APF Authorized TSO Commands

SHARE Session 7589

What we’re about - Ethics

- We are the “system doctors” of the MVS (z/OS) operating system. We have been hired to help the installation to function properly, and to stay up as much as possible. We are there to ensure that the installation receives as large a return on its investment, as possible.

- An employee must make more money for his/her employer, than he/she receives in salary. Otherwise, hiring him or her, doesn’t pay. (Think about it.)

- As such, we must be TRUSTED to have the tools of our trade, and to be trusted to use them properly. Since we may be called upon to have to “repair anything”, we need to have access to an APF-authorized load library for our own use.

- This presentation will show us how to make fuller use of these capabilities, to help your installation, and to help ourselves, too.

My 50-50, 100-100 rule

- When I started my first job, I realized two things:
- The benefit is split between me (the worker) and the employer (the installation).
- I get 50 percent, and the employer gets 50 percent. The employer gets 50 percent, which is the labor. I get 50 percent, which is the EXPERIENCE.
- But the real truth is: That the employer gets 100 percent of the labor, and I get 100 percent of the experience.
- This is a win-win situation. I have ALWAYS had this foremost in my mind, at every job I ever worked at.

Invest some of your time in learning new things

- Everybody has busy times and “less busy” times. My teacher Jeff Broido always encouraged me to invest a half hour a day to learn something new. By doing this, you expand your knowledge, and the installation will eventually reap the “interest benefit” that comes from your increased capability and knowledge.

- This learning (in my case) never impacted the timeliness with which I got my assigned tasks done. I always invoked the “50-50, 100-100 rule” when I was working at a job.

- The MVS (nowadays aka z/OS) operating system has many components that require knowledge and understanding, both about how each one works, but more importantly, about how they hang together. Of utmost importance in this job, is the need to form mental pictures about how the various and many components of the z/OS operating system fit together and work with each other. This knowledge is not easy in coming. It takes time and effort to learn about each of the MVS (z/OS) system components and about their relationships, both on a superficial level, and on deeper levels.

It pays to learn Assembler Language

- Assembler language for the IBM mainframe, is the closest language to the action of the machine. Programming in Assembler language, you can control anything that the machine does.

- If you don’t know Assembler language (yet), you can achieve good results using REXX, but you can’t get the benefit and control that Assembler language gives you in dealing with all aspects of machine operation.

- IBM helps you by providing system macros, and very considerable knowledge and tools, for an Assembler programmer to delve into controlling the system. In my years of experience, I have to praise IBM very much for always providing a lot of tools to access most of the internal components of the operating system. However, IBM has not been as good in providing actual system tools that do a job; in that department, they usually fall somewhat short. There is a good reason for this.

- IBM (internally) has to use their programmer resources to develop the system itself. Anybody who ever worked there, will realize that they could not use their valuable (and “well-paid”) programming staff to flesh out all the tools that us “system doctors” need in our everyday work. True, they give us some good tools themselves, but we ourselves have had to flesh out our own tool collections from either of three sources.

- Either: We buy vendor-supported tools (expensive and licensed for this installation only)

- Or: We write our own tools (OK for a few tools, but time consuming and you need the skill)

- Or: We use free tools that other people have written. This saves time, and has the advantage that each person who wrote a tool has taken the time to learn the requisite skills and do the requisite research. Why “re-invent the wheel” when good tools exist aplenty. And they are free. Where are they? They are in many places, but you can start with the CBT Tape collection (www.cbttape.org).

What is the CBT Tape collection?

- The CBT Tape collection is a free collection of user-written MVS tools which has existed since 1975.
- The CBT Tape collection was started by Arnold Casinghino, who managed it until September 1990. Afterward, the management was done by me.
- In October 1998, Sam Knutson and I started a web site: www.cbttape.org
- We have taken over the MVS SHARE Tape (which is now part of the “CBT Overflow Tape”. We have taken over support for many Xephon materials now, so they are more easily accessible to everyone. Thanks to Bob Thomas.
- The CBT Tape still exists as two tapes: The “regular CBT Tape” and the “CBT Overflow Tape”. If you wanted to write out these tape materials, you would need a 3490E at least, because of the length of the tapes. But the individual files from these tapes are all downloadable (in TSO XMIT format mostly, and zipped) from the www.cbttape.org web site.
- You do not need to be a “member of anything” and you do not need a “password” to use the www.cbttape.org web site.   It is up 24x7 almost always, so when you are doing your weekend upgrades, you can access the materials, almost always.
- Everything on the www.cbttape.org web site is subject to a Disclaimer, whose text is addressable from the home page. You use it at your own risk, but practically speaking, the whole world has tested much of this stuff, and you can too. It is usually VERY reliable, but the disclaimer still is there, and needs to be.

TSO commands in general, and APF-authorized TSO commands

- MVS is very helpful for programmers, in that its interactive portion, called TSO, can perform just as much work, in much the same way, as “batch processing”.
- A TSO command differs from a program, in that upon entry, Register 1 points to a CPPL (Command Processor Parameter List) instead of to a standard program’s PARM list.
- Most TSO work, and most batch jobs, cannot interfere with the “guts” of the system that is necessary for it to run.
- But APF-authorized TSO commands (as well as APF-authorized programs) can do just about anything to the system, either preserving it, or destroying it. We are here to preserve the system, and as “system doctors” we need APF-authorized programs and APF-authorized TSO commands as our tools.
- In order for a program to run as APF-authorized, it needs two things, and to run authorized under TSO, it needs three things. First, the load module has to be link-edited as SETCODE AC(1). This is a setting in the pds directory entry of the load module. Second, the load library has to be in the APF-authorized library list. This (nowadays) is determined by a listing in the PROGxx PARMLIB member.
- For TSO programs and TSO commands, there is a third requirement for a program or command to run APF-authorized. This is going to be the main topic of our discussion today. There are four tables that are accessed by your TSO session: They are named after their old CSECT names, from TSO of bygone years. Alternately, they are also named after their settings in the IKJTSOxx PARMLIB member.
- The four tables are: (IKJEFTE2 – AUTHCMD), (IKJEFTE8 – AUTHPGM), (IKJEFTAP – AUTHTSF), and (IKJEFTNS – NOTBKGND). For a TSO command to run authorized, its name (up to 8 characters) must be in the AUTHCMD list (or the E2 list, as we will call it). For a called program to run authorized under this TSO session, its name must be in the AUTHPGM (or the E8) list. For a program to be authorizable by the TSO Service Facility, its name must reside in the AUTHTSF (or AP) table of program names. Finally, if you want to specify that a TSO command can’t be run under TSO-in-Batch, as a batch program, you specify its name in the NOTBKGND (NS) table.

Where to the four Auth Tables reside?

- We can set the contents of the four TSO authorization tables in either of two places normally. Today, we'll show you how to do it for your own session only, in a third place.

- The first place is to code the program names in the IKJTSOxx PARMLIB member, under AUTHCMD (NAMES(…)), AUTHPGM(NAMES(…)), and so forth. These settings go into effect either at IPL time, or after a PARMLIB UPDATE(xx) TSO command is issued by an administrator, or after an operator command SET IKJTSO=xx.

- The second place is to assemble a load module called IKJTABLS, with CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS, and place it in an APF-authorized library, that your TSO session will STEPLIB to, in its LOGON procedure. You can see a sample assembly of the auth tables in CBT Tape File 185.

- Upon LOGON to your TSO session, these tables will be copied to your TSO session's address space, and pointed to by the LWA (Logon Work Area) which is a control block created especially for your TSO session. We shall refer to the "LWA-pointed-to" copy of these tables, as the Local Copy of these tables. Your TSO session, when trying to determine if a program or command should execute as APF authorized, refers ONLY to the local copy of the "auth tables". This is an important and very crucial fact in our entire discussion. Note this fact VERY carefully.

- At LOGON time, the TSO initialization routine called IKJEFTP1 will create the LWA and local copies of the auth tables and the TSO exit tables. These are in your TSO address space, in Key 0, Subpool 252 storage. Fields in the LWA will point to the addresses of the local copies of these tables that were created at LOGON time. If an APF-authorized STEPLIB with an IKJTABLS load module and the proper CSECTs (IKJEFTE2, etc.) were present at LOGON time, then the local copy of the tables is copied from them. If there isn't an authorized STEPLIB, or the CSECTs were missing from there, then the tables are copied from the common-storage copies which were created at IPL time (or PARMLIB UPDATE(xx)) time. Finally, if no AUTHCMD or AUTHPGM or AUTHTSF list was coded in the active IKJTSOxx member of PARMLIB, then the local tables don't exist, but the addresses pointed to, in the LWA, are those in the default IKJTABLS load module in LPA. (The IBM-supplied ones don't contain much.)

How can we replace our own Tables?

- Now that we know that our TSO session's auth tables, which are actually used to make the judgments, are in our own Local Copy in our own TSO address space's storage (albeit Key 0 storage), we can go ahead and create new tables of our own choosing, if our program is APF-authorized in the first place. A non-authorized program can't change Key 0 storage.

- What do we do? As a proof-of-concept, we should GETMAIN some Key 0, Subpool 252 storage, copy a bunch of program names into the area, in the format of each of the auth tables, and re-point the appropriate LWA field to the new storage block. Does the system honor the new set of program names and ignore the old set? You betcha. It certainly works that way. Now, the question is merely how to do this reliably, using a tested program or TSO command, copying the list from your own desired list of programs.

- Dan Dalby and I worked on this project last summer. I wrote two programs, called TSUB and LLWA, and Dan wrote two different programs, called LWATMGR and LWATEDIT. These programs can be found on CBT Tape File 797. Dan's programs, and my programs, do slightly different things, and of course, they are written very differently from each other since we two have different programming backgrounds. But they can reload these tables from each of 3 different sources: First, the source can be an IKJTABLS-like load module, second, an IKJTSOxx-like FB-80 Parmlib-like file, and third, from an LRECL=8 list of program names. In the third way, how do these programs (LLWA or LWATMGR) "know" which table to load? From the headers. You put a header "---E2---" at the top of a list of program names, and every program below, will be loaded into the E2 or Authcmd table. Similarly, an "---E8---" entry will cause all the names below to be loaded into the E8 or Authpgm table. And so forth. I think the LRECL=8 way is the best. If the LRECL=8 list is blocked 8000, you can fit 6000 table entries into one 3390 track! Very compact!

- What do our programs do? My TSUB program can substitute any program name for any other program name in any one of the "Local Copy" auth tables. In addition, TSUB can change the properties of the existing tables. (Dan's programs don't do that.) More about this later. LLWA can load new copies of the tables from any of the 3 sources. So can LWATMGR, but with LWATMGR, the load library "origin" library does not have to be APF-authorized, because he does a "directed LOAD". Dan's LWATEDIT program will EDIF (ISPF EDIT) any of the existing tables, and when you SAVE the EDIT, LWATEDIT will create a new table for you by CALL-ing LWATMGR under the covers. Therefore, if you use LWATEDIT, you have to put LWATMGR in BOTH the E2 and E8 tables.

Practical Implications of This

- Since we can now load our own authorization tables, and if we have been granted an APF-authorized load library (by management) for our own use, then we can deploy HUNDREDS of single-purpose user-written APF-authorized TSO commands or programs to do our everyday work with. We don't have to depend on the installation-supported authorization tables. We can make our own tables, for our own TSO session only, and we will not impact the rest of the TSO sessions in the installation, at all.

- One further note. What if someone does a PARMLIB UPDATE(xx) after we made our own tables. The PARMLIB UPDATE(xx) will undo our tables, unless we do one additional thing (LLWA and LWATMGR both do this automatically. And TSUB will do it manually if you ask it to).

- There is a flag bit in the LWA for each of the four tables, to tell you if that table was loaded from the PARMLIB-created table in common storage, or from a copy of the table in an authorized STEPLIB. If the STEPLIB bit was on for that table in the LWA, then a PARMLIB UPDATE(xx) command does NOT replace that table, but leaves it alone. Therefore, both the LLWA and the LWATMGR commands reset the STEPLIB bit ON for each table that they replace. BTW, the TSUB command also has the power to set the STEPLIB bit in the LWA for any table, either ON or OFF. The TSUB command has other LWA value-setting powers as well, such as adjusting the marked lengths of the tables, or marking the lengths as zero for each table individually.

- Length values are assigned to the PARMLIB-created tables in common storage. Length values are NOT assigned to the tables that were copied from an IKJTABLS load module in an authorized STEPLIB. How do the length values, if they exist, affect how the table is read by the TSO control programs?

- This is how it works. All of the tables are delimited by a blank entry. However, if a table has some blank entries at the end, and the next one gets filled in (say by the TSUB program), then the new program name at the end will be honored. Now when an IPL or a PARMLIB UPDATE(xx) or a SET IKJTSO=xx operator command is executed, the table created will have ONE blank entry at the end, but a LENGTH value will also be there, that includes the blank entry. So if you fill up the blank entry with another program name, since the length value will include that last slot, the last program name will be honored, even though the adjacent storage might not have any blanks to delimit the table. In other words, if a table length exists and no blanks delimit, then it is the TABLE LENGTH that is honored. TSUB can adjust all the table lengths in several ways, which we shall not discuss here.

Dan Dalby’s STEPLIB command

- Dan Dalby (CBT Tape File 452) has written a dynamic STEPLIB TSO command which dynamically adds a STEPLIB, exactly in the same way that IBM’s JCL STEPLIB would be put into effect. But Dan’s command does it dynamically, on the fly, and even if ISPF is up. This can be very useful. If Dan’s STEPLIB is invoked to add an APF-authorized load library to the STEPLIB concatenation, and all the STEPLIB libraries are APF-authorized, then the AC(1) programs invoked from that STEPLIB concatenation will run authorized. This is a way to deploy your own APF-authorized library of tools. STEPLIB hooks them up to your TSO session.

- IBM’s TSOLIB command, which is more shaky in the way it operates, and which has to be invoked from TSO READY mode, will also honor APF authorized libraries that are TSOLIB-ed to. So if STEPLIB is unavailable to get your authorized programs to execute from your TSO session, you might try and use TSOLIB to see if it will do what you need. But Dan Dalby’s STEPLIB command will do it better.

- Dan Dalby’s STEPLIB command has to be itself authorized, so the code has to be in an authorized library (perhaps yours, perhaps a system library) and the STEPLIB name has to be in the IKJEFTE2 (Authcmd) table. You can put TSUB alone in the system’s Authcmd table, use it to substitute STEPLIB for an existing entry, or to fill in the first blank entry at the end. Then you can use STEPLIB to add your APF-authorized tool library; afterwards you can run TSUB again to enable you to invoke LWATMGR or LLWA, and you’re on your way.

- If a program that should have been linkedited as AC(1) was not, you can reset the appropriate bit in the pds load module directory entry, using the PDS (8.6) command from CBT Tape File 182. Just point the PDS command to your load library, and issue “ATTRIB module AUTH” against that load module name. The module will be marked AC(1) henceforth, with not much effort on your part.

- All of this will help you, if the installation does not allow you to use a LOGON PROC of your own, with an authorized STEPLIB (your own authorized library). In that case, you just have to put the TSUB and STEPLIB code into a different authorized library, available to your TSO session, and put TSUB into your real PARMLIB Authcmd name list. That’s all. Then use TSUB to put STEPLIB into your E2 table, STEPLIB to add your authorized library and make its programs reachable from your TSO session, then authorize LLWA or LWATMGR (already in your load library) into the E2 table, load your own auth tables, and you’re in business.

Why accumulate APF-authorized programs?

- Every setting in a PARMLIB member will eventually be activated, to put corresponding settings in common system storage. These settings are accessed (by the system programs) using well-documented paths—paths that you can follow, too, using your own programs. If your programs are APF-authorized, you most likely will be able to adjust these PARMLIB-set values to your own liking, and hopefully, in a way that will help the installation achieve its objectives.

- I have heard of some corporate managements, which restrict even a sysprog's access to PARMLIB. This will hamper your ability to do your work sometimes. In such a case, or if you are a consultant who comes in and is asked to do a job, you sometimes must get around artificial limitations which corporate management (right or wrong) might put on you. But they asked you to do your job, and you have to do it. That comes first. You have to do your assigned job.

- So what do you do? You NEED an APF-authorized library that you will have access to. They have to grant that. Otherwise, your hands are completely tied. Then you do what I've said before, to put your tool kit into place. But what tools do you put in the kit? It's a long story, but I have some suggestions. Before I make the suggestions, we'll show an example or three. These examples will illustrate what can be done with tools like this, and how handy they are when we have them. After that, we'll suggest more general tools for the tool kit.

The XMIT command’s problem

- This problem no longer exists from z/OS 1.9 and further on. But it was exasperating while it existed, and something had to be done about getting around it. The problem, and its solutions, will illustrate what we can do.

- Background: IBM’s TSO XMIT command was designed to send data in various formats, across communication lines, from one MVS system to another. In order not to clog the communication lines of the network, a global limit was placed on how many 80-byte records will be sent across. After the limit is reached, the command output is abruptly cut off. For example, if the installation chose to set the limit at 50000 lines (a low value), then any XMIT command transmissions will be cut off after 50000 80-byte records have been sent. If you want to transmit a large file which will be reformatted into more than 50000 80-byte records, you simply can’t. The reformatting process will be cut off at 50000 lines. Tough.

- This limiting number is set in a PARMLIB member. It is also the IKJTSOxx PARMLIB member. But that number is part of the TRANSREC keyword, under the OUTLIM subkeyword.

- You will ask a question. Data is not always in the form of 80-byte records.

- Answer: True, but the XMIT command converts all the transmitted data first, into 80-byte records. Then it transmits it. Then it is converted back, on the target MVS system, using the TSO RECEIVE command, which undoes the initial formatting into 80-byte records, and restores the data into its original format on the target system.

- Byproduct of this: Using the optional OUTDSN(dataset.name) parameter of the XMIT command, the data does not actually have to be transmitted, but it is only reformatted into an 80-byte LRECL sequential file. The RECEIVE INDSN(dataset.name) TSO command will restore the transmitted (read reformatted) data into its original format.

- Problem. If you are just reformatting a large file, and not transmitting it, you are not clogging up transmission lines. But you are just storing the data in reformatted form on the same system. So why do you still have the same limitations and cutoff as though you were actually transmitting the data across communication lines?

- IBM’s solution. After z/OS 1.9, the OUTLIM number is ignored when only reformatting data without transmitting it, and in the case of reformatting only, no cutoffs are performed.

- Our solution before z/OS 1.9. I wrote an APF-authorized TSO command to dynamically change the OUTLIM number in common storage. Then I did my XMIT reformatting. Then I invoked my program to change the number back to what it was. I use the TSO XMIT command for this purpose so often, that it justifies my making the effort to write this command. But once I wrote the command (it is the CINMX command on CBT File 731), it can be used by anybody.

The LOOK command’s problem

- When writing Assembler programs that peek and poke about system storage and follow control blocks, I always use the free “LOOK” TSO command from CBT Tape File 264. The LOOK command will show you 256 bytes at a time, of any system storage in your own address space, or (if authorized) it will issue an SRB (it is an old program) to access storage in any other address space. LOOK also makes it easy to follow control block chains. I’m not going to tell you more about LOOK here, but suffice it to say that it is extremely useful if you write system programs, so you can see the data that you’re programming about.

- In z/OS 1.9, LOOK just stopped working. It started to produce very weird system abends such as a SB0A-5C and others. What went wrong?

- I called my friend Sam Knutson and complained. Sam told me that IBM fixed an issue that he was campaigning about for years. It is a security vulnerability when someone can allocate Key 8 storage in CSA, so IBM did something about it. They made a PARMLIB setting in the DIAGxx member to forbid user key allocation of CSA. The setting of VSM ALLOWUSERKEYCSA(NO) was present in z/OS 1.8, but it became the default in z/OS 1.9. The LOOK command allocates CSA storage in Key 8. Hence these weird abends for violating the new DIAGxx setting.

- So what to do about it? Of course I could have rewritten LOOK so it doesn’t allocate CSA in a user key. But I attacked the problem the other way. A very nice IBM’er, who knows who he is, told me where the bit was, that the DIAGxx setting of VSM ALLOWUSERKEYCSA(NO) sets. Then I wrote an APF-authorized TSO command, called UKEYCSA, which temporarily turns the setting off. I use LOOK for the 5 minutes I need it for, and then turn the setting back on. The vulnerability is only for 5 minutes, and not for the whole time, so the risk is minimal. Meanwhile, I could use the LOOK command to find out about the pieces of storage I needed information about.

PARMLIB settings all cause in-core settings in CSA

- Isn’t it obvious? A setting in PARMLIB has to be system-wide. Or at least LPAR-wide. This means that something machine-readable has to be affected, in commonly accessed machine storage. True? Of course it is!

- So the question is: For every PARMLIB setting, where is the in-core consequence of it? And what is the path of control blocks that you have to take, in order to get to the value which this PARMLIB setting has set?

- It follows, that if you don’t want to change PARMLIB, but if you want to achieve the EFFECT of changing something in PARMLIB, you merely have to find the in-core consequence of the setting, and write an APF-authorized program, or better, an APF-authorized TSO command, to achieve that same change which the PARMLIB setting would have achieved.

- Therefore, a further consequence of this, would be that if certain PARMLIB settings are important enough to change on a regular basis, it would pay to have a collection of APF-authorized programs to do that. So it happens that one can properly and justifiably accumulate a large collection of single-purpose APF-authorized TSO commands, and if he/she knows how to use them, they would come in very handy, very often. We’ve made the effort in that direction.

- Therefore, it is not at all far-fetched, to have to authorize 200 or 300 new TSO commands for yourself, to be able to conveniently do your work. And TSUB, LLWA, LWATMGR, and LWATEDIT are now there, to help you do that for yourself alone, without affecting any of the other TSO users in the shop.

Another Example – adjusting the number of Notices Records produced by ACCOUNT/SYNC

- Because I have written both a free package and a commercial package to manipulate all records in SYS1.BRODCAST, I am personally very close to this example, and I want to submit a request to IBM to change the situation. But meanwhile, I did not wait for IBM. I wrote my own APF-authorized TSO command to solve the problem myself. When IBM solves the problem THEIR way, I'll be happy too, but I did not wait for them to do it.

- In order to format a new SYS1.BRODCAST dataset, you use the ACCOUNT TSO command, followed by the SYNC command. This (by default) produces 100 global notices records. These are lines of messages that the administrator can send to ALL TSO users. These are the messages like: "SYSTEM B will be down on Tuesday from 9 PM to 12 Midnight for maintenance. Please use SYSTEM A at that time." Suppose, when you are doing an ACCOUNT and SYNC to format a new Broadcast Dataset, and you don't want just 100 Global Notices records. You want 150 or 200. How do you change the default number of Global Notices records produced?

- IBM says you are supposed to zap a load module, and re-IPL. That's the only way. This is a very weird thing for IBM to recommend. I thought they were trying to save you from IPL-ing.

- Answer is, that the number of Global Notices records that ACCOUNT and SYNC "listens to" is a fullword quantity in the CVT. So in order to reload that, from a user's point of view, you have to zap a load module and re-IPL, because the CVT gets initialized (mostly) at IPL time. I wrote an APF-authorized TSO command to just plug in another number here, and ACCOUNT and SYNC are happy. I named my program BDMNNOTC (sorry), and it can be found together with my free Broadcast Dataset package on CBT File 247. If I say BDMNNOTC 200, the ACCOUNT/SYNC will produce 200 Notices records. Simple as that.

- How would IBM solve this problem? Simplest way is to create a new keyword in the IKJTSOxx PARMLIB member, such as NUMNOTICES(200). When you do a PARMLIB UPDATE(xx) TSO command, or a SET IKJTSO=xx operator command, the new number would be plugged into the CVT. Simple as that. We have to submit a SHARE requirement. Maybe IBM will get this done. I think that there just wasn't any demand, so IBM just overlooked the obvious thing to do.

- At this point, I HOPE THAT YOU SEE THE GENERAL USEFULNESS OF HAVING SINGLE PURPOSE APF-AUTHORIZED TSO COMMANDS. You may have 200 or 300 of these, but we now have a way to quickly authorize them all to TSO.

Writing your own code - 1

- Back in the “golden age” of Assembler programming, which was in the 1960s and 1970s, there were people who didn’t care to look at publicly distributed free programs. I talked to them. They would tell me: “If I need to have a tool, I’ll write it for myself!” For some people, this might have been good enough, but for most of us, especially nowadays, this approach can have major flaws (even when the person DOES have the skills).

- First off, such a person would only write a program if he or she had a problem, and NEEDED a program to fix it, or to address the situation. No situation, no program written! Their “gem” is not even in the world! Never created.

- Second, what if they DID write a program? I’ve seen many of them. Usually such programs were written for the programmer himself (or herself). They weren’t intended for more widespread use. So (for example) the program definitely might flip the bit that was intended to be flipped. But how would it behave if you entered the wrong parameter? The programmer would tell you that he/she always entered the right parameter. (I’ve heard this too.) We know better. If we want to use this tool, or this idea, we’d better make it safer FOR OTHER PEOPLE to use.

- Third, what if this (skilled Assembler programmer) person suddenly found him/herself in a pinch and a time constraint? There’s no TIME to prepare the defense! If it isn’t ALREADY in your pocket, it won’t be there if you need it all of a sudden.

- Fourth, such a person has not learned to use OTHER PEOPLE’S SKILLS. Fact is, (argue as you might like to) “the whole is greater than the sum of its parts. A person who is an individual, but yet recognizes the contribution of other individuals, will profit from both his/her own efforts and from theirs too. If someone honors the contribution of others, the “honor” has a way of chasing him/her and catching up.

- Last summer I embarked on a project with another programmer (Dan Dalby) whose skills are very different from mine. And I can testify that the parallel products from both our individual efforts, are better than if we had worked separately.

Writing your own code - 2

- If you write your own programs, they will definitely reflect your personality, and the skills you have acquired thus far. Don’t EVER say to yourself that you aren’t as good a programmer as the next guy. Go for THE RESULT, ask for help if you need it, and try to GET THERE. As you program, you’ll acquire more skills to get past the problems that come up. This should be WELCOMED, because it is part of the 50-50, 100-100 rule. You are acquiring more skills and experience (which you get to keep).

- I make it a practice to use the instructions that I have already learned how to use. And I pick up how to use new instructions, one at a time. If you are programming for the newer machines, and the program(s) might get big, it might pay to use the new “jump” instructions instead of “branch” instructions because you can get more addressability.

- The goal is to GET THE JOB DONE. Then fix the code to make it more foolproof. You don’t want someone ruining a system and causing downtime, just because you were careless. But at the beginning, make sure that you’ve achieved the proper result first (i.e. flipped the proper bit(s), etc.) and the system “respects” the changes that you have made. THEN, clean up the code to make sure no unwanted “bad” things can happen with it. “Proof of Concept” is a very important thing to establish first, before you put a great effort into programming something (only to see later that the system doesn’t “care” anyway, after you’ve made the changes).

- Use diagnostic tools that you are comfortable with. If you have something sophisticated and expensive, such as a commercial “debugger” program, then fine. If you like to program by putting “display” code into your program for diagnostic purposes, then fine. I like to look at core with the LOOK TSO command from CBT File 264, which allows you to follow control block chains and see what’s actually there in the live system. That is how I have been able to document some “undocumented” IBM control blocks. You can see what is in them (at least if the fields have been filled). And you can see what data any addresses point to.

Writing your own code - 3

- There is one rule I have: “ANY tool is useful”.
- I have a CLIST which contains one line of code. (Eli Duttman is the author.) I use it all the time. The CLIST is named “X”, and the line of code reads: LOGOFF. Have you ever had to leave your desk and you are in a hurry? You make three or four “finger checks” typing the word LOGOFF and the system doesn’t “understand you”. Finally you get disgusted and walk away. If you typed X, there’s no problem at all!
- So it doesn’t matter if a tool is “trivial”, “elementary”, “mindless”, “brainless”, “elegant”, “not elegant” or whatever. The question to ask is: “Is it USEFUL?”
- Next, you should weigh the idea of whether you should take the time to write something. Of course, your “work tasks” are what you were hired for, and they take precedence. But what if you have some “slack time”? Then, if you think the installation would benefit (THEY are paying you, not the other people, so you have to THINK OF THEM FIRST), then you can make a decision. I’m not going to tell you whether to ask your boss first. I’m also not going to tell you how to drive your car.
- Try and use one of the more modern languages, with Assembler preferred. Not everybody has a PL/I compiler any more. And COBOL changes every five years or so (I am deliberately being cynical – no offense to the COBOL people). But don’t shy away from these languages if they are the only ones you know how to use. Make tools! That comes first.

More About the CBT Tape collection

- I'm a little biased, being the proprietor of this magnificent collection of free tools, which owes its existence to thousands of contributors, and to the supreme dedication of Arnie Casinghino. Arnie's enormous selflessness and dedication to detail, over a span of so many years, has brought forth something unique in "MVS-dom".

- The CBT Tape collection has been existence for just about 35 years! It has been successful because people like you have sent in contributions to it.

- Updated "tapes" are produced whenever we feel we need to. In former times, when the collection was strictly on tapes, updates needed to be distributed to the people in a timely matter. Arnie sometimes made 20 or 30 updates per year. Nowadays, when the stuff is on our website, www.cbttape.org, the updates, even between tape versions, are readily available to everyone on the "Updates" page of the website. My recommendation is that when you need a file, see if a new version of it is available on the Updates page. Then go and check the "CBT" page. So nowadays, we make only 2 or 3 updates to the tape versions, per year.

- I have just a few rules for contributions. They are as follows:

- 1- All disclaimers in the CBT tape documentation always apply, and they override all other statements both
- in the CBT Tape documentation and elsewhere, regarding the fitness and merchantability, etc. of
- the materials in each file in the collection.
-
- 2- There should be no "time-outs". Somebody using the tape ten or more years from now should be able to
- run the materials then too. (Of course if the MVS Operating System changes, we have no control over
- that, and all disclaimers apply. But there must be no artificial time-outs or expirations built into
- the software.)
-
- 3- There should be no restrictions about who can copy the materials. No contributor will be able to stop
- anybody from copying the entire CBT tape for themselves.
-
- 4- The contributor may (if he/she wants) retain ownership of the materials, using copyright notices
- to indicate that fact. However, the owner may not restrict others from copying or using the file. If
- the owner wishes to prohibit commercial use, he/she may do so.

Good Luck! A few (parting) thoughts

- We hope that your working life will be improved by this session. Maybe even your "outside life" might benefit too.

- Always use good sense. Going through life is like driving a car. You never know what will come up, and you always have to keep making judgments. If you are the driver, nobody, and that means NOBODY, can make the judgments for you. You have to steer your own life. You can ask advice (and you should) when there's slack time, but in a crunch, you always have to make the decisions yourself. Accepting responsibility is part of life.

- The aim of this session is to make YOUR job easier. The doctor doesn't let "non-doctors" use his prescription pad and his diagnosis tools (or his operation tools, scalpel, etc.). These must be used only by qualified personnel. Our "personalization of TSO authorization" which we have shown you about, will benefit you as system doctor, and will tend to be inaccessible to unqualified users, if you make it that way. Do your best! You might try and invent some RACF safeguards, or ACF2, or whatever you need. If they work, please send them in to me. (Thanks in advance.)

- Again, best of luck. Use these things wisely.