Alice in Packageland

Brent Hawks
IBM APL2 Development
555 Bailey Avenue
San Jose, CA 95141
408/463-3588

Installation Code: IBM
Project: APL
Session: D60

Abstract
API.2 can take you to a marvelous new place called "packageland." Alice fell into a package (rabbit?) hole and is confused about what packages are, what to package, how to do it, and how to use it. We'll go in after Alice, and find a fascinating world of new ways to make API.2 more productive.

Once Upon a Time...
API. was originally confined to working only with objects within the workspace. When API.2 V1 R2 came along you could then access objects in Assembler, FORTRAN, and REXX. Now, with "the then current release" of API.2 you can add API.2 to that list.

I must be dreaming...
During the PIP (field introduction program) we had one customer give us a fantastic report after using packages. Several API.2 Release 2 workspaces were converted into API.2 Release 3 packages. The customer reported exciting improvements in elapsed time, CPU time, and programmer productivity. These gains were attributed to sharing code among multiple users, the ability to eliminate function files (resulting in simpler code), and the need to only maintain a single copy of a packaged workspace.

What is a package?
A package is simply another form of a saved workspace. Using )SAVE you store the active workspace out onto disk. The PACKAGE function takes the saved workspace and makes an object deck out of it. You can then use the linkage editor to make an object module out of it. However, it is still just a saved workspace, we've just put it into a wrapper that the operating system can understand.

You can get at objects in the packaged workspace with )NA. For example, 3 11 )NA 'TIME' will get you the TIME function from the TIME packaged workspace. If you do )FNS you'll see TIME; however, all you really have in the active workspace is a pointer to the TIME function in the package (if you try )CNR TIME, the result will be an empty matrix). The function itself is still in the package. The package, if not already in memory, will be loaded when you do the first )NA to the package.
A simple packaging example.

It's quite simple to package a workspace. For example, we can quickly package the DISPLAY workspace:

```plaintext
)CLEAR
CLEAR WS

3 11 DNA 'PACKAGE'
1 PACKAGE 'DISPLAY V00000001'
DISPLAY TEXT A
)HOST LINKED DISPLAY (LIBE PKGLIB
CMS(0)

)CLEAR
CLEAR WS

'PKGLIB.DISPLAY' 11 DNA 'DISPLAY'
1 'PKGLIB.DISPLAY' 11 DNA 'DLX DLX'
1 DISPLAY DLX

A+ 'ABCDE'

DISPLAY A

A 'ABCDE'

a PACKAGE A SAVED WORKSPACE
a PACKAGE IT
a LINK EDIT IT

a NOW DNA TO THE PACKAGE
a USE FN AND VAR FROM PACKAGE

+------
| COIBM |
+------

A+ 'ABCDE'

a VAR FROM THE ACTIVE WORKSPACE

a USE FN FROM PACKAGE

You would normally "clean" the workspace before packaging it. You clean it by )COPYing it into a clear workspace, resetting the system variables, and then resaving it.

The example is from CMS (you can tell because the result of the PACKAGE function is a TEXT deck). It turns out that in CMS, you can use a TEXT deck without doing the linkedit, but it is NOT recommended.

In the example we find out what DLX in the DISPLAY workspace is. Doing )NMS in the workspace would give you DLX. 2 A. 2 DISPLAY. 3. However, only A actually exists in the workspace, the others are pointers into the packaged DISPLAY workspace.

You'll notice that we could not do DNA to DLX directly. A quad name may only be accessed via DNA as a surrogate name. Also notice, that DISPLAY and DLX act as if they were in the active workspace.

Why Use Packages?

There are several advantages to using packages:

- Name Isolation
- Run Only Applications
- Shared Code

Name Isolation. Many applications that expect to share a workspace must go to extraordinary lengths to keep from having name conflicts. For example, look at the names of things in the PRINTWS workspace; I tried to print out a workspace that contained a function called HOST; it turns out that PRINTWS also has a function called HOST, so mine got obliterated. If PRINTWS were in a package, then it could use its own HOST function and print mine at the same time.

Run Only Applications. It's easier to hide API from a user, when using packages. It is also possible to make decommenting and unmeaningful names an automatic part of the packaging process.
**Shared Code.** There are a number of advantages to sharing code among multiple users and workspaces.

- Maintenance
- Performance
  - Eliminate Function Files
  - Reduce storage requirements

The customer noted above attributed the gains to the use of shared code. The performance was due to the elimination of function files, and the productivity came from reduced maintenance costs and ease of implementation (versus function files). Please understand, the package is still APL and runs at the same speed as APL in your active workspace. The performance improvements are due to things such as reduced paging (multiple users share one copy of the package), less logic required than for function files, less I/O, etc.

**What should I package?**

Obviously, anything that you want to share (amongst users or workspaces) is a candidate for packaging. Things that you wish to hide from the active workspace (maybe to avoid clutter or name conflicts) or utility functions that require frequent maintenance; “end user” applications are also good candidates for packaging.

**Inside of Packages**

Now we’ll talk a little about name isolation versus localization, changes in a package name scope, and moving around between name scopes.

**Name Isolation**

Name isolation is NOT the same as localization! Within the active workspace is a name table that points to the objects in the workspace. The packaged workspace also has a name table that points objects within the packaged workspace. When you switch execution from the active workspace to the packaged workspace, you work from the name table in the packaged workspace.

This means that if you have a function FN that you have accessed via ON then the name table in the active WS (workspace) will have an entry for FN that points to the name table in the package which actually points to FN in the packaged WS.

Now suppose that we have variables A and B in the active WS and variable B in the packaged WS. Now when we execute FN, we switch name tables and FN will only see the B that exists in the package. An attempt to reference something called A will result in a VALUE ERROR because A exists only in the active WS. If FN changes B, it will change only within the context of the package, the B that’s in the active WS will remain untouched.

**Changes in a packaged WS**

As you may have noticed above, FN can CHANGE things in the package. However, the package may be in non-writable memory, and anyway, I may not want other users of the package to see the change. So, what do I do?

It so happens that when a package is first accessed, its name table is copied into the active workspace. It still points to objects in the package, but now I can make changes to the name table and they affect only my active WS, not someone else’s. Now if anything in the package changes (or is created), then the new/changed version is put into the active workspace and pointed to by the package’s name table (not the active WS’s name table), so it acts as if the package had changed.

If I SAVE the active WS after having made changes in the package, the changes will still be there when I re-LOAD the workspace.

One point of interest. In the CURRENT implementation of packages, any variable that gets referenced (not just new/changed) will get copied into the active WS. Functions are copied only if new or changed.
Packages can call other packages
We already know about ONA as a means for accessing a package from the active WS. It can also be used to access a package from a package, or to access the active WS from a package. There is an external function EXP that can be used to access the \textit{previous} name scope. Consult the System Services Reference and the Using Supplied Routines manuals for more information about the syntax of ONA and EXP.

Stopped in a package
Sooner or later, something will go wrong and you will get a SYNTAX ERROR in some function that's in a packaged workspace. As is usual in APL2, execution will stop inside the function at the point of the error. It is IMPORTANT to remember that APL2 is now using the name table from the package and not the one from the active WS. So, if you do )NMS you will see the names of objects \textit{in the package}. You can edit things, create/change/delete things, run things; in short you can do what you normally do in APL2 - you're just doing it using the name table of the package. )RESET will get you back to the active WS. Any changes that were made in the package name scope will remain. If you subsequently )SAVE the workspace, the changes will still be there when the workspace is again )LOADed.

Packages can make things easier (an example)
Suppose that you have two versions of the same workspace and you would like to know what's changed. Without packages it's rather difficult, you have to get around name conflicts somehow.

In the appendix you can see a sample workspace that will compare the objects in two workspaces and tell you which ones are different. It will even work on itself! The COMPARE workspace only has about four simple functions that do the actual comparisons. The rest of the workspace is documentation, some code to allow packaging to work in CMS or TSO, and some code that allows COMPARE to be packaged.

It works by packaging the two workspaces to be compared and then accesses the objects in the two packaged workspaces with ONA. Name conflicts are avoided by using surrogate names. It will even compare the system variables.

Other interesting tidbits
If you're going to be serious about packaging you'll also want to know about the optional left argument to the PACKAGE function. It's a list of names that are accessible (via ONA) from outside of the package. If the namelist is missing or empty, then the default is to allow all names to be accessed (including system "quad" names). The external function EXP, because it is reaching back to the previous name scope, ignores this name list.

If you do use a name list on a package, you should allow some room for debugging. If you make the namelist too restrictive, you may find it difficult to isolate a problem.

When using the PACKAGE function in TSO you will need to allocate a SYSPUNCH data set. This is where the PACKAGE function will put its object deck. On both CMS and TSO, you will do well to get the OS/VS Linkage Editor manuals.

Finally, if you access a package that is NOT shared (i.e. in a DCSS in CMS or the IPA in TSO) then it will be loaded into your free space. That means, that your memory requirements may change. For example, you may need to use a smaller workspace and a larger free space.
Summary
Packages are simply API.L2 and they do just what you want them to do. Name scopes may be a little confusing (just like the first time you saw indirection or recursion), but are very powerful.

Remember that one customer reported major improvements in elapsed time, CPU time, and programmer productivity when API.L2 Release 2 workspaces were converted to API.L2 Release 3 packages. The customer attributed the gains to: sharing code among multiple users, the ability to eliminate function files, simplified logic, and the need to maintain only a single copy of a packaged workspace.

Packages allow name scope isolation. You have the ability to put out end-user "run only" applications. Users can now SHARE the same copy of API 2 code (without function files) for the first time in APL history. Finally, you may find it a great deal easier to maintain utilities functions as packages. You can maintain a single copy of the utility and all workspaces will use the same code.

Appendix

COMPARE WORKSPACE
(C) COPYRIGHT IBM CORP., 1987

ABSTRACT

COMPARE the objects in two workspaces for differences.

CMP_LOAD

<table>
<thead>
<tr>
<th>PKG</th>
<th>LOAD</th>
</tr>
</thead>
</table>

CMP_OBJ

<table>
<thead>
<tr>
<th>PKG</th>
<th>OBJ</th>
</tr>
</thead>
</table>

CMPLIB

| PKGLIB |

DESCRIBE

The primary purpose of this workspaces is for COMPARING the objects in two workspaces (most of which will be identical) in order to see any differences.

This workspace should normally be LOADED rather than COPIED because it may create objects in the workspace which could cause name conflicts and/or loss of data.

***** NOTE *****
COMPARE creates data sets that may OVERWRITE existing data sets (see warnings in HOW and in PKG).

HOW

***** COMPARE *****
To COMPARE two workspaces, the syntax is:

'wsid_a' COMPARE 'wsid_b'
If the workspaces have objects of the same name that do not compare
then it will create a variable F_name or V_name (F for functions or
operators and V for variables) that is a two row matrix of the
form: F_name=2 (OCR wsd_a.name | OCR wsd_b.name).
For variables, it will just be the value of the variable.
Note also, that for functions/operators, the value for: 2DAT name
will be appended as the last line of the OCR.

Objects which are in only one of the WSs are referred to as
ORPHANS; you'll see a list of names (if any) for each WS.

The system variables (except: DAV, DMC DTS OKA) are also compared.
Any differences are reported and a variable is created with a
prefix of QOC (for example: QOC_10 for O10). These are the same
as above: two element variables containing the value from each WS.

***** SHOWDIF *****
The function SHOWDIF will try to show the differences between
two objects in one of the created variables. For example:
Z=SHOWDIF F_FN
will show the lines from the two versions of FN that don't match.
The lines from FR in the left workspace will be the first item
left of the result. SHOWDIF will return the differences side by side
if they will display within DFW (a two element vector), otherwise
it will leave the result as a two row matrix.

An optional left arg on SHOWDIF will cause SHOWDIF to decomment
its arguments before the comparison. Only the existence of
a left argument is checked; its content is irrelevant.

***** WARNING ***** WARNING ***** WARNING ***** WARNING *****
The PKG function in this workspace will create data sets that may
OVERWRITE existing data sets. See the PKG function for details.

[0] COIBM
[1] '(C) COPYRIGHT IBM CORP, 1987'
[3] * GLOBALS: L_WS R_WS @A @B @C ACRB DCS DATA DAT FR
[4] * FIRST PACKAGE THEM AND DON'T TO THE DNI, OCR, AND DAT OF EACH
[5] 'Packaging WS'S....'*
[6] (V=PKG"(L_WS R_WS)-F" | "R") /0
[7] * QUIT IF CAN'T PKG
[8] OP R @A @B DCS DAT
[9] GET DNI FROM I, R
[10] OP R @A @B DCS DAT
[11] IN CASE I'M A PACKAGE
[12] (L R)-4L"R" TO MAKE DISPLAY LINE UP NICELY
[13] * DO OPERATORS AND FUNCTIONS
[14] *...... Comparing Functions/Operators......'
[15] T-(FA-(FR-(2)@A @B 3 '))-"")+(FR-(2)@A @B 3 ')-""
[16] * COMMON FNS/OPS
[17] FA-(-V/T)/FA
[18] * FNS/OPS FOUND ONLY IN WS_A
[19] FB-(-V/T)/FB
[20] * FNS/OPS FOUND ONLY IN WS_B
[21] $0<FA>/'Orphan Fns/Ops in 'L': 'FA'
[22] $0<FB>/'Orphan Fns/Ops in 'R': 'FB'
[23] COMPARE_FNS"FB" SEE IF THEY ARE THE SAME
[24] * DO VARIABLES
[25] *...... Comparing Variables......'
[26] COMPARE_SYS
[27] * COMPARE SYSTEM VARS FIRST
[28] V=(V=-(2)@A 2 '-'))+(V=(2)@A 2 '-'))
[29] V=-(V/T)/V
[30] * COMMON VARIABLES
[31] V=(V/T)/V
[32] * VARS FOUND ONLY IN WS_A
[33] V=(V/T)/V
[34] * VARS FOUND ONLY IN WS_B
[35] $0<VA>/'Orphan Variables in 'L': 'VA'
[36] $0<VR>/'Orphan Variables in 'R': 'VR'
[37] COMPARE_VAR"VAR" SEE IF THEY ARE THE SAME
[29]  "CLEANUP"
[30]  -OpKGDELETE L;R
[31]  -OpEx"X'K.'S"'F_WS'"LNI'A'"LNI'B'"LGRA'"LCRB'"DATA'"SAB'"EXP'"F
[0]  COMPARE FNS R;A:B
[1]  • SEE IF COMMON FNS/OPS ARE IDENTICAL, IF NOT, PUT THEM INTO CALLER
[2]  A=ACRA R=(xR)~' 
[4]  -1(A=-_B)/0 • THEY MATCH
[5]  • ELSE CREATE THEM IN THE CALLER'S NAMESPACE
[7]  A=3(e[21A]), (e'n) v, Δ DATA R) • PUT TIME-STAMP AT THE END
[8]  B=2(e[21B]), (e'n) v, Δ DATA R)
[9]  -OpEXP('F', 'R')='2 1pA B)
[0]  COMPARE_SYS;DECA;DECB;SYSVARS'A:B
[1]  • SEE IF SYSTEM VARS ARE SAME (EXCEPT: 'DA' 'DC 'TSO 'NA)
[2]  SYSVARS=040VCTDENV'OFC00100D10I0I.TOPFOPKOPROF1005FOTOF0 T0U.
[3]  SYSVARS=(1 SYSVARS='0')SYSVARS
[6]  A=DECA' SYSVARS
[7]  B=DECB' SYSVARS
[8]  -(A=-_B)/0 • THEY ALL MATCH
[9]  • ELSE SOME ARE DIFFERENT
[10]  (SYSVARS A B)= (-A=-_B)/"SYSVARS A B
[11]  ' 'System Vars that don't match:' SYSVARS
[12]  LOOP:=(0=SYSVARS)/0
[14]  (SYSVARS A B)=1+"SYSVARS A B
[15]  • LOOP
[0]  COMPARE_VAR R;A:B
[1]  • SEE IF COMMON VARS ARE IDENTICAL, IF NOT, PUT THEM INTO CALLER
[4]  -(A=-_B)/0 • THEY MATCH
[5]  • ELSE CREATE THEM IN THE CALLER'S NAMESPACE
[0]  Z=L SHOWIF R;A:B;C;O:O
[1]  • SHOW DIFFERENCES BETWEEN THE TWO (SIDE BY SIDE IF CAN)
[2]  • 0=1
[3]  L=2=ONC 'L' • LEFT ARG PRESENT
[4]  C=p'S'=R • ENSURE THAT EACH PART OF R
[5]  R='(2i'1', 'C)'p'R • IS A MATRIX
[7]  $L/'A'+'(+'"1'='A'"1'='A')"L' • ELIMINATE COMMENTS IF LEFT ARG
[8]  $L/'B'+'(+'"1'='B'"1'='B')"L' • RIGHT ARG
[9]  C=(A-DIL'T')A)==-(_R-DIL'T') • COMPARE THEM (W/O LEAD/TRAIL BLANKS)
[10]  Z=D((+1')/A • LINES IN A THAT DIDN'T MATCH
[11]  Z=D((+1')/)B • LINES IN B THAT DIDN'T MATCH
[12]  Z=(C="1pZ')/7Z=2 1pZ' • SHOW SIDE BY SIDE, ELSE 2 1 MATRIX
[0]  Z=DIL'T R
[1]  • DELETE LEADING/TRAILING BLANKS
[2]  Z=(('\n')h\h2'=' ')/R
[0]  Z=L, DNA R
[1]  • PROVIDES DNA COMPATIBILITY IN CMS AND TSO
[2]  Z=((('TSO'='='HOST)/CMPLIB'='),')/11 DNA R
[0]  R=PKG_WS;CMDS;PACKAGE;CTL;DAT;XA;T
[1]  • CONVERTS WS TO A PACKAGED WORKSPACE
[2]  • TSO LEAVES SYSFUNCH (CMF_OBJ) AND CMPLIB (CMF_LDB) ALLOCATED
[3]  • NAMING CONVENTION USED FOR WORKSPACES IS: V,W,SHARE
[4]  • CMPLIB IS A SIMPLE CHAR VEC (E.G. CMPLIB='PKGLIB')
[5]  • IT IS THE FILENAME OF THE LOAD LIBRARY
[6]  • CMF_OBJ IS A TWO ELEMENT VEC OF VICS ('CMF_OBJ' 'CMF_OBJ')
[7]  • 'SYSFUNCH' WILL BE ALLOCATED TO CMF_OBJ
IT WILL CONTAIN THE OBJECT DECK CREATED BY "PACKAGE"

CMP_LOAD IS A TWO ELEMENT VEC OF VECs (i.e. CMP_LOAD=PACK& LOAD)

CMPLIB WILL BE ALLOCATED TO CMP_LOAD

ITS MEMBERS WILL BE THE PACKAGED WORKSPACES

*** NOTE ***

CMS

"PACKAGE" CREATES A FILE: wsname TEXT A. A PRE-EXISTING FILE OF

THAT NAME WILL BE OVERWRITTEN. "PKGDELETE" WILL ERASE THIS FILE.

TSO

THIS FUNCTION ONLY CHECKS TO SEE THAT 'SYSPRINT' AND CMPLIB

ARE ALLOCATED, IT DOES NOT VERIFY THAT THEY ARE ALLOCATED

TO CMP_OBJ AND CMP_LOAD RESPECTIVELY.

THIS FUNCTION DOES NOT PROTECT ANY PRE-EXISTING DATA IN

CMP_OBJ AND CMP_LOAD. HENCE, YOU MUST PROTECT AGAINST Destroying

DATA SETS. ADDITIONALLY, THE FUNCTION "PKGDELETE" DELETES

THESE DATA SETS.

+:('TSO'=-_xHOST)/TSO

+(1=P-3 11 DNA 'PACKAGE')/ERROR

+(0=P-PACKAGE WS, 'APL'SV2')/ERROR

PACKS THE WORKSPACE

R=0

EXIT

TSO:=-O9102 DSOV 360 'CTI.DAT' SEE IF WE ARE IN MVS OR MVS/XA

C:T-0 CHECK THE WORKSPACE ADDRESS

XA=(16x1024x1024)5DAT IN XA IF WORKSPACE ABOVE THE LINE

0=O9102 DSOV 'CMD'

CMD=APL DSI SYSPRINT SEE IF SYSPRINT ALLOCATED

+(=-8 __4CMD)/I1 BRANCH IF ALLOCATED

CMD=APL DSI 'CMPLIB' SEE IF DATA SET EXISTS

+(=-8 __4CMD)/1 BRANCH IF IT EXISTS

ALLOC NEW DATA SET

T='ALLOC F(ISYSPUNCH) DSN('CMPLIB ') NEW SPACE(S,5)

CMD=T 'TRACKS D(_IDIR(2) BLSIZE(4096) RCIPM(U) TSO(O))

+(0=R=CMPLIB)/12

ERROR

L:CM=ALLO F(ISYSPUNCH) DSN('CMPLIB ') SHR' ALLOC OLD DATA SET

(0=R=CMPLIB)/ERROR.

L2:=(1=P-3 11 DNA 'PACKAGE')/ERROR

+(0=R-PACKAGE 'V,'WS)/ERROR

PACKS THE WORKSPACE

CMD=APL DSI 'CMPLIB'

+(=-8 __4CMD)/L1 BRANCH IF ALLOCATED

CMD=APL DSI 'CMPLIB' SEE IF DATA SET EXISTS

+(=-8 __4CMD)/L3 BRANCH IF IT EXISTS

ALLOC NEW DATA SET

T='ALLOC F(CMPLIB ,') DSN('CMPLIB ') NEW SPACE(S,5)

CMD=T 'TRACKS D(_IDIR(2) BLSIZE(4096) RCIPM(U) TSO(O))

+(0=R=CMPLIB)/12

ERROR

L3=CM=ALLO F(CMPLIB ,') DSN('CMPLIB ') SHR' ALLOC OLD DATA SET

(0=R=CMPLIB)/ERROR

L4: OKAY, NOW LINKEDIT THE MESS

T='LINK ,('CMPLIB ,') LOAD('CMPLIB )

CMD=T('WS, ') NOTE: NOPRINT , 'XA,'RMODE(ANY)

+(0=R=CMPLIB)/0

ERROR: 'PKG ERROR ', a

Z=PKDELETE NAMES:CHK;CMD;R

Z=O IF EVERYTHING WAS PROPERLY DELETED OR NEVER THERE

IN CMS, DELETES TEXT DECKS SPECIFIED BY NAMES VECTOR

IN TSO, FORCES PROCESSOR 11 TO CLOSE THE OPEN LOADLIB

THEN DEALLOCATE AND DELETES SYSYMPH AND CMPLIB

O=OPFX '2=CHK C 'CMPLIB' '2=CMPLIB' FOR READABILITY

Z=Z-2=-_100 DSOV 'CMD'

+(CMS'==-_XHOST)/CMS

R=OTHER,OTHER' 11 DNA 'OTHER'

Z=Z+V/O 12=CHK 'FREE F(ISYSPUNCH)'

Z=Z+V/O 12=CHK 'FREE F(CMPLIB ,')

Z=Z+V/O 2=CHK 'DELETE', 'CMPLIB'


```
0) R=HOST;CTL;DAT   # OBTAIN HOST ID VIA AP102.
[2] * WARNING: THE CONTENTS OF THIS FUNCTION ARE SUBJECT TO CHANGE
[3] * BETWEEN AP12 RELEASES.
[4] * RESULT: NORMAL - ENCLOSED CHARACTER VECTOR OF HOST NAME
[5] * ERROR - DEGREE OF COUPLING OR AP102 RETURN CODE
[6] * NOTE: (DAT+240) = (X'FO' IN WS) = (HOST INFO IN AP12 1.1.00)
[8] * SET EVENT TIMER TO 5 SECONDS.
[10] L1:=(Z*=R=102 DSVO 2 3o;CTL;DAT')/L2  # IF SHARE WORKS.
[11] +(O*DSVE)/L1  # IF EVENT OCCURS.
[12] -O  # RETURN WITH DEGREE OF COUPLING.
[13] L2:=(0*R=CTL)/CTL-DSVE=0  # IF AP102 ERROR.
[14] CTL=1,(DAT+240),"  # GET HOST SYSTEM DATA.
[15] R*(O*DAT)  # CONVERT LOW-ORDER BYTF TO RITS
[16] R=R/'TSO' 'CMS' '?32' '?16' '?28' '?14' '?22' '?11'

* A SAMPLE RUN COMPARING TWO VERSIONS OF COMPARE

LOAD COMPARE
SAVED 1987-11-04 14.20.07 (GMT-8)
(C) COPYRIGHT IBM CORP, 1987

'COMPARE' COMPARE 'COMPB'

Packaging WS'S ...

.... Comparing Functions/Operators ....

Orphan Fns/Ops in COMPARE : COIBM DLT HOST SHOWDF
Orphan Fns/Ops in COMPB : COMPARE_FNS HOWUSE SAYAWS SYSTEM

Mis-match in Common Fn/Op: DNA
Mis-match in Common Fn/Op: COMPARE
Mis-match in Common Fn/Op: COMPARE_FNS
Mis-match in Common Fn/Op: COMPARE_SYS
Mis-match in Common Fn/Op: COMPARE_VAR
Mis-match in Common Fn/Op: PKG
Mis-match in Common Fn/Op: PKDEL"E"TE

..... Comparing Variables ..... System Vars that don't match: DIX

Orphan Variables in COMPARE : ABSTRACT CMP_LOAD CMP_ORJ
Orphan Variables in COMPB : WSID

* SHOW BOTH VERSIONS OF "DNA" FUNCTION

F_DNA

Z=L DNA R
* PROVIDES DNA COMPATIBILITY IN CMS AND TSO
Z=(((TSO"=_xHOST')/CMPIR.'").,)11 DNA R
* \1987 11 3 10 16 24 502

Z=L DNA R
* PROVIDES DNA COMPATIBILITY IN CMS AND TSO
Z=(((TSO"=_SYSTEM')/PKGIR.')",,)11 DNA R
* \1987 3 13 10 5 31 0

* SHOW BOTH VERSIONS OF "DIX" SYSTEM VARIABLE

QD_LX  # THE ONE FROM "COMPB" IS EMPTY

COIBM
```