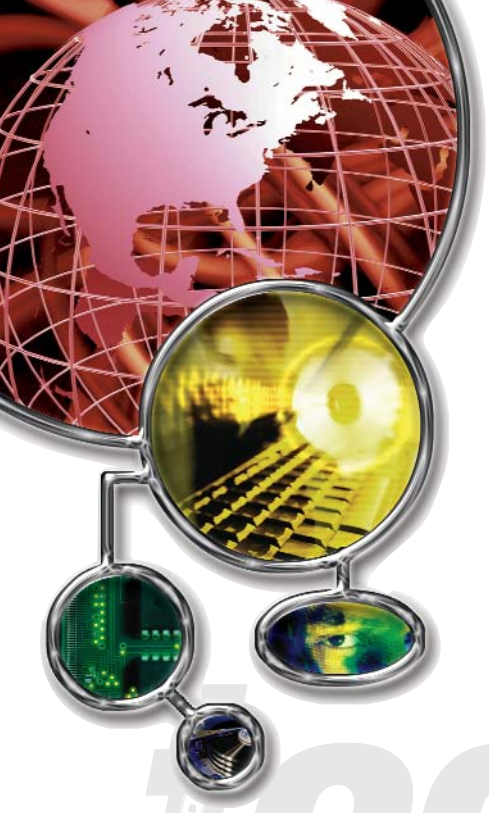

Automated Legacy Execution

By Bill Sweeney

I began working as an MVS System Programmer in 1981, while in the U.S. Army, assigned to the Department of Defense. I was trained by an excellent group of experienced System Programmers, who also taught me Assembly language programming. I was told that if you were able to program in Assembler, you would have a better understanding of the operating system and its components. Back then we had the source for MVS on reel tapes, and you could browse it to try and better understand the workings of MVS.

Automation on MVS consists of a few key components. The ability to trap messages is made easier through the use of the WTO exits. JES2 automatic command processing allows for time of day as well as interval processing for issuing commands and starting tasks. REXX as a high-level language provides a convenient method for performing System Programming type functions at a faster rate. The TSO CONSOLE command with REXX allows you to perform as an Operator, and to take action based on a particular situation. Combining these functions with some Assembler programs makes for a workable automation system.

Most of the Assembler code that I have written through the years has been more in the way of utility programs rather than application programs; programs geared toward improving the current environment or in expediting the migration to a new system. The automation of tasks and processes has been a specific area that I have been involved with in many organizations. I started in the 1980's with a program that initially would issue MVS commands using SVC 34, and continued to enhance this single program to perform other functions. I later wrote a Write to Operator (WTO) exit to trap messages and then use REXX to process these messages. As MVS advanced and provided more functionality, I was able to take advantage of these new functions and automate the system even more.

I replaced an off the shelf automation product with the utilities and procedures that I have developed, which are now running on a z/OS R1.4 system. The system is called ALEX, for Automated Legacy EXecution, and is named after my youngest child. The replacement was not on a large mainframe system, but on a smaller mainframe running a single production LPAR. With that said, I will describe the components of the system, and try to provide some detail as to how it all works.

The main component of ALEX uses the IBM supplied Installation Exit IEAVMXIT (described in detail in the IBM manual, Installation Exits, SC28-1753-05) as the vehicle for capturing and processing messages. IEAVMXIT gets control for every WTO and Write to Operator with Reply (WTOR) that is issued on an MVS system, as well as MVS commands. The IEAVMXIT exit program used by ALEX consists of two parts, which include: the executable code that performs a particular action and a Message Table that defines what messages to process, and how to process them. The executable portion of the exit is never modified, so the introduction of errors into the exit are minimized. The Message Table is link-edited directly with the IEAVMXIT exit program.

Even though IEAVMXIT is invoked for every WTO/WTOR issued by MVS, the impact on the operating system and applications is minimal. The exit searches through the coded Message Table using the system supplied message, and if no match is found then IEAVMXIT exits without having performed any other functions (e.g. GETMAIN to acquire storage). Further filtering is built into the Message Table entries to either select or exclude a message based on a search argument or a particular JOB name. All efforts have been made to build efficiency, functionality and flexibility into this one exit.

The use of the IEAVMXIT exit, coupled with how you define the messages to be processed, will allow you to perform automation of activities on your system. The ALEX IEAVMXIT will perform certain functions directly from the exit, to include: reply to outstanding WTORs, suppress messages, highlight messages, process multi-line WTOs, and issue commands directly to MVS. In the instances where further processing is necessary, the exit will issue an MVS START command, and pass the message to a Started Task (STC). The STC will then invoke a REXX EXEC to process the message.

An ISPF/REXX interface to define the messages that you wish to process and automate by IEAVMXIT was developed with the system, and allows you to dynamically add your requests. The ISPF/REXX

interface consists of the REXX EXEC ALXREXT, ISPF panels and messages, and a batch job to assemble, link and refresh the IEAVMXIT exit.

See FIGURE 1 for an example of the ISPF panel.

You can get an idea as to what is available for your message selection criteria, and the actions you can take by looking at the above panel. This panel will process multiple message requests. When complete, it will submit a job to assemble and link the MSGTABLE component, and then refresh the IEAVMXIT exit. Once refreshed, the messages added can be processed immediately, if the message is issued. There is a MESSAGE Macro included with the system that builds the internal structure of the message to be processed by IEAVMXIT.

In the event that the message cannot be handled within the exit, an MVS command is issued to start a task to process the message. The default STC name is OPSAUTO, and the first eight characters of the Message ID will be the name of a REXX EXEC to handle this message. OPSAUTO uses the IKJEFT01 TSO batch program, and the message to be processed is passed as symbolic parameters.

There are two limitations in the system. The issuance of multiple MVS START commands could flood a system if the same message is issued repetitively (e.g. 100 times a second). The exit has a built-in facility to not process the same message within a three second hard-coded limit. The other limitation in the system is that the MVS commands are limited to a maximum of 126 characters. The message passed, to include the S OPSAUTO, is truncated at 126 characters, so some information from the message passed could be lost. The exit will include the JOBNAME at the end of the message, if there is enough room.

The following is an example of the MVS START command issued for a message processed by IEAVMXIT:

```
S OPSAUTO,M1='DFHKE179 DFHKE1799 PROD
  TERMINATION OF CICS IS',M2='COMPLETE.
  J=PROD'
```

The message ID processed, DFHKE1799 is truncated to eight characters, and the SYSEXEC DD statement defined in the OPSAUTO STC contains a REXX EXEC named DFHKE179. The DFHKE179 REXX EXEC then processes this message and performs whatever action is necessary. This automation

FIGURE 1: ISPF PANEL

```
------------------          Automated Operations       ------------------
OPTION==>

      Add WTO/WTOR Messages for Automated Operations Processing

Message ID    ==>
   Enter up to 20 characters of the Message you wish to process

Action to Perform    ==>                 Highlite RED (Y/N)  ==>
   Valid Actions are REPLY, COMMAND, EXEC, EVENT or HIGHLITE

Reply    ==>
  If Action was REPLY then specify the REPLY to the Message

Command    ==>
  If Action was COMMAND then specify the COMMAND to issue

Search Argument    ==>
     ( Single Quotes  not allowed in Search Argument; Case sensitive)
Select if Argument Found  =>       Select if Argument JOBNAME  =>

Process Multi-Line WTO (Y/N/A)  ==>

PF1 - Help/Description of Facility    PF3 - End/Assemble    PF4 - Cancel/NoUpdate
```

FIGURE 2: HOW ALEXCMDS CAN BE USED TO SHUTDOWN A CICS REGION

```
* Shutdown test CICS and then wait 10 seconds
  F CICSTEST,CEMT PERF SHUT
#WAIT=10
*  Check to see if CICSTEST is running and wait another 30 seconds
%STRT=CICSTEST;#WAIT=30
*  Check to see if CICSTEST is running and issue next command
%STRT=CICSTEST;F CICSTEST,CEMT PERF SHUT IMM
*  If still running wait another 60 seconds
%STRT=CICSTEST;#WAIT=60
* If still running then cancel
%STRT=CICSTEST;C CICSTEST
```

system, like many others, relies heavily on the use of REXX. The replacement of the off the shelf package was possible because the existing REXX EXEC was written to support message processing, and required very little modification to work with ALEX.

It is possible to use REXX as an automation tool to function like an Operator with the use of the TSO CONSOLE command. The CONSOLE command allows you to issue MVS commands, and process the output of these MVS commands. The CONSOLE command requires definition to your security system, and should be limited to the OPSAUTO assigned Userid, and the personnel assigned to testing and defining messages to ALEX. REXX supplies the GETMSG function to process the output of commands issued using the CONSOLE command.

The following RACF commands are being used at one installation to establish access to the TSO CONSOLE command as part of their Disaster Recovery (DR) procedures. The first RACF command, RDEF, actually comes back as being already defined at the DR location. The RACF commands are:

```
RDEF TSOAUTH CONSOLE OWNER(SYS1)
  UACC(NONE)
ALU Userid OPERPARM(ROUTCODE(ALL)
  AUTH(ALL))
PERMIT CONSOLE CLASS(TSOAUTH) ID(Userid)
  ACCESS(READ)
SETR RACLIST(TSOAUTH) REFRESH
```

To further enhance the use of REXX I began writing Assembler REXX functions in the mid 1990's. I was doing analysis work for a DR contract, and had tried using REXX as a tool for performing the analysis. When it took over 20 hours to process all the data I realized that it was not practical to rely exclusively on REXX for this project and rewrote some of the analysis software in Assembler. It then ran in less than 30 minutes. After this project I started writing the Assembler REXX functions to address inefficiencies in REXX.

The following Assembler REXX functions are available with ALEX, to include:

ALXRWTO—issue un-highlighted and highlighted WTOs. Options are 'ROLL' to roll off the console, 'NOROLL' highlighted white, and 'NOROLLRED' as highlighted red.

```
WTOMSG = 'Test WTO message'
OK = ALXRWTO(WTOMSG,'ROLL')
```

ALXRWTOR—issue a WTOR, and provide an automatic reply after a specified time. The first argument is the WTOR, second argument is the max length of the reply, the third argument is the timeout value, and the fourth argument is the reply if the timeout value is reached.

```
WTORMSG = 'Reply Y or N if you wish to continue'
GREPLY1 = ALXRWTOR(WTORMSG,3,30,'Y')
```

ALXRDOZE—wait for a specified number of seconds

```
OK = ALXRDOZE(30)
```

ALXRDASD—return all DASD or TAPE UCBs with status information

ALXRQSCN—uses GQSCAN to return all enqueues

PDS Processing—various functions available for processing PDS files to include: reading directory entries, reading a member, writing a member, renaming, scratching or defining an ALIAS for a member. In some of the Assembler REXX functions the IBM supplied IRXEXCOM is used to assist in the storing and fetching of REXX variables. The subroutine ALXEXCOM is supplied to assist in the setup and invocation of IRXEXCOM.

The final component used in ALEX is the ALEXCMDS utility. I originally wrote this program during the mid 1980's to issue MVS commands using SVC 34. The program requires Authorized Program Facility (APF) authorization and placement in an APF authorized library. It uses control card input to determine what processing to perform. ALEXCMDS has been modified through the years to support other functions, including: issue highlighted and un-highlighted WTOs, reply to outstanding WTORs, search the Address Space Control Blocks (ASCBs) for a specific address space, issue STIMER waits, call IRXJCL to process a REXX EXEC, and perform simple scheduling using date and time processing and JES2 automatic commands. The simple scheduling component is rarely, if ever used, because almost all data centers have an Independent Software Vendor (ISV) scheduler product.

The ALEXCMDS program is used by a few data centers in the Washington, D.C. area. It is used to handle processing at IPL startup and at system shutdown. The ALEX automated operations system will use ALEXCMDS when it is convenient or at times when there are problems with the TSO CONSOLE command. It is a versatile program, with simple input, driven by specific information in column one. If there is no data in column one, or if there is not a valid ALEXCMDS character in column one, then control card is passed directly to the system as a command. The letter 'C' in column 72 allows for a single line continuation for MVS commands only (remember, an MVS command cannot exceed 126 characters).

Provided in FIGURE 2 are some control card examples as to how ALEXCMDS could be used to shutdown a CICS region. The asterisk in column one is treated as a comment by ALEXCMDS.

If CICSTEST were not running, then the first command wouldn't do anything and the first 10 second wait would be the only time taken. It is important that you understand your own system, and have reviewed SYSLOG data to determine valid time limits for your system. Once you understand the valid times, you can code the control cards for ALEXCMDS and use it during IPL startup or at shutdown.

The final item in ALEX is security. I have been able to get ALEX to work on z/OS R1.4, using both IBM® RACF© and Computer Associates® (CA) Top Secret©. CA supplies an option in their web support interface to ask questions about implementation. CA Top Secret was very helpful in providing the specific commands to help get the TSO CONSOLE command to work, and with some other automation questions. IBM was also helpful through the years dealing with RACF automation issues, and the IEAVMXIT exit facility. I once worked with an IBM Level 2 person on an IEAVMXIT problem, and walked through a dump with them. It was impressive as the Level 2 person described what was happening in the subpools.

The IEAVMXIT exit program does contain a RACROUTE REQUEST=VERIFYX to establish the security token used by the MGCRE Macro (MGCRE issues SVC 34). The use of REQUEST=VERIFYX on the MGCRE Macro is documented in the Installation Exits manual for the IEAVMXIT exit. I encountered problems when processing certain messages in the IEAVMXIT exit program (e.g. RACF WTOR ICH302D), that were resolved by coding the RACROUTE REQUEST=VERIFYX statement.

Getting many of the components to work required trial and error and a review of source code examples to determine how to accomplish a task. The SHARE tape and the CBT tape have been invaluable as research and teaching tools during my career. It has always been helpful being able to look at code to know exactly how something was done.

I have written many utility type programs, and have been somewhat remiss at getting them packaged up for the CBT tape. I do have examples of some of the code at the web site www.sscmainframe.com, though not the complete ALEX system. It is my intention to have submitted my software for inclusion on the CBT tape by the time this article is published.

Bill Sweeney is a Mainframe System Programmer that has been working as an independent consultant for SSC, Inc. for the past 15 years.