Battling the Y2K

BY SAM GOLOB

The "magic deadline" of January 1, 2000 is quickly approaching, and as systems programmers, it's our job to make sure the bad subroutines have been replaced. To this end, I'd like to introduce you to a technique that once literally saved my company when we faced a similar situation when the clock struck midnight on January 1. As you already know, this is serious business and it's our business. It could cost your company many hours or days of production time, and possibly, for some people, their jobs.

THE SUBROUTINE REPLACEMENT PROBLEM

I faced this problem on January 1, 1991, when a time-dependent subroutine that did date conversion for our company's production processing failed. The consequences of the failure were catastrophic — a user abend was produced by the subroutine when it didn't work. Batch programs that used this subroutine failed with the user abend. CICS and other online transactions that used this subroutine fared even worse. At that time, each CICS region used only a single TCB — remember? In other words, a failure of the subroutine in any transaction would bring the entire CICS region down. Moreover (and this is the kicker), this subroutine was directly linkedited into many of the company's batch and online production programs. We didn't know which programs contained it. There were upwards of 3,300 load modules in the batch load library, and there were more than 1,100 load modules in the online load library. Using "standard IBM methods," this problem would take months to fix! Can you imagine the production impact?

It didn't take long for a programmer to fix the actual subroutine, which was a relatively simple small assembler program. The big question was, how could we find and replace the bad version of that subroutine in all the load modules that contained it?

The "magic deadline" of January 1, 2000 is quickly approaching, and as systems programmers, it's our job to make sure the bad subroutines have been replaced.

IBM doesn't help. Load module names don't indicate any of the subroutine names, and ISPF member lists only list load module names, not any subroutine names. The standard "brute force" IBM solution would be to take an AMBLIST listing of all the load modules to find the presence of the bad subroutine (4,500 load modules?). Then you'd have to write re-linkedited JCL to include the new subroutine that would replace the bad one — by hand! How could you make sure the JCL was right? You could cause far worse production problems if it was wrong. You'd have to carefully examine the linkage editor report for each reworked load module. This sticky situation would take a long time to fix, and you'd never be sure that you didn't miss a load module somewhere.

I fixed the problem with complete accuracy in two hours flat on January 2, 1991, and I didn't miss one load module. Read on to see how I accomplished this feat.

SOLVING THE PROBLEM

My quick solution to replacing a bad subroutine in many places requires the installation of a product. The up-side, though, is that the product is free. And, for many of you, this product is already installed at your shops. I've also posted the latest source code for it on a web site so you can download and install a new version for yourselves. With this product in place, you can solve the "global subroutine replacement problem," and many other problems as well.

The product is the PDS program package, currently at version 8.5, which resides on File 182 of the CBT MVS Utilities Tape, or can be downloaded from http://members.aol.com/freepds. The CBT Tape may be obtained from the NaSPA office, and Volume 1 Version 4.0 of the NaSPA CD-ROM contains CBT Tape V.416. The CBT Tape is a huge conglomeration of systems programmer goodies that no MVS or OS/390 systems programmer should be without. There is a vendor-supported version of PDS, called STARTOOL, from SERENA International in Burlingame, Calif., that contains all the functionality of "free PDS" and much more. So, if your installation is licensed for STARTOOL, you don't have to install anything else.

The PDS package works as follows: PDS is a TSO command that can be used with or without ISPF. You point PDS at a dataset, and you are presented with numerous choices of what to do afterward. PDS, even the free version, has perhaps 50 separate subcommands that allow more than 1,000 separate utility functions to be done to the pds or to some of its members. PDS automatically treats "load module" datasets (RECFM=U) differently from "source type" datasets (RECFM=FB or VB).

The PDS package (and, of course, STARTOOL) has a powerful facility for keeping track of "member subgroups" of the current partitioned dataset it is pointing at. One of

the main purposes of having member subgroups is to be able to perform an action on all the members of the subgroup as if they were one member. In other words, PDS has the capability of dealing with all the members from the current subgroup as one unit. You just refer to an asterisk '*' instead of the member name and a selected operation will be performed on all members of the current "member subgroup," instead of just one individual member. This capability can help us very much.

Choosing the proper member subgroup in each load library is an important key to solving our sticky problem. To be specific, we'd like to isolate a subgroup of load module members that all contain a CSECT having a given name (the name of our subroutine). Then we can re-linkedit and fix only those load modules. In other words, we've got to find them, then fix them.

Sometimes, even that is not enough. Suppose there are two different versions of the subroutine floating around in our libraries with the same CSECT name. Or perhaps (in a tougher case) there are two entirely different programs with the same CSECT name. We might want to further refine our member subgroup to contain only one of those versions or programs but not both. Using the PDS package, that can be done also without much additional effort. We're getting an indication at this point that with the PDS command, we have a lot of power.

It remains, once we have a proper and complete list of load modules to rework, that we need to re-linkedit them and replace the version of the subroutine that is wrong with the correct version.

The PDS package comes up strong in that department also. PDS has a facility that allows it to look at a load module and generate accurate JCL and linkage editor control statements to re-linkedit the load module exactly as it was before. All the proper attributes are generated, such as reentrancy, refreshability, APF authorization, and so forth. Also, ORDER statements are generated by PDS in the linkage editor control statements so that the re-linkedited module has all its CSECTs in the same order as the original module.

It is now obvious that we have all the ingredients in place to solve our problem. Besides that, there are a few other considerations. If your load library is a PDSE, copy it with IEBCOPY to a pds and then you can use the free PDS package on it. Free PDS doesn't have PDSE support, however, STARTOOL does. If you're using STARTOOL for this, it doesn't matter if the library is a pds or a PDSE. The PDS package is written in assembler language, it runs at assembler speed (i.e., "quick"), and it has optimized I/O routines to fetch the members of any partitioned dataset with the greatest speed.

For most of this, we have Bruce Leland and Steve Smith to thank. They are the principal "enhancers" and builders of this product, and they maintain the vendor version, STARTOOL. John Kalinich is maintaining the free version of PDS at this time, and he's doing a great job, too. So far, you've seen that PDS can do this formerly tedious job very quickly, and now we'll look at the specifics.

Documentation about the CBT MVS Tapes can be found on the web at http://members.aol.com/cbttape.

CORRECTLY FIXING ONLY THE BROKEN LOAD MODULES

In fixing our "globally scattered subroutine" problem, we must first determine which libraries are affected. These, most likely, will be application load module libraries. First, you must determine the production libraries involved and then you should probably make the same changes to the programmer libraries and development libraries so that the old error will never work its way back into production again. If you have a change control system, you'll have to work within its restrictions and parameters. In any case, if you've anticipated the problem, you'll be able to be orderly. If not, you'll have to be just as orderly, but also fast.

Once you've got the list of libraries, you'll deal with them one at a time. For simplicity, we'll pretend that only one load library is affected. However, that usually isn't the actual case, but you'll just repeat the same procedure for all the libraries that you have to rework.

The first task is to determine all the load modules that contain the affected CSECT name. This will probably be the name of the subroutine you want to replace. PDS and STARTOOL both have a powerful method of finding only those load modules that contain a CSECT with a certain name. This is the "IF" subcommand of PDS, a little known but powerful tool. The purpose of the IF subcommand is to create a subgroup of members based on certain criteria. The IF subcommand will select a subgroup of members, starting from all members, or it will further refine a subgroup that was previously selected. Let's invoke PDS and point it to the proper library. Under TSO, you'd say: PDS 'library.name' to invoke the PDS command, and at the same time, to point it to the proper library. Then you deploy the IF subcommand and say:

```
IF : MODULE(csectnam) THEN(SUBLIST)
```

The IF subcommand will now look at the starting member group, which is the colon ":". The colon means "all members." From this starting member group, IF examines the structure of every load module, and, if it finds a CSECT matching the given (partial or complete) name you specified in the MODULE() keyword, it selects that load module for inclusion in the new "member subgroup." Otherwise, IF excludes the load module from the subgroup. So, we now know that after this operation every member of the current member subgroup contains a CSECT with the given module name. We've done the "find" part of the "find and fix" process.

Well, maybe not entirely. For example, sometimes, there might be two or more versions of that CSECT floating around in our libraries. To further select one of the two, we could do any of a number of things. One thing we could do is to find a character or hex string that exists in one of the versions but not in the other. This string should be so unique that it's not likely to be found in a different CSECT than the one you're dealing with. Under the PDS package, the FIND subcommand not only finds strings but it can select members for inclusion into a member subgroup based on whether a certain string is found in the member. FIND, when used in this way, works like IF. Remember that the asterisk "*" refers to all members of the current member subgroup.

Then, you can say:

```
FIND * /charstring/ MODULE(csectnam)
THEN(SUBLIST)
```

or

```
FIND * xhexstringx  MODULE(csectnam)
THEN(SUBLIST)
```

This operation will further refine the member subgroup, picking only from members of the former member subgroup that contain the string somewhere within the given CSECT. There are other ways you can pick and choose members. The ultimate goal is to make sure that you have a member subgroup that contains only the members you want and no others.

Now the re-linkedit part gets deployed. The PDS subcommand that looks at a load module and generates re-linkedit JCL automatically is the "MAP" subcommand with its RELINK keyword. So, once you've determined a proper member subgroup, you use the MAP subcommand as follows by typing in:

```
MAP * RELINK
```

This produces an output listing to the PDS LOG (under ISPF mode) or to the terminal (under line mode). That output may be written to a file using the CONTROL DSN() subcommand so it can be edited. CONTROL DSN() causes PDS to write its log output to a dataset. To cause PDS to write its log normally again, use the CONTROL NODSN subcommand. This brings us to the last step.

The re-linkedit JCL produced by the PDS program, assumes that the SYSLMOD library and the original library are the same partitioned dataset — the load library that PDS was pointing at. After PDS generates re-linkedit JCL for all the load modules in the member subgroup, you must edit each SYSLMOD DD statement to point to a different, newly allocated output library. This ensures that the re-linkedited load modules do not replace the original members immediately, during this re-linkedit operation. Only when these new load modules are tested and checked out can you copy them over the old members in the original library. You may not want to overlay the members in your original library, but you should use whatever "safe" production control procedures you have.

The resulting JCL usually needs only a JOB card to run it, once the SYSLMOD DD cards are fixed. But in our case, where we need to replace the bad CSECT, we need two more things. First, a repaired version of the subroutine, as a load module or as an object deck, must be placed in a library and a DD card must be inserted into each step of the re-linkedit JCL to point to that library where the repaired module resides. Then, an INCLUDE linkage editor control card that points to the name of the CSECT and the DDNAME referring to its library must be inserted before the main INCLUDE cards, so the new CSECT will replace the old one that has the same name. This must be done for each module you want to re-linkedit.

That being done for all the generated JCL, for all the load modules in the member subgroup, the JCL is run, and all the necessary load modules will have been found and fixed. Now we must test the new load modules and copy them to the original libraries. Or, you can run production from new libraries containing the newly fixed modules in addition to all the other ones.

The latest version of the free PDS package can be downloaded from the web at http://members.aol.com/freepds.

DETERMINING YOUR VULNERABILITY IN ADVANCE

It is important to look at all your production programs and to determine if any subroutines are used often during the processing. Once this is determined, you can do all of the IF processing in advance to find these subroutines. Then, you can generate the re-linkedit JCL in advance so that it will be ready in case of emergency on January 1, 2000. As you become more familiar with the process, it should be redone whenever there are significant changes to the production load libraries. In case any of these subroutines has to be fixed, you've got the mechanisms so the fixed versions can be put into place quickly and accurately, wherever they belong.

Just for reference, in a system module context, the SMP/E language is to have multiple LMOD entries for a single MOD. However, we're dealing with application load modules here, and we have to do this job without the help of SMP/E and its record-keeping.

I sincerely hope that you learn this technique. With the clock ticking, this technique is so important, that I cannot, in good conscience, neglect to teach it. You might be able to easily fix a program, but replacing it wherever it is found in production processing may be a far more difficult matter. Now, it doesn't have to be so hard. You've just got to know how to do it right. Good luck. I'll see you next month. ts

NaSPA member Sam Golob is a senior systems programmer. He also participates in library tours and book signings with his wife, author Courtney Taylor. Sam can be contacted at sbgolob@aol.com or sbgolob@ibm.net.