BY LIONEL B. DYCK

Using System Name/Token Pairs for Dynamic Access to Site-Specific Information

System name/token pairs can be used to make local information available to system or application programs simply and easily, and local information can be dynamically updated without an IPL.

WHAT is a system name/token pair? Simply stated, it is an area of storage that is managed by OS/390 services and contains a one- to 16-byte name with a one- to 16-byte token. The name can contain characters, integers, blanks, or an address. The token can contain any 16 bytes of information that you want to associate with the provided name. The token is accessed by specifying the name to be retrieved. The beauty of this is that OS/390 manages the storage allocation and access, allowing the programmer to concentrate on just using the information.

This article examines the system level usage of the name/token pair as a repository of job accounting information that is accessed by IEFUJV for validation purposes. There are three other levels available (task level, home address space, and primary address space); however, these levels are more limited and don't provide the functionality that was needed at my site.

System tokens have been available at least since OS/390 Version 1 Release 1. Connect to the OS/390 Internet Library web site (www.s390.ibm.com/os390/bkserv) and search for IEANTCR. You will get a match on 20 different publications if you search on the OS/390 V2R10 library. The IBM publications are excellent resources for coding information.

THE PROJECT

The reason that I was looking for the system name/token pairs was to solve a problem at my site. That problem involved a very sophisticated technique for validating correct job accounting. The technique used a table within a load module in the linklist that was loaded and then deleted by IEFUJV for every job. To update this table, the assembler source had to be updated, assembled, and linkedited into a linklist library, and an LLA update had to be performed to make the updates available. On the bright side, we now have the ability to do an LLA update because when this was written we had to wait for an IPL. One downside to this approach is that the linklist library needs to be compressed occasionally, which we only discovered after the linkedit failed. Another downside is the overhead of the LOAD and DELETE for the table load module from the linklist. The code in IEFUJV to search the table was OK, but the code could have been improved with a binary search; however, that overhead is small compared to the LOAD/DELETE overhead, so we decided not to bother with it. It was time to move into the 21st Century, or at least the late 1990s. The name/token pair processing routines reside in LPA so the overhead of the LOAD/DELETE is minimal. And, since the name/token information also resides in memory, there will be no DASD I/O to slow things down.

The logic that we used was much simpler and more dynamic than the logic used in the previous technique:

- 1. Create and maintain a sequential file or PDS member with the valid accounting information.
- 2. Write a program (started task) to:
 - read each record
 - delete the name/token associated with the account code
- 3. Create the name/token associated for the account code
- 4. Update IEFUJV to do a Name Get for the job's account code:
 - If the requested account token is present, then validate the information in the token.
 - If the requested account token is not present, the code is invalid.

By coding the name/token pair create/update program, the program can run as a started task immediately after IPL under

| (| FIGUF | re 1: Sampi | _e progra | | | N// S | ~.) |
|---|------------|----------------------|-------------------|---------------------------------------|------------|----------|-----|
| | 1. | | | SETCODE AC(1) ' | | | |
| | 2. 3. | TOKCREAT TOKCREAT | CSECT AMODE 31 | | | | |
| | 3. 4. | TOKCREAT | RMODE 24 | | | | |
| | 5. | | BAKR R1 | 4,RO | | | |
| | 6. 7. | | | 2,R15 KCREAT,R12 | | | |
| | 7. 8. | | | P=IEANTDL | | | |
| | 9. | | ST R | 0,IEANTDL | | | |
| | 10. 11. | | | P=IEANTCR D,IEANTCR | | | |
| | 12. | | | OKIN | | | |
| | 13. | GETTOKEN | | OKIN | | | |
| | 14. 15. | | | 5,R1 (R5),C'*' | | | |
| | 16. | | | ETTOKEN | | | |
| | 17. | | | AME,0(R5) | | | |
| | 18. 19. | | | OKEN,19(R5) TONAME,NAME | | | |
| | 20. | | MVC W | TOTOKEN,TOKEN | | | |
| | 21. 22. | | | KEY=ZERO,MODE=SUA 15,IEANTDL |) | | |
| | 23. | | | 15),(LEVEL,NAME, | RETCODE) | | |
| | 24. | | | 3,RETCODE | | | |
| | 25. 26. | | | 3,=F'04' REATE | | | |
| | 27. | | L R | B,RETCODE | | | |
| | 28. 29. | | LR R ABEND 1 | 9,R15 | | | |
| | 30. | CREATE | DS 0 | | | | |
| | 31. | | | 15,IEANTCR | | ETCODE) | |
| | 32. 33. | | | 15),(LEVEL,NAME, KEY=NZERO,MODE=PP | | (EICODE) | |
| | 34. | | CLC R | ETCODE,=F'0' | | | |
| | 35. 36. | | | BEND2 1,WTOA | | | |
| | 37. | | SVC 3 | | | | |
| | 38. 39. | ABEND2 | | ETTOKEN | | | |
| | 39. 40. | ADENUZ | | B,RETCODE | | | |
| | 41. | | LR R | 9,R15 | | | |
| | 42. 43. | EXIT | ABEND 2 DS 0 | | | | |
| | 44. | 2 | | TOKIN) | | | |
| | 45. 46 | | | P=IEANTCR P=IEANDTL | | | |
| | 46. 47. | | | 15,R15 | | | |
| | 48. | | PR | | | | |
| | 49. 50. | | EJECT YREGS , | | | | |
| | 51. | | EJECT | | | | |
| | 52. 53. | | IEANTASM LTORG | | | | |
| | 54. | IEANTDL | DS F | | | | |
| | 55. 56. | IEANTCR | DS F | (TEANT SYSTEM LE | | | |
| | 50. 57. | LEVEL NAME | | (IEANT_SYSTEM_LE' L16 | VEL) | | |
| | 58. | TOKEN | | L16 | | | |
| | 59. 60. | PERSOPT RETCODE | DC A DS F | (IEANT_PERSIST) | | | |
| | 61. | REFOODE | ORG , | | | | |
| | 62. | WITOA | DS 0 | | 2(0) | | |
| | 63. 64. | WTOA WTOMSG | | L2(WTOAE-WTOA),A 'TOKEN CREATION: | (U) • | | |
| | 65. | WTONAME | | L16 | | | |
| | 66. 67. | WTOTOKEN | | 'VALUE:' L16 | | | |
| | 68. | WTOAE | EQU * | | | | |
| | 69. 70. | TOKIN | DS 0 DCB D | F SORG=PS,RECFM=FB | RFC =80 | Х | |
| | 71. | | М | ACRF=GL,DDNAME=T | | X | |
| | 72. 73. | | END | ODAD=EXIT | | | |
| I | , | | END , | | | | |

SUB=MSTR, as well as whenever there is a need to update or add new account codes. The dynamic update can include just the new or changed codes or a complete refresh.

However, before modifying IEFUJV I needed to verify that I understood the name/token pair processing. To verify my understanding of this process, I wrote two sample programs, TOKCREAT and TOKGET, as shown in Figures 1 and 2. The JCL used to test these programs is shown in Figures 3 and 4.

Could this process be any easier? Not really. And the beauty is that OS/390 will manage the storage of our account table and will do the search for us. All we have to do is to create the name/token pairs and ask OS/390 to tell us if the account code (the name) is there. **Note:** This process could work for almost any application or product that needs to have a customizable set of options established before the application or product is used. (Wouldn't it be nice if the ISPF configuration options were managed this way?)

Now let's examine the sample code. For the implementation for job accounting the TOKCREAT will remain mostly unchanged. The TOKGET code will be changed to better integrate into IEFUJV.

GETTING STARTED

First, I went to the IBM OS/390 Internet Library (www.s390.ibm.com/os390/bkserv) and searched for "Token" and then narrowed it down with a search for "IEANCTR SYS-TEM." The search yielded three publications for my review. In addition, section 83.1.11 of the book, OS/390 MVS Programming: Assembler Services Reference Document Number GC28-1910-07, contains a great example of the TOKCREAT logic that I used in my programming.

THE TOKCREAT PROGRAM

This program will create the name/token pairs that will be accessed by the TOKGET program. The code is shown in Figure 1.

Statement 1 forces the program to be linkedited as authorized. Statements 2 through 6 are the initialization and base register set up. Statements 8 through 11 are where entry point addresses for the Token Create routine (IEANTCR) and the Token Delete routine (IEANTDL) are preloaded and saved for future use. **Note:** These modules reside in LPA.

- Statement 13 is the start of the routine to read the name/token pair information from the input card images. The card image is read using the Get Locate process with the address of the current record returned in register 1, which we then copied into register 5 in case we decide to insert code later to alter R1. A test is made for a comment record as indicated by an "*" in column 1, if so then that record is ignored.
- Statements 17 through 20 move the input information to the parameter lists for the name/token processing and for the WTO that will be issued if successful.
- Statement 21 changes the program to Supervisor State and Key Zero which is required for using the system level name/token pair processes.
- Statements 22 through 29 are used to delete the currently requested name if it exists. The call to the IEANTDL passes the level (of system), the name as provided in the current input record, and a fullword for the return code.
- Statements 30 through 38 are used to create the new name/token pair. The call in statement 32 passes the level (of system), the name as provided in the input record, the token value as provided in the input record, a persistence code (persist in this case so it will stay around after this program ends), and a fullword for the return code.
- Statements 36 and 37 issue a WTO if the create was successful, thus providing evidence in the system log, as well as in the log for the started task/job of what was created.
- Statements 39 through 42 are executed if the name/token pair creation failed.
- Statements 43 through 48 are the clean up and exit for the program.
- Statement 50 is the YREGS macro that maps the registers so that I can code R15 instead of 15 to reference the registers.
- Statement 52 is the IEANTASM macro, which provides the necessary equates for easily mapping to the name/token pair parameter lists.
- Statement 53 causes any literals used in the program to be generated at this point in the program.
- Statements 54 and 55 are where the entry point addresses for the

| FIGURE 2: SAMPLE PROGRAM TOKGET | | | | | | | | | |
|---|--|--|--|--|--|--|--|--|--|
| 1. TOKGET | CSECT | | | | | | | | |
| 2. TOKGET | AMODE 31 | | | | | | | | |
| 3. TOKGET | RMODE 24 | | | | | | | | |
| 4. | BAKR R14,RO | | | | | | | | |
| 5. | LR R12,R15 | | | | | | | | |
| 6. | USING TOKGET,R12 | | | | | | | | |
| 7. | LOAD EP=IEANTRT | | | | | | | | |
| 8. | ST RO,IEANTRT | | | | | | | | |
| 9. | OPEN TOKIN | | | | | | | | |
| 10. GETTOKEN | GET TOKIN | | | | | | | | |
| 11. | LR R5,R1 | | | | | | | | |
| 12. | MVC NAME,O(R5) | | | | | | | | |
| 13. | L R15,IEANTRT | | | | | | | | |
| 14. | CALL (15),(LEVEL,NAME,TOKEN,PERSOPT,RETCODE) | | | | | | | | |
| 15. | MVC WTONAME,NAME | | | | | | | | |
| 16. | MVC WTOTOKEN,TOKEN | | | | | | | | |
| 17. | LA R1,WTOA | | | | | | | | |
| 18. | SVC 35 | | | | | | | | |
| 19. | B GETTOKEN | | | | | | | | |
| 20. EXIT | DS OH | | | | | | | | |
| 21. | CLOSE TOKIN | | | | | | | | |
| 22. | DELETE EP=IEANTRT | | | | | | | | |
| 23. | PR | | | | | | | | |
| 24. | EJECT | | | | | | | | |
| 25. 26. 27. 28. 29. IEANTRT 30. LEVEL 31. NAME 32. TOKEN 33. PERSOPT 34. RETCODE 35. WTOA 36. 37. WTONAME 38. 39. | YREGS EJECT IEANTASM LTORG DS F DC A(IEANT_SYSTEM_LEVEL) DS CL16 DC A(IEANT_PERSIST) DS F DC AL2(WTOE-WTOA).AL2(0) DC C'TOKEN NAME: ' DC CL16' ' DC C'TOKEN VALUE: ' | | | | | | | | |
| 40. WTOTOKEN | DC CL16' ' | | | | | | | | |
| 41. WTOE | EQU * | | | | | | | | |
| 42. TOKIN | DCB DSORG=PS,RECFM=FB,LRECL=80,DDNAME=TOKIN,EODAD=EXIT, XXXX | | | | | | | | |
| 43. | MACRF=GL | | | | | | | | |
| 44. | END , | | | | | | | | |

FIGURE 3: TOKCREAT STARTED TASK JCL

FIGURE 4: TOKGET JCL

| //TOKGET // //HOLD //GETTOK //STEPLIB //TOKIN A#.CODE00 A#.CODE1 | JOB SYSTEMS, LIONEL DYCK', CLASS=V, NOTIFY=&SYSUID, MSGLEVEL=(1,1), MSGCLASS=X OUTPUT JESDS=ALL, DEFAULT=Y, OUTDISP=(HOLD, HOLD) EXEC PGM=TOKGET, REGION=2M DD DISP=SHR, DSN=SYSX.TEST.LOAD DD * | |
|---|---|--|
|---|---|--|

name/token pair create and delete routines are stored.

- Statements 56 to 60 are the information passed to the name/token pair create and/or delete routines.
- Statements 63 to 68 are the WTO parameter lists and data. Statements 70 to 72 are the DCB definition for the input file.

©2000 Technical Enterprises, Inc. Reproduction of this document without permission is prohibited.

THE TOKGET PROGRAM

This simple program, as shown in Figure 2, just reads in a file with the names of the name/token pairs that we want to display. The TOKGET program does not have to be authorized, but it does have to be AMODE of 31. The call to the name/token pair get (IEANTRT) routine in statement 14 must tell the routine that it is looking for a name that is at the system level and persistent. The name is passed along with a location to return the matching token.

Once the get request has been satisfied, the results are displayed using a WTO in statements 15 through 18.

THE JCL

The JCL in Figure 3 demonstrates how to create a name/token pair with the name A#.CODE1. Note that the name can be any combination of characters, integers, blanks, or an address. The A#. is used to designate that these are accounting names, and CODE1 is an account code id. The token value of SYSTEMS PROG is used as the name for the group that owns that account code.

The JCL in Figure 4 demonstrates how to obtain two name/token pairs by name. The request for A#.CODE0 will cause an error because this name does not exist. The next request will be successful.

IEF403I TOKCREAT - STARTED - TIME=15.26.28 TOKEN CREATION: A#.CODE1 VALUE: SYSTEMS PROG ACTRT01I TOKCREAT STEP CODE CODE = 00

IEF404I TOKCREAT - ENDED - TIME=15.26.28

FIGURE 5: RESULTS FROM TOKCREAT JOB



IEF403I TOKGET - STARTED - TIME=15.26.28 +TOKEN NAME: A#.CODE0 TOKEN VALUE: +TOKEN NAME: A#.CODE1 TOKEN VALUE: SYSTEMS PROG ACTRTO1I TOKGET STEP GETTOK CODE = 00 IEF404I TOKGET - ENDED - TIME=15.26.28

See Figures 5 and 6 for snippets from the JESMSGLG for the jobs in Figures 3 and 4.

CONCLUSION AND COMMENTS

Obviously the coding for test routines could be improved with mandatory comments. The use of WTO could be eliminated for the TOKGET and changed to output to a SYSOUT if desired. The code for the TOKGET also needs to be changed so that it will be reentrant as required for SMF exits and thus work within IEFUJV.

Remember that these examples are intended to demonstrate the process only and perform no other useful or productive function. I am confident that you can see the usefulness of this code for accounting validation as well as other uses for information that needs to be available during execution without the overhead of having to allocate and read a DASD file. If you are a software developer, then this technique can be used in lieu of providing a load module for your customer to modify for local customizations, and it is faster than providing a parmlib set up for local custom settings.



NaSPA member Lionel B. Dyck is a lead MVS systems programmer for a large HMO in California. He has been in systems programming since 1972 and has written numerous ISPF dialogs over the years. He is an active member of NaSPA and SHARE, and can be reached via email at Lionel.B.Dyck@kp.org.