



Technical Articles

Troubleshooting RTS 173, Called program not found in drive/directory

Now that dynamic subroutine linkage at runtime is the norm, the "cob" command's invocation of the system's "ld" (linker) command does not often catch missing or, more often, misplaced subroutines at compile time. Instead, call name resolution is deferred until run time, where the symptom becomes an RTS 173 error. Most of the time, the name of the called routine in the error message is sufficiently helpful. Other times, the name is unfamiliar or the name string is empty.

The most common RTS 173 problem brought to SupportLine is missing "something.so" (or .sl or .so.1, depending on the specific UNIX platform). The name is unfamiliar to the user, e.g. "libcobrts.so". As in this example, the callable "s"shared "o"bject (.so) not found is a component of the Micro Focus runtime system. The usual reason for the failure is that a particular environment variable is not set correctly.

The name of the variable is "LIBPATH" on IBM AIX, "SHLIB_PATH" on HP-UX, and LD_LIBRARY_PATH on all other UNIX platforms. This article will refer to it generically as "library path". As with PATH, the value of the library path is a series of directories separated by colons (":"). One element on the library path must be the "lib" subdirectory of the product to which COBDIR points. If "echo \$COBDIR" yields "/opt/lib/cobol/", then "/opt/lib/cobol/lib" must be an element of the library path. There should be no other "lib" or "coblib" directories from other Micro Focus Cobol products ahead of the correct one.

The most common way to lose the library path setting is to use the UNIX "su" command to change to another user. This is often done to gain root access. A side effect of the "su" command is to unset the library path variable. The value is lost. To retain the value across an "su" command, something similar to the following is necessary:

```
LIBSAVE=$LD_LIBRARY_PATH
export LIBSAVE
su
LD_LIBRARY_PATH=$LIBSAVE
export LD_LIBRARY_PATH
```

Sometimes the "export" command for the library pathing variable is omitted or mistyped in a user's .profile. The "env" command output will look correct. A good way to see whether the environment is set correctly is to use the "mfsupport" command. The output file is either "mfpoll" or "mfpoll.txt", depending upon product version. The first part of the file is list of exported variable names and their values. Checking that list may reveal a problem not shown by the "env" command.

Your own callable shared objects, compiled with "cob -z .." may be anywhere along the directory list specified by the library path. As with PATH, your current directory is not used unless it is an element on the value of library path.

A callable shared object may contain more than one subroutine. However, only one subroutine's name ("entry point") may be the same as the external name of the file. For example, callable shared object "group.so" may contain subroutines "a", "b" and "c". If so, a call to any of "a", "b", or "c" will fail until a call to "group" loads the object and thus reveals the names contained within. If it is not convenient always to call "group" first in a

program, the problem can be solved without source code changes. Write a trivial trigger routine whose name is unique, e.g. "trigger". The code would, at a minimum, be:

```
identificaiton division.  
program-id. trigger.  
procedure division.  
goback.
```

Build the new shared object with "cob -z trigger.cbl group.cbl a.cbl, b.c, c.o" (object code, C source, and COBOL source are equally acceptable). Compile all using applicatoins with an additional directive, INITCALL, in this case INITCALL"trigger". At run time, "trigger" will be called before the first statement in the procedure division is executed. That will cause a search along the library path to find trigger.so. The object will be loaded, "trigger" will do nothing but return control, and the load process will map entry points "group", "a", "b", and "c". Those entry points may now be called in any order.

Be cautious, however, with use of the CANCEL command and grouped entry point names. In the example above, if you call "b", then CANCEL it, the entire shared object and all its entry points (trigger, group, a, b, and c) will be removed from memory and unmapped. Thus it is possible that a CANCEL of "b" will cause an RTS 173 error when "a" is called later in the run. This is true despite the fact that "a" may have been called successfully any number of times earlier.

(c) Micro Focus.

SupportLine Newsletter is a free, monthly email publication for the developer community. Email all correspondence to the editor via SupportLine.Newsletter@microfocus.com