to understand what 'Dynamic Load' means.

There is no Cobol syntax to specify a library name. You can only specify an entry point name. When you do a CALL, Unix doesn't search every library in the library path. That would be hopelessly slow. It searches only the libraries named in the executable's ELF header.

Libraries normally contain many programs and entry points. They are not one-for-one like the example here.

[the usual insults snipped]

With Fujitsu Cobol the DLOAD directive tells the system that CALLs might be dynamic. In these cases there is no need for the library containing the called routine to be known to the ELF binary, there is no need for a -l, nor for the library to even exist at link (ld) time.

Normal Unix binding, outside Cobol, is to enter the names of called libraries in the executable's header, using -l. Doing so does NOT mean they are statically linked as the term is understood by mainframers and others from the Old School. It means the library files will be loaded (if not already in memory) when the executable starts. If any cannot be found, the program will not start, which guarantees the program will not abort mid-run because a called program is missing. It also allows useful diagnostics to be run before starting a long batch job.

A Windows dll works almost exactly the same as a Unix library (.so).

Re: compile+link Fujitsu Linux

## Re: compile+link Fujitsu Linux

Cobol can, and I believe should, use the normal Unix binding mechanism. There are several advantages and no good reason not to. In addition to the advantages given above, one specific to Cobol is non-proliferation of program files, which I'll explain below.

Dynamic binding is the alternative. A running program can ask the OS to load an additional library by specifying its FILE NAME. Technically, this is done with a call to `dlopen()` (or `LoadLibrary()` on Windows). The OS first checks to see whether the library is already loaded in memory, either because it was in the program's own header or because some other running process is using it. If not, it searches the library path looking for the file name and loads it. In either case it returns a handle to THE FILE that the program can use to search for a specific entry point or the file's default entry point (with the `dlsym()` function).

The problem with dynamic binding in Cobol is that Cobol has no concept of library FILE NAME, it only knows about entry point names, which are essentially the same as program names. Thus, Charles' application with "hundreds of COBOL programs", if packaged for dynamic call, will have HUNDREDS OF LIBRARY FILES, each named `libPROGRAM.so`. I find that ugly and, from a deployment point of view, clunky and amateurish. It would be cleaner to package his hundreds of programs into one or a few libraries that are linked to the main executable.

How many third-party apps have you seen that came with hundreds of libraries or dlls?

At run time a dynamic call will look for a file called `libNAME.so` along the `LD_LIBRARY_PATH`, where NAME is that in the CALL, and it will load that and find the required entry point.

It will first look for the symbol NAME already loaded. If not found THEN it will load `libNAME.so`.

As you should be able to tell from the results in Charles' 2nd message your BAD and WRONG advice has meant that his program does not work as he expected. He statically linked the dynamic library which will cause the CANCEL to fail to unload it and thus a reload does not find a new copy but reuses the old.

Wrong. If Cobol issues a `dlclose()` on `libMYSUB1.so`, even though it was 'statically' linked with `-l`, it will be unloaded .. and reloaded the next time he calls it.

If it is packaged in a library with a hundred other programs, the whole library might be unloaded, because the handle for NAME is the handle for `libMYSUBS.so`. Or Fujitsu might be smart enough to know it wasn't dynamically loaded. If he really needs initial values, he should use the Cobol INITIAL clause.

Incidentally, CANCEL is not guaranteed to unload the program your way, either. A dynamically loaded library can only be unloaded when its user count is zero. If another developer happens to be using the same called program, CANCEL will fail, and Cobol has no

## Re: compile+link Fujitsu Linux

way of checking for that. If you want to be hackerish, issue the CANCEL 100 times. Each call to dlopen() subtracts 1 from the user count. Then listen for cursing from the next cubicle. :) The reason Windows installers want you to reboot is because they don't have a reliable way of deleting old dlls they just replaced. The solution is to call FreeLibrary until it disappears from memory.

.