June 2023

I recently completed participation in a "lift and shift" migration project. I would like to review the lift and shift methodology and provide my personal feedback on my lift and shift experiences.

SIMPLE LIFT AND SHIFT MIGRATION

- Identify the source code, data files, JCL, transaction processing environment, etc. from zOS.
- Copy / FTP the above to Windows
- Compile the source code on Windows
- Create a test environment
- Test and vet the "lift and shift" applications on Windows
- Promote and deploy to a Linux or Windows pre-production server
- Further test
- Promote to production

NO MODERNIZATION OF THE APPLICATION SYSTEM(S) IS CONTEMPLATED

Merely recreate the existing, **already functional zOS application** environment on Windows or Linux.

No conversion of COBOL to Java, C#, or whatever language du jour is popular. The zOS COBOL source will be recompiled as COBOL on Windows / Linux

Assembler programs will be an issue - more on that later. A reality check - "lift and shift" is not a simple process. Identifying and transferring zOS assets is a complex endeavor. Expert technical support staff / resources from zOS and Windows will be required. A capture plan (of the zOS resources) is a requirement. Successful execution of a thoroughly vetted plan will dictate the success of this portion of the lift and shift migration.

See Appendix "A" for a list of common zOS components that will need to be conveyed to Windows.

Given the extensive inventory of zOS components to be migrated it will be necessary to include in the project staffing, support specialists from the zOS environment. A need will exist to extract source files in total from archive / repository products such as Endevor, Panvalet, Librarian, IBM SCM, or whatever is in use on zOS. If the BMS / IMS macros are in a 3rd party dataset (for example GT SOFTWARE's BMS/TS Assist/TS) they must be extracted and presented as individual mapsets. JCL compile proc(s) perhaps have steps that require explanation by systems programmers

THE DOWNLOAD PROTOCOL MATTERS.

FTP, WSCON, or other transfer methods will translate the zOS components from EBCDIC to ASCII. Normally that is okay – except when the source code contains over punched literals.

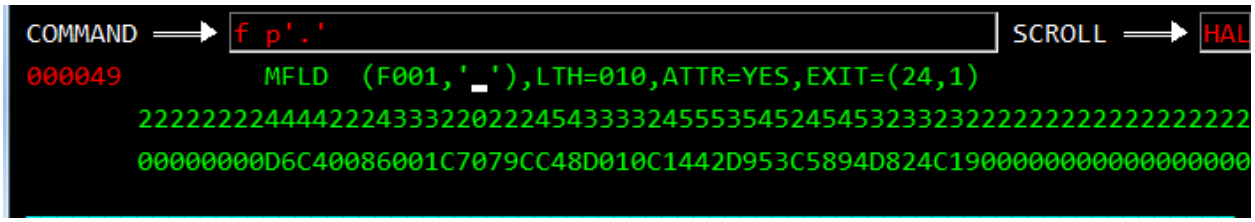
The origin of over punched characters is the ISPF editor. By turning hex on in the ISPF editor and then entering the hex value in zone over the decimal format.

Example:

```
MOVE  '3' TO SNAMEBX  (HEX 3F decimal 63 it is converted to HEX 1A decimal 26)
       F
MOVE X'3F' TO SNAMEBX   (proper method)
```

A scanner program must be run to detect characters outside normal EBDIC code points x'40' thru x'F9' One byte fields are more suspect than multiple byte fields - but not necessarily. The zOS ISPF editor can identify suspect fields with **FIND P'.'** as a command line.

The display below is using a Windows ISPF editor with the same command line, same results as zOS ISPF. Notice the vertical nibbles underneath the cursor. ISPF edit searched for and found a literal 0
0



NO MODERNIZATION

Literally means the source from zOS is already functional and in production executing on zOS. In theory every COBOL program conveyed to Windows will function as it does under zOS. No modifications are required (perhaps my optimistic viewpoint). There is absolutely no need for a fanciful Integrated Development Environment (IDE).

The popular Java and "C" based IDE(s)are foreign territory to professional zOS programmers. The learning curve of a du jour IDE for zOS professionals is long, tedious, and frustrating. The most basic of zOS tools is the ISPF Editor. SPF editing is not provided by any of the popular IDE(s).

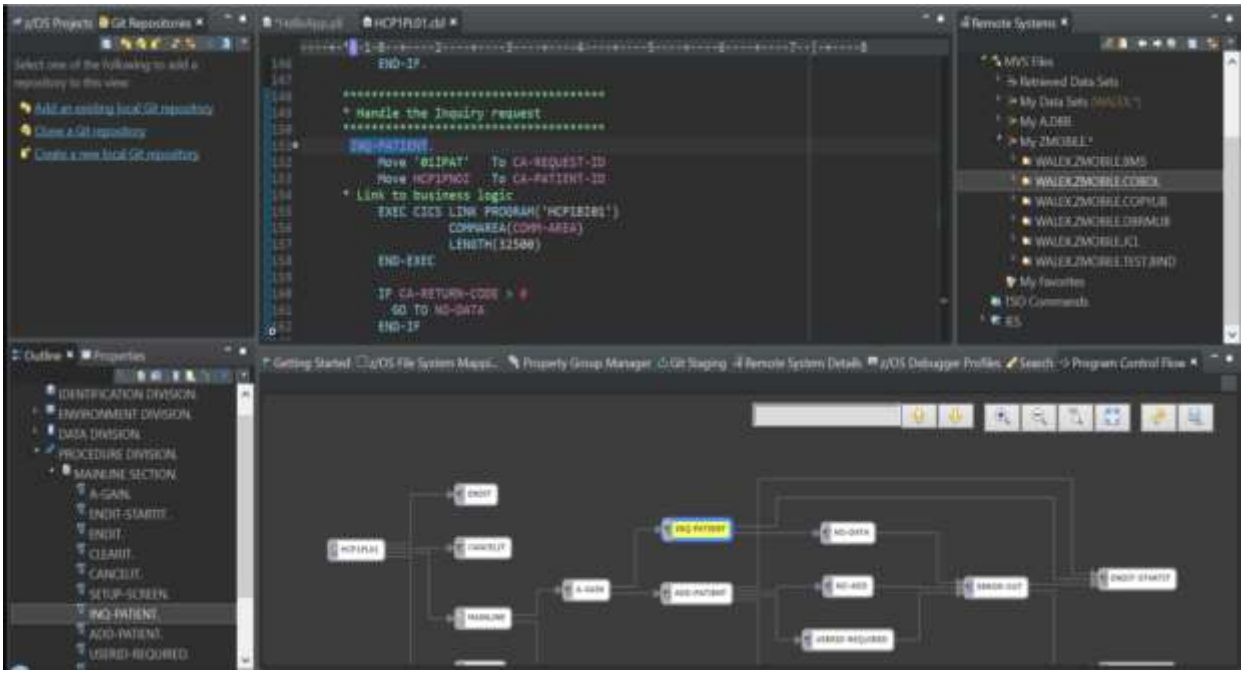
Most IDE(s) will impose an editor of their own design or suggest a "C" / Java centric 3rd party editor. Few if any of the editors offer the familiarity and power of the zOS ISPF editor.

Recall that this discussion is limited to lift and shift - stable functional zOS COBOL applications. As such, there is no need for these "improvements". The existing zOS development paradigm does not require such fanciful interfaces. There are plentiful IDE "solutions" in need of a practical problem to solve.

This is an example of a solution that solves some perplexing problem confronting lift and shift - Not!

"**When Show In > Program Control Flow** action is used, the node in the diagram representing that paragraph/section will be highlighted making it easy to identify the specific location.

Helpful in lift and shift? Probably not. Attribution: The below was copy / pasted from an IBM website.



KEEP THE DATASETS IN EBCIDC

Do not even think about converting terabytes+(petabytes?) of EBCDIC datasets to ASCII. The reality of converting from EBCDIC to ASCII is hookah based nonsense (my opinion obviously).

A lift and shift migration are not one big mammoth project. But rather a collection of applications.

Typically, In the zOS environment, dedicated IT professionals create, enhance, and maintain software by an application grouping.

For example; accounts payable, inventory control, sales order entry, billings, bookings. . . etc.

Subject matter expertise (SME) is developed within each application group. That expertise (SME) will be invaluable once testing commences. SME availability in virtually all phases of a migration is a de facto, important and real requirement for success. Segregate source components by application - a must do requirement. Avoid lumping 4,000 (or whatever) programs into a single migration project. Segregate by application.

SUMMARY SO FAR

zOS technical resources will have assisted in identifying the zOS location of components required by Windows. Windows technical resources will have created folders to harbor the various source components. JOBLIB, STEPLIB folders that will retain compiled and linked binaries have been allocated and populated. zOS components will have been downloaded - scanned by zOS or Windows utilities for over punched literals - then converted to ASCII.

The source files will be organized into zOS related projects (sales order entry, inventory control. . .) Compiles will have been successfully completed.

A review of the toolsets that are available for a lift and shift migration is appropriate. These and other activities will be discussed in the next chapter(s).

Wishing it to be a simple conversion doesn't make it so :-)Contemplate the impact of the next section.

CHALLENGES ON THE HORIZON

1. Assembler programs are referenced by COBOL programs (or perhaps assembler mains in JCL)
2. S806 - load module(s) not found (discovered while testing)
3. 3rd party zOS products are not available on Windows / Linux
4. Windows implementation of zOS utilities are missing functionality on Windows / Linux
5. SVC99 (DYNAM) support in Windows is insufficient when compared to zOS
6. SQL emulation lacks support for zOS column types - codeset collating sequence incompatible
7. DB/2 verbs used on zOS are not implemented in the selected SQL emulation
8. Windows based CICS / IMS emulation cannot accommodate zOS exits in Windows / Linux
9. Routine zOS utilities,3rd party utilities, and sub-systems not available on Windows / Linux
10. REXX / TSO / ISPF dialog manager services are not supported - for instance zOS ISPF panel 3.4
11. FILEAID is not supported
12. SPUFI is not supported
13. ABENDAID is not supported
14. SDSF is not supported
15. Xpeditor is not supported
16. Endevor is not supported
17. RACF is not supported
18. CLIST is not supported
19. EASYTRIEVE, DYL280, CA-TELON, Natural, are not available on migration platform
20. Database 3rd party products - ADABAS, DATACOM, IDMS, SUPRA, TOTAL is not available on migration platform
21. Forms control elements differ on Windows / Linux
22. zOS metrics from SAS reporting; SMF, RACF, IRRDBU, IRRADU . . . are not available.
23. A menagerie of other services used by zOS but not available on Windows / Linux

Presuming the migration can go forward regardless the challenges (or having resolved the challenges) the next objective is that of compiling the COBOL source that has been conveyed to Windows / Linux.

A simple but effective method is as follows:

1) Edit (if necessary) with an ISPF editor that is functional operating in Windows.
2) Compile
3) Resolve missing copybooks
4) Repeat 1 until a clean compile is achieved.

If the capture plan was concise and well executed the reiterative process above should be simple. There is absolutely no need to deploy analysis tools on a lift and shift. Recall the applications are functional in the zOS environment.

Using analysis tools to determine copybook "A" is consumed in "N" programs is a worthless metric. Ranking programs by complexity - worthless. Analysis now - worthless.

Once again, this is a lift and shift discussion of functional zOS production applications.

TOOL SETS

If "modernization" is lurking in the background as an objective, i.e., a graphical UI / HTML presentation, the agility hype, then my last choice of a programming language is COBOL.

CRITICAL CONSIDERATIONS USED IN SELECTING A LIFT AND SHIFT TOOL SET

1. Continue the use of EBCDIC datasets as opposed to converting all datasets to ASCII
2. Preserve 3270 presentation ("green screen", "crunchy UI")
3. JCL
4. CICS
5. IMS
6. DB/2
7. zOS Utilities

ITEM 1 – EBCDIC Limits the selection of a toolset vendor to only Micro Focus.

I am unaware of any other vendor that supports the processing of native EBCDIC datasets by a Windows or Linux COBOL or PL/I program. I do not endorse, I do not recommend, the Micro Focus Enterprise Developer product for a lift and shift project. I do endorse the Micro Focus Mainframe Express product. Albeit "retired" by Micro Focus.

The alternative to native EBCDIC datasets requires conversion of all datasets to ASCII. Conversion to ASCII requires that any USAGE DISPLAY (for instance PIC X(nn) PIC 9(nn)) data variable be translated to ASCII and retained in a new dataset.

My primary issue with conversion of EBCDIC to ASCII is that the translation must be context sensitive. A r**edefined** PIC X field, in a record definition, could be binary or a packed decimal (comp-3) data depending upon the **context** as determined by the executing program. The **context is determined at execution** time. **The context was not present if a utility converted all usage display data fields** in a dataset.

A simple example, an FD where the 01 level is redefined numerous times to reflect disparate record types. I have observed numerous implementations of this strategy. Some have redefined the FD 01 level a hundred times or more.

Not a best practice? Perhaps? Updating to a different strategy violates the lift and shift paradigm.

Another example, a record definition containing an OCCURS DEPENDING UPON clause. Typically used by RECFM=VB datasets. Without resolution of the object of the OCCURS clause (available only at execution) conversion of USAGE DISPLAY variables within the OCCURS clause is not possible. As the case would be in a static / utility conversion.

How is it possible to convert IMS database segments without an unload, convert, reload? How is it that 1000's of migrated (HSM) datasets can be converted? How is it that terabytes of data can be converted in a timely manner within a project cut-over time frame?

None of the above examples are problematic (or the severity is lessened) if the datasets remain in EBCDIC.

Other issues, are primarily concerned with DB/2 emulation. Binary data represented in zOS big endian form as opposed to little endian. zOS packed decimal format, comp-3, is also not a valid column type in some Windows / Linux relational database managers. Datasets containing arrays.

Extensive research is required in selecting a zOS DB2 emulator.

ITEM 2 - 3270

Enhancing a user interface, as would be the case where the application is a client / prospect / customer facing dialog (the retail Amazon shopping dialog is a great example of an enhanced dialog) such an enhancement is not germane to maintaining employee 3270 dialogs via CICS or IMS transactions. Lift and shift can only address existing CICS and IMS terminal presentations. There are some zOS applications where the front end (customer facing) dialog has been modernized to interact with a zOS backend. Lift and shift can only address the backend elements.

ITEM 3 - JCL

The ability to transparently utilize zOS JCL on Windows / Linux is another Micro Focus only feature. JCL is the zOS atomic basis of executing batch jobs.

To support JCL on Windows and Linux it is required that zOS dataset naming conventions be supported and understood. This is implemented by emulating the zOS system catalog and the services associated with the zOS catalog.

This is representative JCL that executes successfully on Windows or Linux

```
//AGDGTEST JOB 'WINZOS AIX SUPPORT',CLASS=A,MSGCLASS=A
//STEP1OF1 EXEC PGM=PARMTEST,PARM='0 WRITE GDG DATASET BIAS AT +1'
//SYSUT1   DD DSN=TSUBLUH.SNPOLICY.DATA3,DISP=SHR
//         DD DSN=TSUBLUH.SNPOLICY.DATA2,DISP=SHR
//         DD DSN=TSUBLUH.SNPOLICY.DATA1,DISP=SHR
//SYSGDG   DD DSN=TSUBLUH.Z.GDG(+1),DISP=(,CATLG,DELETE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=0)
//LISTDD   DD SYSOUT=*
//SYSUT2   DD SYSOUT=(,INTRDR)
//
```

Notice that the JCL has concatenated datasets, uses a GDG, and writes to the internal reader. In this example program, select records within the concatenated datasets will be rewritten. It is not sufficient to concatenate the datasets into a single file, a technique employed by solutions other than Micro Focus, because of the rewrite.

ITEM 4 - CICS

Multiple vendors for CICS are available. Most often the CICS implementation will preprocess EXEC CICS statements. The preprocessor will emit a COBOL based interface to their CICS Executive "engine". The preprocessor can perhaps be tailored to emit expanded COBOL that is supported by the COBOL that has been deployed for the migration. This is speculative on my behalf. Again, if the VSAM datasets are represented in EBCDIC, Micro Focus as the CICS vendor is the only choice.

ITEM 5 - IMS

I know of only one vendor of IMS support. Again, Micro focus. The MF IMS support is both batch and transactional.

ITEM 4 AND ITEM 5 CONSIDERATIONS

If transaction throughput is a concern, I highly advise creating a proto type test suite that exercises a target volume of transactions within a given time frame. It makes no sense to continue with a migration if transaction volume over time requirements cannot be achieved. The proto typing should simulate inserting a high volume of transactions, retrieving those transactions, and executing typical database I-O. I-O statements and the terminal queuing overhead dictate throughput. Computational instructions are minimal as compared to I-O statements and terminal I-O.

Creation of record level locking / release should be implemented. Production specified check points should be included in the test bed. Perhaps driving IMS or CICS via MQ with several transaction queues could function as the basis of a performance test bed.

ITEM 6 - DB/2

Many SQL relational database managers are available. Bear in mind that in most cases there will be a delta from the zOS behavior. I suggest the IBM product because it is the closest to zOS configuration and behavior. Conversion of EXEC SQL syntax to interact with the RDBM is done via a preprocessor.

The preprocessor will identify EXEC SQL expressions and convert the EXEC SQL to COBOL. The infused COBOL will be used to communicate with the RDBM executive program.

ITEM 7 - ZOS UTILITIES

This is a list of the common zOS supplied JCL initiated services. A subset of these services is more than likely to be found in the migrated JCL.

- DFSRCCO
- DSNMTV01
- DFSORT / SYNCSORT / SORT
- IEBGENER
- IEFBR14
- IDCAMS
- IEHPROGM
- IEBCOPY
- IEBPTPCH
- IEBDG
- IEBUPDTE
- IKJEFT01 or IKJEFT1A or IKJEFT1B

TOOLSET SELECTION CONTINUED

Selecting a toolset for a lift and shift migration will be dictated by the elements of what constitutes a lift and shift migration. Clearly, Micro Focus does the best job in meeting the criteria of a seamless lift and shift implementation. It is as seamless as it can be. The seam is interrupted by items indicated in the "Challenges" portion of this document. Also interrupting the seam are 3rd party products or zOS features and services not supported by Micro Focus.

My issue with Micro Focus is with Enterprise Developer (ED). ED imposes a significant learning curve. It is a monstrous implementation of what should be a simple edit, compile, and test paradigm. There is very little association with zOS development practices. Instead, ED relies on Visual Studio or Eclipse IDE(s) to effect edit, compilation, and testing. ED in no way represents the DevOp practices of zOS development, maintenance, and testing. It is a foreign world to many zOS professionals.

To put this in perspective, a very competent zOS professional recently endured 3 weeks of ED / ES formal training. After the training she was still novice (or worse) when attempting to compile and test zOS COBOL application programs using ED as the development tool. Her critique begins with the absence of an ISPF

editor and ends with the conglomeration of features in ED that are totally unrelated to zOS development. Personally, I will not participate in a lift and shift migration that utilizes ED as the development mechanism. I write programs in COBOL, PL/I, and assembler. I do not write C# or Java programs. I believe the advent of Eclipse / VS relieved Micro Focus of having to maintain a COBOL centric IDE. I further believe Micro Focus injected COBOL into the VS / Eclipse / ED realm, discarding their previous COBOL centric IDE. For zOS related development the ED "fit" is unnatural. So many ED solutions looking for a problem to solve. **All cluttering up the primary objective - edit, compile, and test a zOS COBOL program.**

Perhaps for new development ED might prove useful. But then again in a zOS environment I would clone (copy) a COBOL program that was closest in functionality to the objectives of the new assignment. Edit the clone to facilitate the requirements of the new assignment. Compile and test the cloned program.

I have also noted IBM promoting an Eclipse based "solution" for zOS development, IBM® Developer for z/OS® (IDz). I have yet to encounter a zOS site that has deployed IDz. Presumably there are accounts using IDz - but for a zOS lift and shift migration IDz is probably on par with ED.

The end product of an ED development effort is the creation of Windows or Linux binaries from zOS COBOL source programs. The binaries in the Micro Focus realm are promoted to Enterprise Server (ES). ES functions as the new zOS operating environment. Jobs are submitted via ES. CICS and IMS regions are started via ES. SDSF SYSOUT is managed via an ES component. The ES UI does not resemble zOS whatsoever.

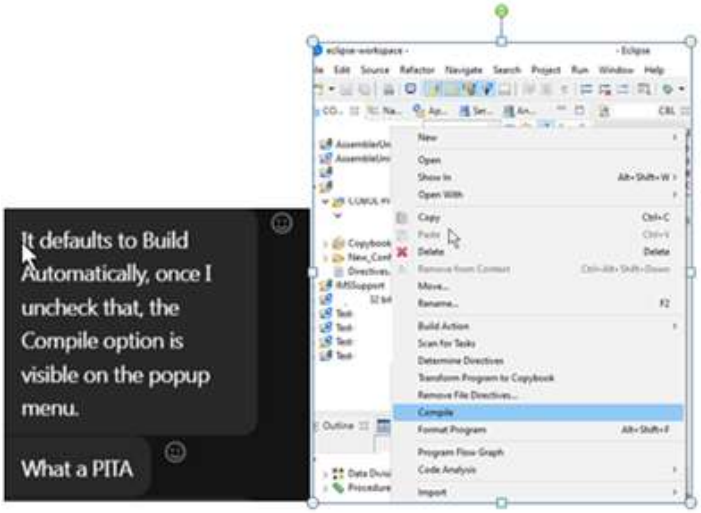
I personally use the Mainframe Express (MFE) product from Micro Focus. I edit with ISPF, compile and test with MFE. Unfortunately, Micro Focus retired MFE in favor of a progression of Micro Focus products that I speculate are geared to a du jour C# Java development paradigm. The transition from a focused mainframe product, MFE, began with Micro Focus Studio and morphed into ED and ES. I can't fathom why Micro Focus is not addressing the mainstream zOS user community and instead has created Micro Focus products that seem to address the du jour hype of the media.

Perhaps if you ask MF to provide MFE for your lift and shift migration they will relent and reluctantly make MFE available. Of course, with the caveat that MFE has been retired. I have trained zOS professionals in the use of MFE. The training time can be as little as 2 hours. A wee bit more with training on the use of Animator - the kinetic COBOL oriented debugger.

This chat session is worth sharing. It highlights my issues with Eclipse / ED. It is not a "fake" chat session.

This is an actual chat session with a team member versed in zOS and Windows development. She is attempting to use Eclipse / ED to compile a COBOL program. I did not prompt or solicit any of the chat session below.

I have redacted any information that would indicate the project (workspace?) and the team member. The EV / Eclipse dialog, in my opinion, is not zOS friendly.



SKILL SETS

Obtuse as obtuse can get.

A zOS professional has no idea what an ED RTS error 163 is.

A Windows professional has no idea as to what a JCL refer back error is.

Based on my experience and I guess what is a norm, the migration is often outsourced to a contractor. Rarely have I observed the migration effort being done in house. The contractor will staff the migration with experienced staff - true sometimes, sometimes not. This discussion will assume that the migration project has been outsourced to a contractor.

I have observed contactor staff requesting that all components of the migration be made delivered in one code drop. My example; 4,000 programs, 15,000 copy books, 1,000 job streams, etc. My preferred delivery of migration components is staged - by application. The staging project plan, a collaboration of the zOS personnel and the contractor personnel.

Deliver the application proto type components first. The zOS client determines what gets delivered, in what order, when? In theory it would be great. In reality, my suspicion is that an outsourced project is a fixed price contract.

Expedience will dictate the project plan. Irrespective of the project plan the following discussion is germane. On one hand there is a need for experienced Windows / Linux technical staff. On the other hand, a need for experienced zOS technical staff.

Early on in the project there is need of zOS systems support to assist in extracting source components from zOS. Just a few areas where system programming support may be required.

- Where are the production proc libs?
- Where are BMS / IMS macro stored?
- How do I retrieve all related source members from the zOS source control manager?
- Is FTP available and authorized?
- How do I download / FTP all members of a PDSE? (Hint use IEBPTPCH)

On the Windows side knowledgeable personnel are required to create the work spaces. ED is a complex install process. Especially when ED is to be used by multiple programmers.

Naming conventions for components should be compatible with ED. Naming convention should be standardized - for instance;

- .COB for COBOL
- .JCL for JCL
- .CPY for copybook members
- .BMS
- .MFS

The dichotomy:

- Windows personnel have no familiarity with client applications
- Windows personnel have no familiarity with client zOS conventions for batch and online
- zOS personnel have no experience with the Windows development tools

Imagine the communication issues in problem solving.

Unbelievably, ES under Linux, presented a need to add DISPLAY verbs to debug the promoted Linux COBOL programs. In March 2023, Linux ES still lacked an Eclipse / VS COBOL debugger. The recourse was to implement Circa 1980 debugging via the **insertion of DISPLAY / ACCCEPT statements**.

The Linux ES test bed was fully wrapped up in LDAP security. For application test purposes the security should have been minimal. As implemented, the effect was to extend delivery times significantly.

Ideally the contractor would provide Windows, Micro Focus, and zOS expertise. The client would make available zOS expertise and subject matter experts.

Lesson learned - Compile and unit test with ED only. Only use ES for string and acceptance testing.

ASSEMBLER PROGRAMS AND SUBROUTINES

It is a fact that some assembler to COBOL products can reliably convert problem state assembler opcodes to a COBOL expression. Problem state opcodes are those opcodes that require no special authorization. The problem state opcodes are the only opcodes that would be encountered in COBOL calls to an assembler language subroutine. Thus, 100% of application code is problem state code.

I am familiar with an assembler to COBOL conversion product. It is called A2C.I have used A2C several times with good results.

Conversion of the assembler opcodes to COBOL is only a part of the challenge. The ability to emulate, on Windows / Linux, the zOS services that are invoked by the assembler program adds to the challenge.

There are many services in COBOL that can resolve some of these zOS services. Many of the COBOL services are internal to COBOL but are documented in published API(s).

Two elements are required to resolve an assembler issue with a COBOL solution.

1. An understanding of what service the assembler program is calling out. What is the purpose?
2. An understanding of the COBOL API(S) available to satisfy the assembler service

An example assembler snippet:

```
COBOLVER EQU   *
         L     15,4(13)            GET CALLERS SAVE ADDR
         L     1,4(15)             GET CALLERS SAVE ADDR (COBOL2 RTS)
         L     1,456(1)            POINT TO DATE/TIME
         CLC   =C'WORKING',34(1)   AT WORKING STORAGE CONSTANT? OSVS
         BNE   NEWCOBOL            MUST BE COMPILED WITH COBOL2+
         MVC   36(20,2),0(1)       GET DATE TIME
         MVC   27(8,2),25(1)       GET PROGRAM NAME
         B     GOTCOBOL            CONTINUE
NEWCOBOL EQU   *
         L     1,4(15)             GET CALLERS SAVE ADDR (COBOL2 RTS)
         L     1,276(1)            POINT TO WORKING STORAGE
         CLC   =C'WORKING',10(1)   AT WORKING STORAGE CONSTANT
         BNE   NOTCOBOL            NO MUST NOT BE COBOL
         MVC   27(8,2),1(1)        GET PROGRAM NAME
         L     1,4(15)             GET CALLERS SAVE ADDR (COBOL2 RTS)
         L     1,256(1)            POINT TO DATE/TIME
         MVC   37(14,2),23(1)      GET DATE TIME
         MVC   54(2,2),LOCTYPE     LOCATION TYPE
GOTCOBOL EQU   *
         .
         .
NOTCOBOL EQU   *
         .
         .
```

Analyzing the assembler comments and the code I determined that the assembler code is retrieving the calling program name and the date / time the calling program was linked into the production library.

Windows does not implement the zOS linkage parameter convention L 15,4(13) and then chain to the callers save area. COBOL does not need the linkage convention to provide the same service as the assembler program. There is no COBOL support for zOS save areas.

Instead, there are 2 COBOL routines that will provide the required caller name, the date / time the program was linked into a Windows library.

THE COBOL EQUIVALENT

```
CALL 'CBL_GET_PROGRAM_INFO'
   USING
   BY VALUE        GPI-FUNCTION
   BY REFERENCE    GPI-PARAM-BLOCK
   BY REFERENCE    GPI-NAME-BUF
   BY REFERENCE    GPI-NAME-LEN
   RETURNING       GPI-STATUS-CODE.

CALL 'CBL_CHECK_FILE_EXIST' USING
                   SETUP-DSN
                   FILE-DETAILS
```

The CBL_GET_PROGRAM_INFO is called to find the name of the calling program.

The program name, returned from above, is used as a parameter to CBL_CHECK_FILE_EXIST, fetching the date and time returning when the called program was compiled and linked on Windows or Linux.

The same service the assembler program implemented can be implemented in COBOL. As you can see there is a need for zOS RTS expertise and zOS assembler language expertise.

There is also a need for COBOL internals expertise- expertise that can identify and implement the service required by the assembler program by using COBOL internal services.

COBOL can also support a fairly robust set of zOS control blocks. Or COBOL did, I do not know if the MFE support of zOS control blocks conveyed to ED. In order for COBOL to use the zOS control blocks (and their chaining) the Intel pointers / addresses must be in linear form. A Micro Focus COBOL directive MFPM. MFPM maintains a mainframe virtual address space to map mainframe-style above and below the line addresses to real 32-bit PC memory addresses.

Snippet of COBOL interacting with control blocks.

```
*     JOB NAME AND STEP NAME
*       PSA + X'21C' -> TCB -> TIOT
      MOVE X'0000021C' TO ZOS-TCB-ADDRESS-POINTER.
      SET ADDRESS OF TCB-POINTER TO ZOS-TCB-ADDR-POINTER.
      SET ADDRESS OF TCB TO TCB-POINTER.
      SET ADDRESS OF TIOT TO TIOT-POINTER.
```

Conversion of assembler programs to COBOL is feasible. It requires retaining personnel with the required expertise - zOS, assembler, and COBOL internals. Working alongside the A2C staff I have converted assembler programs successfully.

The A2C conversion process requires a code transfer of assembler programs and macros be made to the A2C staff. A2C will then run the assembler code thru complex translators that emit a COBOL program. The resultant COBOL program will compile successfully. However, some zOS service routines will be marked as a comment, indicating further analysis is required.

Common SYS1.MACLIB / SYS1.MODGEN members do not need to be provided - user written assembler macros must be provided.

Self-modifying code, example: code that changes opcodes, say add packed (AP) to subtract packed (SP) by moving a literal 'FB' to the original AP instruction ('FA') will require more A2C technical time.

The general architecture of A2C is in creating a set of 16 general purpose registers via COBOL pointer definition. These virtual registers are maintained in accordance to what was specified in the assembler program.

```
* POINTER REGISTERS 32 BIT ARCHITECTURE
       01  RP.
           05   R0P                    POINTER VALUE NULL.
           05   R1P                    POINTER VALUE NULL.
           05   R2P                    POINTER VALUE NULL.
           05   R3P                    POINTER VALUE NULL.
           05   R4P                    POINTER VALUE NULL.
           05   R5P                    POINTER VALUE NULL.
           05   R6P                    POINTER VALUE NULL.
           05   R7P                    POINTER VALUE NULL.
           05   R8P                    POINTER VALUE NULL.
           05   R9P                    POINTER VALUE NULL.
           05   R10P                   POINTER VALUE NULL.
           05   R11P                   POINTER VALUE NULL.
           05   R12P                   POINTER VALUE NULL.
           05   R13P                   POINTER VALUE NULL.
           05   R14P                   POINTER VALUE NULL.
```

A snippet of code illustrating the conversion paradigm

```
           MOVE A2C-AR1 ((R1N - AR1-IP + 1) : 1) TO
             A2C-AR15 ((R15N - AR15-IP + 1) : 1)*> MVC   0(1,R15),0(R1)
           ADD 1 TO R1N     *>LA    R1,1(R1)
           ADD 1 TO R15N    *> LA   R15,1(R15)
```

The resultant COBOL program will display the assembler as a comment adjacent to the COBOL statement.

Maintainability of the COBOL program from the conversion is possible. It requires an understanding of the A2C architecture as well as the ability to modify the COBOL code in accordance with the A2C representation and not break anything.

FINAL THOUGHTS

Lift and shift is a concept not a reality.

Availability of zOS SME(s) is critical. SME needs can be met on demand but must be timely.

Invest in creating prototypes of critical processing throughput criteria.

Micro Focus is the only solution if the parameters of the project are to maintain transparent COBOL, EBCDIC file system, transparent JCL, IMS.

Alternative solutions certainly exist. Albeit further away from the lift and shift concept. More than likely the source COBOL will be transpiled into Java.

EBCDIC dataset processing will probably give way to converting datasets to ASCII.

Regardless the vendor, more so than not, a lift and shift project will be over budget and late to deliver.

CONTACT INFORMATION

Email: development@winadc.com
Website: WINCOBOL.COM

OPINION DISCLAIMER

The views, experiences, feedback, and opinions expressed in this document are solely those of the author. Any content provided by the author are of his opinion and are not intended to malign any organization, company, individual or anyone or anything.

APPENDIX A

- COBOL
  - COPYBOOKS / INCLUDES
  - PL/I
  - Assembler
  - MACROS
    - PL/I
    - BMS
    - MFS
    - Assembler
  - CICS resource definitions (RDF elements)
    - Terminal definitions
    - PPT
    - PCT
    - FCT
    - PLT
    - TST
    - DCT
  - IMS Definitions
    - PSB
    - DBD
    - TRANSACTION ID to Program Name
  - DB/2
    - Include
    - DCLGEN
    - Plan
- Plus, many other configuration criteria
  - JCL
    - PROCS
    - SMS parameters
    - Control Card datasets for DFSORT / SYNCSORT
    - Control datasets FROM DD STATEMENT (SYSIN for example)
- MORE
  - MQ Queues
  - triggers
  - Others specifics and unknowns too numerous to mention