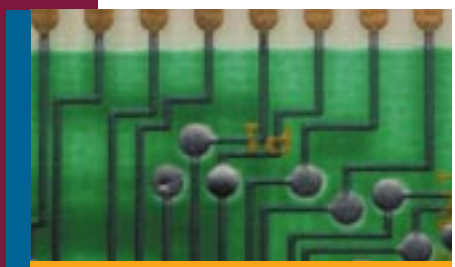

BY CRAIG COLLINS
AND DAVE CHRISTIANSON

# Implementing IBM's Language Environment: Part II — Assessing LE's Impact

This concluding article explains how the authors assessed the impact of the LE implementation at their site, examines some of the problems that surfaced during the implementation, and discusses solutions to those problems.

IN Part I (***Technical Support***, December 1997), we described what LE is, examined why you need to consider converting to LE now, and presented a couple of key issues regarding potential problems. This month, we explain what we did to assess the impact of the LE implementation, examine some of the problems that surfaced during our implementation, and discuss solutions to those problems.

## OUR ENVIRONMENT

To better understand our approach, some background is necessary. As we mentioned in Part I, our shop is the data center for the State of Wisconsin. We have a capacity of approximately 1200 MIPS and have about 4.5TB of online storage and 15TB of migrated storage. Our data center supports CICS 3.3, CICS 4.1, IMS 4.1, DB2 4.1, and IDMS 12.0.1. Due to the differences in the business requirements of the various agencies within the State of Wisconsin, we have a very diverse application portfolio. This diversity is compounded by the fact that the agencies were once spread among three different data centers with distinctly different systems management philosophies. As the consolidated data center has evolved, we have created a set of LPARs that separate systems programmer, development, production, and communications workloads. Our run-time libraries are in the linklists of each system. This made our conversion to LE easier since STEPLIBs did not exist in the JCL, and therefore did not need to be changed.

## SYSTEMS SOFTWARE AND LE

Before beginning to analyze your organization's application set for possible LE incompatibilities, make sure you check with the vendors of your systems software or purchased applications. It is very important that these systems or applications are compatible with the release of LE that you intend to install. While most software is compatible with LE, you may need to upgrade to a new release or install compatibility PTFs to some of the systems software. A few phone calls can save you a great deal of trouble during LE implementation.

## ANALYZING YOUR APPLICATION SET

At IBM presentations dealing with LE, IBM has stressed that an application inventory must be taken in order to solve both the LE conversion issue and the Year 2000 problem. While this sounds nice in theory, in practice it is extremely hard to accomplish. This is especially true for large or diverse data centers that support a number of separate application groups. While we applaud any organization that can actually accomplish such a feat, we looked for another solution to fit our environment.

## THE EDGE PORTFOLIO ANALYZER

Determining the compatibility of in-house applications can be a challenge. However, there is a tool that can be used to analyze load modules. In addition to being endorsed by IBM, the Edge Portfolio Analyzer (Edge Information Group, Des Plaines, Ill.) had a couple of features that were extremely useful in getting a handle on the LE conversion issue.

First, Edge can be used to create reports and/or a dataset with information about load modules and their different csects. This includes such items as time and date of compilation and linkage, language and language maintenance level used to create

the module, linkage parameters, and compile parameters. Based upon the items that IBM lists as compatibility issues for LE, the dataset that Edge produces can be used to find a subset of modules for which compatibility problems are expected.

Second, we were able to use Edge to produce link-edit cards for a module. This was very useful when we wanted to re-link a module with LE to determine whether a compatibility issue was resolved. For each of the run-time csects found in the original module, Edge produces REPLACE statements that can be fed directly into the binder (IEWL).

A third way that Edge was able to help us was through some of its canned reports. One of these reports shows which modules are not re-entrant. This report is very important due to the CICS re-entrancy issue described later in this article.

## SUBSETTING AND REPORTING

We used SAS to help us analyze the data produced by Edge. This gave us the flexibility to subset the data based upon certain criteria and produce reports for the application developers to review. To give the developers some additional data about these modules and help limit the scope of which modules could be effected by LE, we used some data from our accounting system to report whether a module had been used over a given period of time. The data from Edge has also provided us with the opportunity to do additional analysis when previously unknown compatibility problems present themselves or when compatibility issues we expected do not surface.

Using the reports created through the use of Edge and SAS, we were able to narrow the list of potential problem modules from a total of approximately 145,000 to 2,000 that needed further investigation. Besides helping the developers figure out where to focus, these reports were also useful in developing a timeline for the conversion and convincing management that the plan was sound.

## SMALL AND CONTAINED APPLICATION TESTS

We also performed some small application tests with a few of the application development areas. Our customers used the reports we had produced using Edge and SAS to determine which modules to test. Using STEPLIBs, they were able to run a representative sample of both modules that we reported as likely to fail and those we expected to work correctly under LE. These tests were extremely valuable to our analysis. We found that we interpreted a number of IBM's documented conversion issues too broadly. There were quite a few things that worked which we expected to fail. Based on IBM's documentation, we thought that all of our remaining OS/VS COBOL programs would need to be re-linked. As it turned out, most of our OS/VS COBOL programs worked just fine under LE. The few that failed did not work after a re-link and needed to be converted to COBOL II. The rest of the program failures we encountered were related to either PL/I or inter-language calls between COBOL II and PL/I. We were able to determine that re-links fixed most of these problems. This allowed us to further limit the scope of our analysis routines.

**Before beginning to analyze your organization's application set for possible LE incompatibilities, make sure you check with the vendors of your systems software or purchased applications.**

Besides helping us to further refine our analysis, the tests had other beneficial effects. Most important, performing the tests in conjunction with the application development areas gave both the systems programmers and the developers a better understanding of what was involved and what problems could be expected. It also helped to take some of the fear out of the conversion and provide a comfort level with one another that would definitely be necessary during the actual conversion. The tests gave all of us an opportunity to communicate about LE without the pressures that are inherent once production problems start popping up. Finally, it provided an opportunity to look at the potential problems and test the solutions we had developed, including re-linking modules with LE in order to develop a plan for future LE conversion issues.

## IMPLEMENTING LE IN STAGES

### Separate LPARs for Test/Production

Our operating environment was designed to allow changes to percolate up through various systems levels from a systems programmer environment to development and then to production. Using this methodology, we are more likely to catch potential problems before they affect production. For LE implementation this is a very important facet of our strategy. Changing the entire run-time for the third generation languages (3GLs) is a daunting task that has the potential to affect a very large segment of production work. We were able to test LE with systems products and plan for necessary compatibility changes at the systems programmer level, thereby eliminating those problems at the development level. Similarly, applications programs could be tested in development before moving to production. We moved LE right into the linklist at each level, replacing the old run-time libraries. We also had the startup decks changed for CICS and IMS.

We feel that this methodology was much more complete than trying to STEPLIB to LE. Besides eliminating the need for customer JCL to be changed in order to ensure that they were using LE, it also helped us identify application areas that were as of yet unfamiliar with LE and what we were doing. We were able to identify areas that were not going to be ready by our planned production date and develop contingency plans so that the production work would be able to continue.

We prepared for the production implementation by developing procedures to deal with the problems that were encountered during the testing. Besides having this "cookbook" for our own use, we provided the information to our customer's technical staff. Their staff prepared by identifying programs that were similar to those that failed in testing, fixing them during the production implementation with the methods identified as part of the testing. As a result, we estimate that less than 100 programs failed in production due to the implementation of LE, of which at most 10 programs were not working with LE within 24 hours of failure.

### Using STEPLIBs as a Temporary Solution for Batch Programs That Won't Work Under LE

As the first contingency plan for production, we had the customers STEPLIB to the old run-time libraries when production programs were abending after implementing LE. If it was an actual LE problem, using this method, the production work could be completed without interruption and the problem investigated further. This was also useful since some

things were initially blamed on LE that were not LE problems. Certain program failures occurred using the old run-times as well, the discovery of which turned the focus to the real problem with the program.

Most of the actual LE problems identified have been fixed and these additional STEPLIBs removed from those jobs. There are a couple of outstanding issues that will be covered later in this article.

## CICS ISSUES

Our migration to LE occurred at the same time as the initiative to change from CICS No-Protect to CICS Protect was happening in the test environments. This caused a bit of initial confusion for our customers due to the re-entrant requirement for VS COBOL II programs running under LE in CICS and the fact that some code was abending due to CICS Protect. These are two different issues with two different causes.

### Handling the CICS Reentrancy Requirement

The CICS reentrancy requirement is an issue that is introduced with LE. The LE run-time requires that VS COBOL II programs be re-entrant. Any VS COBOL II program that is run through the CICS precompiler will be compiled re-entrant because the precompiler inserts a CBL statement into the source. However, if the module also contains VS COBOL II code that was not compiled re-entrant, the program will fail with the IBM message:

```
IGZ0018S - On CICS, an attempt was
made to run a COBOL program that is
not re-entrant. The Program Name is
XXXXXXXX
```

The most common occurrence of this was with subroutines that are used for both batch and online. The problem is corrected once the program was recompiled and re-linked re-entrant.

### The CICS Protect Issue

While CICS Protect problems are not caused by LE, we feel it is important to explain this issue so that there isn't confusion between CICS Protect and CICS Re-entrancy. CICS V3R3 introduced read-only storage as a performance and security enhancement. Programs that are linked re-entrant are loaded into this area where they can be protected from inadvertent modification. This is great in theory but poses a problem in practice. Programs that are not re-entrant can be linked re-entrant. Unfortunately, CICS cannot tell if a module is really re-entrant, it can only look at the linkage options.

Assembler subroutines using the STM (Store Multiple) instruction are the main culprit. When a linked re-entrant program containing non-reentrant code is executed in a CICS region with the protect option on, the transaction will abend with a S0C4. This is an issue with conversion to CICS Protect but not an LE issue.

## PRODUCTION IMPLEMENTATION IMPRESSIONS

One of the problems we encountered was a CEE3204S Protection Exception, which appeared to occur right at the end of the COBOL II program when the files were being closed. This has been an intermittent problem that we're trying to resolve. As a temporary solution, the job using this program is being STEPLIB'd to the old COBOL II run-time library as a temporary solution.

Another problem we encountered in a COBOL II program was caused by a READ INTO statement on a file that had the RECORDSIZE EQUAL 0 clause specified. The READ INTO statement generates a MVCL (move character long) instruction that uses the beginning address of the WORKING-STORAGE field as the starting address of the move and the logical record length (from the 01 record element of the FD) for the length. The 01 record element was a 999-byte character field. The actual file had a 537-byte record length. In this case, 999 bytes were not left on the current page of working storage so the instruction spanned unrelated pages causing the S0C4.

This was not an LE problem per se. However, the program would not abend when we STEPLIB'd to the COBOL II run-time, so the conversion to LE brought this problem to the surface. After consulting with IBM, the code was changed from a READ INTO statement to a READ, and logic was added to move only the length of the actual 537-byte record, not the 999-byte logical record. The only explanation we could come up as to why this would work under the COBOL II run-time and not LE was that storage is allocated somewhat differently between the two run times.

We also had some PL/I programs abend with error message

```
USER COMPLETION CODE=4039 REASON
   CODE=00000000
IBM0201S ONCODE=81
```

This problem appears to occur when the DCB attributes of a PL/I output file are not specified in the run JCL or there is a mismatch between the definition in the JCL and the definition in the program. Specifying the attributes in the JCL corrected this problem.

Also, some load modules containing PLISHRE may need to be re-linked with the OS PL/I Library Routine Replacement Tool. You can find information explaining how to do the re-link in Chapters 4 and 9 of the *PL/I Compiler and Run Time Migration Guide*. Our best advice is to re-link abending modules that contain PLISHRE to a test library using the replacement tool and test it again before doing too much analysis. The re-link will fix most problems you encounter with this sort of module.

Overall, we were pleased that the LE implementation caused very few problems at our site. Of course, every installation has a different mix of applications, so your results may vary. However, our experiences were far better than what we expected based upon the IBM presentations we attended. The fire and brimstone preached by IBM certainly caused us to approach this implementation with caution, to complete the necessary analysis, and to develop detailed contingency plans. The results were a surprise, but a happy surprise, and that's the best kind. ts

*Craig Collins has been a systems software specialist at the State of Wisconsin for the past five years. Prior to this he was a systems programmer for four years at an insurance company.*

*Dave Christianson has been a systems software specialist at the State of Wisconsin for the past three years. Prior to this he was a financial applications developer for 13 years at several manufacturing and utility firms.*