

TSO Times

Fall 2004

In This Issue

What's New in TSO/ISPF?

Five 3270 Emulator Tips

Guerilla ISPF Training by Steve Comstock

The SHOWDSNS Dialog by Jim Moore

Are You Getting Quality Training? by Olivia Carmandi

Working With Packed Decimal Data in REXX by Steve Comstock

The Power of DDLIST by Jim Moore

TSO Assistance for the MVS Professional

What's New in TSO/ISPF?

Do the programmers and users of TSO/ISPF that work in your data center get the highest and best use out of the TSO/ISPF software they use every day? Are they aware of the huge number of improvements, enhancements and new features that have been added to the TSO/ISPF environment over the last several years?

New utilities, commands, dialog services, improved methods of working and much more have poured out of the ISPF developers at IBM recently. All of these great additions, coupled with the old stand-bys, make some form of training almost a requirement.

In this issue of The TSO Times, there is a bit of a training focus. Steve Comstock of The Trainer's Friend writes about what he calls Guerilla ISPF Training and Olivia Carmandi of MVS Training contributes on the topic as well.

Jim Moore tosses out what he calls a "training dialog" - an Assembler/REXX combination named SHOWDSNS that allows for an interactive browse of the TIOT, similar to DDLIST.

As you read these articles, ask yourself: Can I incorporate any of these new and improved techniques in my day-to-day work? Do I need to know more about how these enhancements will make my job easier?

Use these three simple steps:

- Discover
- Experiment
- Incorporate

If you need more help, contact the training professionals who can answer all of your questions and cut through all of the befuddlement.

Five 3270 Emulator Tips

If you still work on an actual 3270 terminal, these tips won't have much value for you. But if you use any one of the many 3270-emulator packages (Reflection, Extra!, Rumba, Vista, etc.) to work on a mainframe, they might come in handy.

1. Learn to use BOTH cursors available to you - the mainframe cursor as well as the "toothpick, I-beam" mouse cursor. Rolling and then left-clicking the mouse is a much faster method to get to an insert point on the screen than tabbing or using the arrow keys.
2. If your emulator supports the "left double-click to highlight text" Windows standard (Reflection does), always use this method to highlight short strings (like dataset names) as opposed to "swiping". Follow on with the keyboard shortcut CTRL-C to Copy the highlighted text to the clipboard.
3. Test to see if your emulator supports the Cut Text (CTRL-X) function. The better ones do and are sensitive enough to know when the highlighted text can truly be "cut". That is, in an environment like ISPF edit, a good emulator will in fact remove highlighted text in response to a CTRL-X. In environments like ISPF Browse, they will not. They WILL place the highlighted text on the clipboard as if you had requested a Copy (CTRL-C) in this situation.
4. In ISPF Edit, if your emulated terminal has Insert On and you want to Paste a piece of text (CTRL-V) from the clipboard, will your emulator insert the pasted text? Good emulators will - right-shifting the existing text to accommodate the pasted text (NULLS ON must be set in ISPF Edit profile) coming from the clipboard.
5. In ISPF edit, issue the HILITE command and activate the Colors drop-down.

...continued on page 6

www.tsotimes.com

Guerrilla ISPF Training

by Steve Comstock

For most z/OS applications programmers, ISPF is their window into the mainframe. It is the tool for writing and editing code, JCL and control statements. It is the mechanism for submitting jobs and examining the output. It is the center for utility work such as copying files for test, backing up data (and restoring it when necessary), for renaming files, even for getting into z/OS UNIX.

It would make sense to ensure ISPF users are up-to-date on the latest features of ISPF - that they are as efficient and effective as possible when using this powerful tool. Over the last ten years or so, ISPF has undergone tremendous changes - both in interface design and changes in commands with enhancements to capabilities that vastly improve what was available before.

Yet management refuses to recognize that some organized training to update their peoples' skills in ISPF would bring enormous returns on a small investment. "They already know that," is a common management mantra.

Not in my experience.

We offer several courses for ISPF skills, yet we rarely are able to sell them to customers. Which is a shame because these classes improve developers' skills and effectiveness in a short period of time.

Because we believe so strongly in the need for the content in these courses to be disseminated, we have started including fragments from these courses when we teach other classes. In this article, I will outline the techniques we most often add to other classes based on what we observe about the students' skills and knowledge of ISPF. Since this training is not scheduled, requested, nor paid for, I call it *Guerrilla Training*.

Some of these techniques have been included in various articles, but I believe this is the first time they have been pulled together with the intent of placing them into class material (or used for "brown bag" lunch technical sessions or other informal settings).

Technique 1- Settings

The changes from ISPF V3R5 to V4R1 (around 1996-1997) introduced a major change to the user interface. A lot of the IBM-supplied defaults were not the most user-friendly and I see a lot of people working with these in effect simply because they are not aware of alternatives. So the first thing to do is to select ISPF *Option 0* (zero) or from a command line issue the *Settings* command (it has no operands). In either case, you will see a panel like the one shown in Figure 1. On this panel, I direct the students to at least remove the "/" from these options:

- *Command line at bottom* - This is a personal preference item but most people seem to prefer the line near the top.

- *Tab to action bar choices* - With this de-selected, the terminal *Home* key sends the cursor to the command line (if the command line is at the top of the panel).

- *Always show split line* - The split line is a line of dots when you have the screen split. Removing this line gives you one more line of screen real estate to work with.

Technique 2 -Multiple split screens

Most students know about split screens and command stacking but ISPF has extended screen splits from a maximum of two screens to a maximum of 32 (although the default is normally eight). Each screen is called a logical screen. Normally, you can only see one at a time, although the *Split* command can allow you to see part of each of two screens at one time. The following commands are of enormous benefit in becoming more productive.

- *Start [option]* - Start a new screen - better than the *Split* command since you can specify a starting option and the new screen is a whole screen, not dependent upon where the cursor is positioned. Note that the *Split* command can also split more than two screens if you type *Split New* as an ISPF command. All logical screens are numbered, beginning with "1". You can assign a name to a screen using the *Scrname* command. Some variations:

...continued on next page

MVS/Quick-Ref™

The #1 Programmer's Choice

MVS/Quick-Ref is an on-line quick reference tool for users of the MVS operating system. It has the ability to "pop-up" over an active ISPF application and give you a quick answer to an MVS-related question and then go away without affecting the active application.

**Instant access to over 27 million lines of MVS/OS/
390 reference information.**

**Why do you need MVS/Quick-Ref?
It saves you time and money!**

For more information or to order a FREE trial, give us a call at 603-643-4002 or visit us on-line at www.chicago-soft.com



Guerrilla ISPF Training

...continued from previous page

1. *Scrname on* - Causes the name you assign to a screen to display in the upper left-hand corner of the screen. This is optional.
2. *Scrname name [perm]* - *Name* is your choice. It must be 2-8 alphanumeric characters the first of which is not numeric and not one of: NEXT, PREV, LIST, OFF, ON. The *perm* parameter will keep the name tied to the logical screen even if you change the screen content.

Finally, you get from logical screen to logical screen by using the *Swap* command. There are several variations:

- *Swap next* - Swap to the next logical screen.
- *Swap prev* - Swap to the previous logical screen.
- *Swap n* - Swap to screen number *n*.
- *Swap name* - Swap to the screen with name *name*.
- *Swap list* - Show a pop-up list of all screens currently established. From this pop-up you can choose a screen or even start a new screen.

Typically, I recommend that students change their function key assignments so that PF9 is assigned to *Swap Next* (although some people prefer *Swap List*). Note that you may have to set this in multiple screens, since each screen may have a separate *Keylist* associated with it. Once you set the PF9 values on all of your screens, the setting will be remembered across logons. As always in ISPF, you must make sure you exit ISPF cleanly though, instead of timing out, for the settings to be remembered.

Multiple Screen Scenario

Suppose you start out with your first screen and you have some maintenance to do on a program. You might issue the following sequence of commands:

- *Scrname On* - This will cause the screen name to display in the upper left-hand corner.
- *Scrname CODE Perm* - This assigns a name of CODE to this first screen.
- *Option 3.4* - Specify high level qualifier(s) you want and press <Enter>
- Enter an *E* next to source library
- Enter an *S* next to member to edit. You are now in edit on the code. Make changes to the code and enter *Save*.
- *Start 3.4* - This will start a second screen, running DSLIST (Option 3.4).
- *Scrname JCL Perm* - This assigns a name of JCL to this second screen.
- Specify high level qualifier(s) you want and press <Enter>
- Enter an *E* next to appropriate JCL library
- Enter an *S* next to member to edit. You are now in edit on your JCL (perhaps a job to compile the source code just edited).

- *Start SDSF* - This will start a third screen, running SDSF.
- *Scrname SDSF Perm* - This assigns a name of SDSF to this third screen.
- Enter the *ST* (Status) command to watch jobs. Note that this and the previous two steps depend upon the product you use to view job output and what command or option you use to start that product. This example uses the IBM-supplied product named SDSF but there are others.
- *Swap JCL;Sub;Swap SDSF* - Go back to the JCL screen, submit the job and return to the SDSF screen

When your job has finished you can look at the output. If you see you need to make a change, swap back to the CODE screen and make your changes, then issue:

- *Save;Swap JCL;Sub;Swap SDSF;End* - This saves your code changes, swaps to the screen with your JCL, submits the job, swaps to SDSF and then closes the job output you were looking at previously.

Walking through this scenario, especially drawing pictures of the screens on a white board or flip chart, really brings it home to students how more than two logical screens can improve their effectiveness (although I do find some people resist this - they still prefer to stay working with one, or at most, two screens).

Technique 3 - Command Retrieval

The final topic I usually squeeze in when doing Guerrilla Training is how to keep from re-typing long command strings. Most students are aware of two ways:

- 1) Assign a string to a function key
- 2) Assign *Retrieve* to a function key (usually PF12)

But there is another, relatively new technique that can also be very helpful- The *RETP* command. Issue this command (or press a function key assigned to this command) and you will see a pop-up list of the last several commands (the number depends on

...continued on next page

TSO Times is published by Chicago-Soft, LTD., which develops and markets corporate productivity and management software for main-frame, Windows and web-based systems. Intended as a forum for the exchange of technical information of interest to TSO and ISPF users, its content will be mostly technical, with some promotional materials.

All inquiries concerning subscriptions, requests and change of address should be sent to Chicago-Soft, LTD, ATTN: TSO Times, One Maple Street, Hanover, NH 03755, TEL: 603-643-4002, FAX: 603-643-4571, email: information@tsotimes.com

Additional information can be found at www.tsotimes.com.
Chicago-Soft, LTD information can be found at www.chicago-soft.com

Copyright © 2004 Chicago-Soft, LTD All rights reserved.
All trademarks are property of their respective holders.

Guerrilla ISPF Training

...continued from previous page

how many commands have been issued and the fact that the command buffer is limited in size). Figure 2 shows an example of what you might see.

This list is scrollable, and you can select a command, which is then copied to the command line, and you can modify the command if you need to before pressing <Enter>. This can save a lot of typing.

Conclusion

This article is no substitute for a full-fledged ISPF class. But the information presented would seem to be useful as a bare minimum for bringing ISPF users up to speed in how to become more effective and productive using modern releases of ISPF.

```
Figure 1 - Settings Panel (ISPF Option 0)
-----
Log/List  Function keys  Colors  Environ  Workstation  Identifier  Help
-----
Command ==>
ISPF Settings
Options
Enter "/" to select option
/ Command line at bottom
/ Panel display CUA mode
/ Long message in pop-up
/ Tab to action bar choices
/ Tab to point-and-shoot fields
/ Restore TEST/TRACE options
/ Session Manager mode
/ Jump from leader dots
/ Edit PRINTDS Command
/ Always show split line
/ Enable EURO sign
Print Graphics
Family printer type 2
Device name . . . .
Aspect ratio . . . 0
General
Input field pad . . B
Command delimiter ; ;
Terminal Characteristics
Screen format 3 1. Data 2. Std 3. Max 4. Part
Terminal Type 3 1. 3277 2. 3277A 3. 3278 4. 3278A
5. 3290A 6. 3278T 7. 3278CF 8. 3277KN
9. 3278KN 10. 3278AR 11. 3278CY 12. 3278HN
13. 3278HO 14. 3278IS 15. 3278L2 16. BE163
17. BE190 18. 3278TH 19. 3278CU 20. DEU78
21. DEU78A 22. DEU90A 23. SW116 24. SW131
25. SW500
```

```
Figure 2 - Pop-up displayed by the RETP command
+----- Retrieve -----+
| Options Help          |
+-----+
| CODE SPF Retrieve Panel |
| Select the command     |
| to be retrieved       |
| More: +               |
| 1. SWAP CODE;SAVE;SWAP> |
| 2. ST                  |
| 3. sub;swap sdsf      |
| 4. scrname jcl perm   |
| 5. start m.5          |
| 6. SCRNAME SDSF PERM  |
| 7. start m.5          |
| 8. SORT CHANGED      |
| 9. M                  |
| 10. num on std;unnum  |
| 11. unnum             |
| 12. SORT CHANGED     |
| 13. 3.4               |
+-----+

```

Steve Comstock has many years of mainframe experience. He is President of The Trainer's Friend, Inc. (www.trainersfriend.com) which offers many training courses in TSO/ISPF - both usage and dialog manager development.

The SHOWDSNS Dialog

by Jim Moore

Now that DDLIST is an entrenched tool within ISPF, I feel a bit sheepish about writing about this older "home-grown" ISPF dialog. I wrote it many years ago when I became frustrated with the lack of interaction to the all-important TIOT (Task Input Output Table) available under TSO/ISPF. LISTALC STATUS was just too dated and primitive.

Doug Nadel's DDLIST (formerly known as ISRDDN) solved this whole "lack of interaction" and then some.

A critical thing that a TSO/ISPF dialog/macro developer has to be aware of is: Just where are various components of dialogs, commands and other executables being stored, retrieved and executed from? This can get quite confusing and one thing that helps a lot is to have a clear (and interactive) view of the entire set of currently allocated datasets - concatenated or in single allocations.

My overall design goal when writing the SHOWDSNS dialog back in the early 1990s was simple: Write an Assembler ISPF dialog function that transferred the contents of the TIOT into a standard ISPF table. Once that was done, a REXX (or CLIST or COBOL or C or whatever) function could handle all of the interactive portions of displaying the table, allowing line commands and all that other good stuff.

Further, isolating the table creation logic into an Assembler program that had no real human interface allows for a good deal of reuse. If you fiddle around with the dialog, I bet you can think of a few other uses for the table of TSO-allocated datasets that the Assembler program creates.

I decided to use SVC 99 with Verb Code 7, all in conjunction with the DINRELNO text unit to access each entry in the TIOT. Why did I choose this method?

Long Term Support

If I directly accessed the TIOT, walking memory and address pointers, it might be possible that IBM would eventually change something in the TIOT layout or the addresses that point to the TIOT. In fact, they did exactly that recently with the SWA=ABOVE parameter.

By using SVC 99, I was able to use the same method that IBM themselves uses and I never have to worry much about memory layout changes.

Curiosity

I had noticed the DINRELNO text unit over the years and had always wondered: Of what use would a relative request for information be? To clarify, DINRELNO allows you to request information about the "nth allocation" (the 10th, 21st, 45th, etc). How would I, as a programmer, know in advance which relative "slot" a particular allocation took up in the TIOT?

...continued on next page

The SHOWDSNS Dialog

...continued from previous page

I wouldn't unless I had already accessed it and examined it. However, it dawned on me that the real purpose of DINRELNO was to start a counter at 1, retrieve the information about the 1st allocation and then add 1 to the counter and continue retrieving this way until "End-of-TIOT". And that's exactly the technique used - a serial access through the TIOT extracting the required information about each allocation in turn.

REXX

I chose REXX as the language for the interface portion of the SHOWDSNS dialog. Why? Ease of coding, pure and simple. I **DO NOT** like coding complex interface code in Assembler. Plus, one of the greatest things about the TSO/ISPF environment is the fact that you can mix-and-match just about any languages that you want.

Conclusion

The entire SHOWDSNS dialog consists of :

- One Assembler program
- One REXX EXEC
- Three panels

No messages, skeletons or saved tables.

SHOWDSNS is a good example of learning dialog because it demonstrates some useful techniques. These are:

- Creating a table in one function (program) but displaying (or accessing) it in another.
- Passing variables between function pools by way of the shared pool.
- Using multiple languages - Assembler and REXX
- Doing "file-less" programming. That is, no files are opened, read, written or closed in SHOWDSNS.

The entire dialog is posted at The TSO Times web site in IEBUPDTE format. There is also a small slide show there as well that shows how the screens look. Download it and try it out.

Jim Moore is the Editor of The TSO Times and has been known to write a few ISPF dialogs and edit macros.

FREE **SUBSCRIPTION**

Are you reading this thanks to a friend or colleague? You can sign up for your own subscription by visiting www.tsotimes.com/subscribe.html and fill out our on-line subscription form.

Are You Getting Quality Training?

Olivia Carmandi, President and Founder of MVS-Training Inc. wants you to consider these points with regard to training:

- **Evaluate the course materials** – They should be written and have detailed text related to the course – not just pictures and graphics.
- **Consider the style of training** – It should be adaptive, participation-oriented and have as its main objective, improved job skills. Not just raw knowledge.
- **Always verify the training specialist's service** – Call previous companies that the trainer has taught at and ask for an objective analysis of their skills.
- **Realize that this analysis is necessary** – If you don't verify the trainer, who will?

Here's Olivia's *Quality Training Checklist*:

What do training sessions provide?

- ___ Verify that the training manual contains substantial, useful information.
- ___ How often do training material updates occur?
- ___ At least 60% doing practical tasks—not all theory?
- ___ Are there written learning checks every dozen pages to reinforce the learning process?
- ___ (Team Challenges) where participants perform tasks they'd do on their job.
- ___ Can the training be tailored to your site's needs?

Call a telephone reference and ask:

- ___ Does training vendor work with client to help analyze their training needs?
- ___ Does trainer adapt when necessary?
- ___ How did the trainer reply to participant's questions?
- ___ Did trainer exhibit practical experience?
- ___ What happens when the training is over?

Can you quantitatively measure the success of the training process?

- ___ Does training include before and after questions?
- ___ Ask to see past participant evaluations.
- ___ Give daily evaluations, review them with management, and adapt accordingly.

Working With Packed Decimal Data in REXX

by Steve Comstock

Although all the REXX documents tell you that REXX deals with all data as “typeless character strings”, REXX is really quite aware of numeric data, and can even be used to process records containing numeric data, including packed decimal numeric fields.

In this article an example is shown of working with a file that contains [at least] one packed decimal field. The presumption is that you know how to read and write records, using record I/O (EXECIO) or stream I/O (linein, charin, lineout, charout, and so on), so the mechanics of locating or updating records will not be covered. Instead, the processing of numeric fields will be explained.

Another assumption made is that the reader is aware that packed decimal data is numeric data stored as two decimal digits (0-9) per byte, except the right most byte, which contains a digit (0-9) and a sign (X'A'-X'F'). Further, recall that hex values A, C, E, and F represent positive numbers while hex B and D represent negative numbers.

Suppose there is an inventory file with 100 byte records and that locations 35-39 contain the unit price for an item, in dollars and cents. For example, if the field contains x'000004725C' this represents a price of \$47.25.

Now suppose we are running an ISPF dialog driven by a REXX exec that takes online orders and a customer puts in a request to buy some quantity of a particular item, say 12 Blue Wombats. We need to calculate the total price, so we need to multiply the quantity being ordered by the unit price (then, of course, figure applicable taxes and handling and shipping charges). Assume the quantity ordered is in the REXX variable QTY, and the item description is in a field called DESC.

Further, assume we are able to locate the relevant inventory record and read it into a variable called “inrecord”. The next step we must take is convert the data in positions 35-39 to REXX type numeric data.

Use the Built-In Functions

The tool to use here is the REXX built-in function C2X (“Character-to-Hex”). This function takes any string as input and returns a character string that represents the hexadecimal digits. For example, C2X('A') returns the characters C1 since a character A is X'C1'. If you take a hex string as input, you get the hex characters as a character string for output. In our case, we can code:

```
un_price = C2X(substr(inrecord,35,5))
```

This ends up with un_price containing the character string '000004725C'. Since we are only dealing with positive numbers here, we can actually delete the last character, by:

```
un_price = substr(un_price,1,9)
```

Now un_price has a numeric value, so we can do numeric calcula-

tions. To account for the two decimal places, we divide by 100:

```
un_price = un_price/100
```

Then we can calculate the amount as:

```
amt = qty * un_price
```

Finally, we can display this amount in a formatted way using the FORMAT built in function:

```
say qty desc '@ $' || format(un_price,,2) ' = $' || format(amt,,2)
```

and the user will see:

```
12 Blue Wombats @ $47.25 = $567.00
```

If you like to reduce the number of lines of code, you can combine many of these steps into just a few, so the above could be:

```
un_price = substr(c2x(substr(inrecord,4,5)),1,9)/100
amt = qty * un_price
say qty desc '@ $' || format(un_price,,2) ' = $' || format(amt,,2)
```

Conclusion

Look for an article online at www.tsotimes.com that explains how to work with binary data in REXX. The concepts are similar.

Steve Comstock has many years of mainframe experience. He is president of The Trainer's Friend, Inc. - www.trainersfriend.com - which offers many training courses in TSO/ISPF - both usage and dialog manager development.

Five 3270 Emulator Tips

...continued from page 1

Select 1, Overtyping Color. Set it to pink. End the HILITE dialog. Now, copy and paste some text into your edit session's data. Is the pasted text pink? That's right, a Windows clipboard paste is considered an “Overtyping”.

These are just five simple tips for exploiting the features of your emulator software in conjunction with the older 3270-screen model. There are many more things that can make you more productive if you stop to consider the possibilities. In general, learn to mix mouse movements and keystrokes, developing fluid hand movement between the two devices. Use one hand for the mouse and the opposite hand for the keyboard. Doing so might even help prevent repetitive stress injury, also known as “carpal tunnel syndrome”.

The Power of DDLIST

by Jim Moore

Who needs an interactive, scrollable list of their TSO session allocations? Need it or not, DDLIST is much more than just a list of allocated datasets.

Here's a list of the powerful primary commands available in DDLIST at the z/OS V1R4 level:

Apf	Browse	Con	CList	COUnt	CUstom
DUPLicates	Enq	EXclude	Find	Locate	LOAD
LONG	LPa	Member	MList	Only	Parmlib
Reset	Select	Short			

What a list it is!

The *APF*, *PARMLIB* and *LPA* commands (and *LINKLIST*, which curiously, is not documented but works) all create *pseudo-DD* names that show the APF (Authorized Program Facility), PARMLIB, LPA (Link Pack Area) and LINKLIST concatenations.

Once issued, these *pseudo-DD* names can be drilled through with the *Member* command to find member names, *Located* like any other DD name and individual libraries in the concatenation can be browsed, viewed or edited with line commands. Handy indeed.

Need a quick snapshot of any enqueue contention on a system? Issue the *CON* (Contention) command.

One of my favorite DDLIST commands is *ENQ*. It allows any TSO/ISPF user to really delve into the various and sundry enqueues that are issued in a complex z/OS environment – by major/minor name, by job name, by just about any criterion that you can think of.

I wrote a column about the *ENQ* DDLIST command back in March 2003 for NaSPA (Network and Systems Professionals Association). Have a look at this column at <http://www.naspa.com/PDF/2003/0303/T0303006.pdf> to see how *ENQ* can be used when testing complex applications; particularly ISPF dialogs that issue operating system enqueues.

About now, you might be asking yourself: What do all of these things have to do with my TSO session's dataset allocations? The answer is: Not much, but who cares? Run with it!

Need to know if an executable program is available on the z/OS search path? Try issuing the *LOAD* or *SELECT* commands. Try: *LOAD IEFBR14* just to get a feel for how these commands work. Read the tutorial (by pressing your HELP PF Key) to find out the difference between *LOAD* and *SELECT*. Can you say: CSVQUERY?

I'm still not 100% sure what the values shown by the *CUSTOM* command are. Here's what the tutorial says about *CUSTOM*:

- Show the values in ISPTCM and some ISPF configurations.

I'll tell you what, when I figure it out, I'll be sure to pass it

along. I'd love to see *CUSTOM* include the *ISPF Configuration Table* settings that apply globally across an entire site.

CLIST, as a command (as opposed to the language) will create a simple CLIST consisting of TSO ALLOC statements that will allow you to easily tailor your logon concatenations. Try it.

Try the other commands after a brief read about them in the tutorial.

I've saved the best for last and the best, in my opinion is the *BROWSE* command.

This isn't a dataset browse. It is a memory browse. There is so much functionality to the *BROWSE* command, that I cannot cover it entirely in the small amount of space allotted here.

When it comes to testing, learning and examining memory on a z/OS machine, the *BROWSE* command of DDLIST is a clear window into the innards. Start with a simple *BROWSE* of the Prefixed Save Area (PSA, DSECT IHAPSA) and follow along.

- Invoke DDLIST
- Type Browse 0.

Note that the dot after the zero is required ("Browse 0."). Now, find various addresses that are anchored in the PSA by using the IHAPSA DSECT and see where they lead. Try the PSATNEW field at offset +21C (decimal 540).

Just point the cursor at the fullword address at +21C, type (or RETRIEVE) the *BROWSE* command and press enter.

You are now at your TSO Session's TCB (Task Control Block, DSECT IKJTCB).

Can you see how this powerful command might come in handy when debugging and investigating things?

Conclusion

There are also a number of line commands available in the DDLIST utility. I didn't cover them because they are related to the initial screen of allocated datasets. They are all explained in the tutorial.

I find it fascinating how the DDLIST tool has just morphed into a powerful system-wide workbench with its ultra-slick set of primary commands. Some have little to do with "dataset allocations" but who cares?

For me, DDLIST has become an indispensable tool in ISPF. I hang out there almost as much as places like Option 3.4 (DSLIST). Try it and see if you can exploit it yourself.

Jim Moore is the editor of The TSO Times and wants to say that his upcoming November 2004 Working Smarter column for Technical Support (NaSPA) magazine is entirely devoted to the Browse command of DDLIST.



One Maple Street
Hanover, NH 03755

Presorted Standard
US Postage Paid
New London, NH
Permit #11

TSO Times

For the Power ISPF User

Have you visited www.tsotimes.com yet? Several of the articles in the paper edition are continued online. The web site is also the place to go for code downloads, examples and other interesting articles that appear ONLY on the web site. Have a look. You'll be glad you did.

TSO Times Advertising

Rates

Full Page (7 1/2" x 10")	\$500
1/2 Vertical (3 1/2" x 10")	\$400
1/2 Horizontal (7 1/2" x 5")	\$400
1/4 Vertical (3 1/2" x 4 3/4")	\$150
Business Card (3 1/2" x 2")	\$75

Please submit your ad in one of the following PC-compatible digital formats: Illustrator, Photoshop, PageMaker or Freehand. Images/graphics should be saved in one of the following formats: .eps, .tif, .jpg, .bmp, or .wmf and have a minimum resolution of 300 dpi/ppi when scanned or scaled. You may send your ad on a cd-rom, floppy or Iomega ZIP disc to Chicago-Soft, LTD, ATTN: TSO Times, One Maple Street, Hanover, NH 03755 or by email to jbergeron@chicago-soft.com.

The TSO Times has a circulation of 15,000 people.

Publication Schedule

Spring 2004 / Fall 2004 / Spring 2005 / Fall 2005

FAXBACK - 603-643-4571

To receive more information or product trials, simply fill out this form and fax it to Chicago-Soft, LTD at the number above. You can also go to our website at www.chicago-soft.com.

Please send me:

- More Information on:
- MVS/Quick-Ref™
 - Quick-Ref for Windows™
 - Web/Quick-Ref™
- FREE Product Trial for:
- MVS/Quick-Ref™
 - Quick-Ref for Windows™
 - Web/Quick-Ref™

Name: _____

Title: _____

Company: _____

Address: _____

City/State/ZIP: _____

E-mail: _____

Phone: _____

Fax: _____