System Programmer's Guide to Tailoring Your APL2/TSO System

18 February, 1988

Ray Trimble

IBM Corporation M46/B25
P.O. Box 49023
San Jose CA 95161-9023

IBM SJ 167

Preface

Installation and customization of APL2 under TSO involves a large number of APL2/TSO installation options, installation exit points, PROCLIBs, PARMLIBs, LOGON procs and CLISTs. The goal of this paper is to help you make the right decisions in installing, customizing, and maintaining APL2 so that:

- your users see a fast and smooth APL system;
- they can get at what they need, but not what they shouldn't;
- you can maintain accountability of resource use; and
- the system runs itself for the most part.

Note: Those are goals, not promises, and this paper can at best provide help, not a panacea.

The topics to be covered include:

- "Workspace Library Choices" on page 1.
- "CLISTs and Logon PROCs" on page 5.
- "Invocation Options and their Defaults" on page 7.
- "Changing Installation Options" on page 13.
- "Installation Exit Routines" on page 17.

Workspace Library Choices

The Nature of SAM Library Support

When workspaces are kept in SAM libraries, each workspace is stored as a separate MVS dataset, written and read using the Basic Sequential Access Method (BSAM). Blocksize is installation selectable, but is always forced to a multiple of 80 bytes.

The grouping of workspaces into libraries is done by using a specific dataset naming scheme.[1] This makes it possible to use the MVS system catalog as the directory through which all workspaces are located. An implication of this is that a single APL library structure applies to all users of the computer complex.[2]

APL2 does its own dynamic allocation, both to create new workspace datasets and to access existing ones. There is a sizable list of installation options controlling where and how new datasets are created. Once a dataset has been created, that same dataset is reused for all updates to the workspace. This ensures that any RACF controls placed on the dataset will be retained across *)SAVE*. Since each *)SAVE* is a complete replacement, it also means, however, that the moment a *)SAVE* has started, the previous version of the the workspace has been destroyed.

Three classes of workspace libraries exist, with somewhat confusing names.

PRIVATE libraries contain workspaces which are only known to a single TSO user, or a group of TSO users who share the same TSO PROFILE PREFIX.[3] Each user has one private library, which is always his library 1001. This is the one exception to the rule that all APL users see the same set of libraries.

PUBLIC libraries contain workspaces that are all owned by the installation. Ordinary users typically cannot update any workspaces in these libraries. Prior to Release 3, libraries 1 through 999 were always public libraries. Beginning with Release 3 a new installation option can be used to specify the upper limit of the public library range.

PROJECT libraries contain workspaces which are owned by individual users but may (RACF permitting) be accessed by other users. Specifically, the owner's PROFILE PREFIX is used as the high level qualifier for the workspace dataset name. All workspaces within a single project library are owned by the same PROFILE PREFIX. All library numbers that are not public or private, by the rules above, are treated as project libraries. The first user to save a workspace in a particular library becomes the owner of that library from then on.[4]

Note that a user's *default* library (the one assumed if a library command omits the library number) is always ↑⎕*AI*. That in turn is controlled by the ID invocation option, or in some cases an installation exit. If 1001=↑⎕*AI* then the default is a private library, and its workspaces cannot normally be accessed by any other users. If 1001<↑⎕*AI* then the default is a project library, and the user's "private" workspaces (to use conflicting terminology from the past) can be accessed by others who know his ID number and have the appropriate RACF authorization to his PROFILE PREFIX.

[1] APL2 Release 3 allows an installation exit to replace this scheme.

[2] Unless multiple copies of APL2 have been installed with carefully distinguished installation options.

[3] In most installations the PROFILE PREFIX is the same as the TSO user ID.

[4] Except that the LIBKEEP installation option can be used to indicate that empty libraries are to revert to an unowned status.

IBM SJ · 167

The Nature of VSAM Library Support

All of the workspaces in one VSAM library are stored within a single VSAM cluster (a KSDS to be specific). The association between APL library numbers and VSAM cluster names is made by ALLOCATE (or DD) statements provided by the user or installation, typically in an APL2 invocation CLIST. This user association means that there is no universal numbering scheme for workspaces in the complex. One user's library 7 can be the same physical data as another user's library 1234. And a third user can have a library 1234 which is completely different.[5]

The DDNAME used is "W*nnnn*," where *nnnn* is the library number with no leading zeros. "W0" can be used as a special case representing the library whose number is ↑⎕*AI*. On all workspace library requests the system looks first for the W*nnnn* DDname. The library is treated as a VSAM library if that DDname exists, or as a SAM library if it does not.

VSAM libraries are not created automatically by APL2. Instead, users or system administrators must create them with Access Method Services DEFINE statements or the equivalent TSO DEFINE commands. The DEFINE command has a complex syntax and an overwhelming number of parameters, but its simplest form might be:

```
DEFINE CLUSTER( NAME(my.name) MODEL(existing.name) )
```

Since VSAM controls space allocation within the cluster, there is no inherent reason why saving a new version of a workspace would have to begin by destroying the previous one. Unfortunately, that is exactly what happens in the current implementation.

Three classes of workspace libraries exist, with names that are not only confusing, but have different meanings than for SAM libraries.

PRIVATE libraries are kept open, once used, for the remainder of the APL2 session. This means they cannot be dynamically unallocated (TSO FREE), that only a single user can access a given library as private, and that no one else can depend on writing to that library. Private libraries are efficient for all library commands because there is no OPEN/CLOSE overhead for individual requests. Only the W0 library (or W*nnnn* where *nnnn*=↑⎕*AI*) is treated as a private VSAM library.

PUBLIC libraries that are only being read are kept open for the remainder of the APL2 session, but they are closed if a)*SAVE* is done into them. This means they cannot normally be unallocated dynamically, but that multiple users can write into the same library. Public libraries are efficient for)*LIB*,)*LOAD* and)*COPY*, but less efficient for)*SAVE*. Library numbers from 1 through 999 are always treated as public.[6]

PROJECT libraries are always opened and closed for each library command. This means they can be unallocated at any time, that multiple users can write into the same library, and that other users will always be able to)*LOAD* or)*COPY* the latest version of a workspace. But project libraries are less efficient than private or public libraries. All library numbers that are not public or private, by the rules above, are treated as project libraries.

Note that for VSAM libraries it is not the library itself which is private, public, or project, but the way it is being accessed in a particular APL2 session. It would be possible for three users to have concurrent access to a single VSAM cluster, one treating it as private, a second as public, and a third as project. It would, in fact, be quite typical for one user to access another user's private library as if it were a read-only public or project library.

[5] This has both advantages and disadvantages. See the next section for a discussion.

[6] The new installation option for SAM public library upper limit has no effect on VSAM libraries.

When is SAM better, and when is VSAM better?

The first point to be made is that this is not an either/or issue. In many installations a combination of some VSAM libraries with some SAM libraries may be the best approach. APL2 makes the choice dynamically for each user at the time of each library command, based on whether a *Wnnnn* DDNAME is allocated at that point in time.

For many installations the most important criterion is the effect on their DASD space management strategies.

- If HSM is used heavily, it is important to be able to migrate individual workspaces. Typically a library will contain a number of workspaces that are rarely used, but others that are used frequently. This is a strong argument for SAM libraries, which have separate datasets for each workspace.
- SAM dataset creation depends on static UNIT and VOLSER parameters which must be specified when the APL2 product is installed. Generic or esoteric units may be used, or the system may be allowed to default to public volumes, but these do not provide for volume selection based on userid, and often do not dovetail with installation rules for location of permanent datasets. Where this is a problem, manually defined VSAM clusters may provide the simplest solution. It is also possible to write installation exit routines to circumvent the problem.
- Installations which want to control the total DASD space allocated to each user may find that VSAM clusters are preferable. The system administrator can create a cluster of the proper size for each user, and dynamic dataset creation can be disabled. This is not feasible with SAM libraries, since it would prevent all *)SAVE*s except for existing workspaces.

Performance may also be an important criterion.

- SAM libraries require ALLOC/OPEN/CLOSE/FREE operations for each system command. VSAM libraries always avoid the ALLOC/FREE, and often the OPEN/CLOSE. (See the previous section for details.)
- The actual read/write operations are faster for SAM libraries than for VSAM libraries. (This is inherent in the DASD data structures currently used, but could change in the future.)

The net effect is that SAM is currently faster for large workspaces and for project libraries, while VSAM is faster for small workspaces being read from private or public libraries.

A number of other factors may also be critical in particular cases.

- SAM datasets have a rigid three-level naming convention which may violate the rules of an installation. This can be modified (beginning next year) by writing installation exit routines to create different dataset names. Or it can be avoided completely by using VSAM libraries.
- RACF protection for SAM libraries can be specified to the individual workspace level, with generic profiles that operate at either a user or library level. VSAM libraries can be protected only at a user or library level.
- Often an installation might want to maintain separate "test," "production," and "obsolescent" versions of a set of workspaces. With SAM, these must be kept in separate libraries. Libraries are normally copied as they mature, and programs or manual procedures must be modified to access different versions.[7] With VSAM, only the allocation need be changed. The applications and operating procedures are identical no matter which version is being accessed.

[7] The PUBQLFR installation option does allow some versioning.

- Some installations are so biased against VSAM that they will avoid it wherever possible. This probably does not apply to you, since if you were that biased you would have laughed when you saw the heading for this section, and skipped it completely.
- New library creation is automatic with SAM, manual with VSAM. Installations which don't want their users randomly grabbing new libraries may prefer the manual approach. Overworked system administrators would certainly prefer the automatic approach unless their users are sophisticated enough to do their own library creation. Users would undoubtedly prefer the automatic approach in all cases.

CLISTs and Logon PROCs

This section addresses only allocations, whether by ALLOC commands or DD statements. See also "Invocation Options and their Defaults" on page 7.

There are other DDnames you will often need to allocate, in particular ADMSYMBOL, AP2TN011, and (as of Release 3) APL2LANG. For batch jobs and TERMCODE(-1) you also need APLIN and APLOUT. These are handled by the installation process, and not discussed further here.

Providing for Trace and Dump Output

APL2 honors optional DDnames of APLTRACE and APLDUMP, but only when it is invoked. This is somewhat unfortunate, since it is usually not until later that you discover you would like to use the features. Sorry, but *)HOST ALLOC* will do you no good at all.

Trace output is controlled by the TRACE invocation option, but this is frequently modified dynamically using *)CHECK SYSTEM TRACE*(*numbers*). If no APLTRACE DDname existed when APL2 was invoked all trace output is directed to the user's terminal. This is really what you want anyway for interactive debugging. In general it is probably better to *omit* the APLTRACE DD unless you know before invoking APL2 that you will want to record trace output.

APL dumps come in several flavors. One kind may appear on the user's terminal along with a *SYSTEM ERROR* message, and may be accompanied by a DUMP*nnnn* workspace being saved. These "dumps" are associated with problems in the APL2 interpreter or the internal structure of the active workspace. They always go where they will go, and are unaffected by any DDnames that may be allocated.

If the APL2 system detects an error outside of the interpreter it usually attempts to produce an MVS SNAP dump. It is this dump which uses the APLDUMP allocation. If there is no APLDUMP DD, the dump is simply bypassed, APL2 recovers to the best of its ability, and the only diagnostic information available will probably be a single cryptic message. If you want to have your problems fixed, we **strongly recommend** that you include an APLDUMP allocation in all of your APL2 invocation procedures.

There is a third class of errors, those that APL2 is unable to detect. These include errors in other products called by APL2, as well as errors in critical parts of APL2 itself while responding to other errors. This class of errors will normally result in an attempted MVS ABEND dump. Like all such dumps, MVS will attempt to use SYSUDUMP, SYSABEND, or SYSMDUMP to record the dump, and will also produce an indicative dump at the user terminal. Presumably standard installation procedures for TSO sessions will cover this class appropriately. From an APL2 viewpoint problems in this class are quite rare.

Spill Files for)COPY

While processing *)COPY*, *)MCOPY*, or *)PCOPY* commands the system needs space to manipulate the source workspace, the active workspace, and intermediate forms of the copied data, all concurrently. To the extent possible this is done in virtual storage within the user's address space. If that is not adequate, spill files will be written to the user's private APL file library if it exists.[8] If no file library is allocated, or it is not large enough, spill files are written using the CPYSPILL and CPYSWAP DDnames.

These are temporary files which are written and read in a strictly sequential order. Any direct access (or even tape!) storage would work, but VIO is probably the most appropriate. In most installations the two files should be allocated as a standard part of the APL2 invocation procedures. The maximum size of the CPYSWAP file is the size of the active workspace. The theoretical limit on the size of the CPYSPILL file is much higher, but in practice a similar size is normally adequate.

Accessing Modules in Private Libraries

In some installations, part or all of the APL2 code itself may be in private libraries, not in LPA or the LINKLIB concatenation. It is frequently true that user programs called from APL may be in private libraries. APL2 provides a LOADLIB DD to help with such problems. But the behavior of this file is somewhat confusing.

First, it is obvious that no filename passed to APL2 can help in locating the primary APL2 load module itself. That must be in LPA or LINKLIB or a STEPLIB defined in the logon PROC.[9]

Once the APL2 module has been loaded and invoked, many other modules called by it can be located through LOADLIB. This includes modules brought in as a part of the APL2 invocation such as AP2TACTL, AP2INTRP, AP2TN11, AP2TYSTX, AP2TMEXC, AP2T127, AP2X104, and any other auxiliary processors. It also includes programs loaded by Processor 11, if the NAMES file entry does not specify the **:load** tag.

The situation becomes much more confusing for commands and CLISTs invoked by AP 100. Whether LOADLIB is searched depends on:

- whether APL2 is invoked under ISPF,
- whether you are running TSO or TSO/E, and if TSO/E, what release and modification level of it,
- what release of APL2 you are using and what PTFs have been applied,
- whether a command or a CLIST was specified to AP 100, and
- which variation of AP 100 command syntax was used.

It is probably not worth the trouble to try to fill in all data points in that five dimensional array. Our general direction, however, is away from using LOADLIB unless explicitly requested in the syntax of the AP 100 request. The "**APL ATTACH** command" has always used LOADLIB and will continue to do so. The "**TSO** command" has often used it in the past, and will do so consistently beginning with **Release** 3.

Finally, a comment about allocating LOADLIB versus the **LOADLIB** invocation option. The invocation option is precisely equivalent to doing an **ALLOC** with the **REUS** option during APL2 invocation, and a **FREE** during APL2 termination. Thus the invocation option overrides any earlier allocation I personally wish we did not support the the invocation option. We have an entire module devoted to it, and the function seems to be completely redundant. But compatibility arguments will probably force us to continue our support forever.

[8] This is a VSAM library which is also used for the log files maintained by the APL2 session manager, but is distinct from the private workspace library.

[9] One other possibility is to invoke APL2 from another program which uses a private load library. TSO TEST and ISPF both have this capability. Some installations also have a very nice little command which does nothing but this. Ours is called #.

Invocation Options and their Defaults

Invocation options can be supplied from a combination of three sources. It is important to understand how they are merged. The three sources, in the order they are considered, are:

1. The **DEFAULT** parameter of the **AP2TITOP** macro in the installation options module, **AP2TIOPT**.
2. The options specified on the APL2 command.
3. The **OVERIDE** parameter of the **AP2TITOP** macro.

Each of these sources provides a character string. The three strings are effectively catenated in the order shown above. The result may, of course, include the same keyword more than once. It does not matter whether the multiple references to a keyword came from different sources, or the same source. The combined string is processed from left to right, and in general the last option encountered replaces any earlier ones. (See "The Boolean Options: DEBUG, SYSDEBUG, and TRACE" on page 10 for an exception to this.)

Who am I? The ID option

APL has a long tradition of depending on a user number. That tradition has has become a language requirement in the first element of ⎕*AI*, the left argument to ⎕*SVO*, and the result of ⎕*SVQ*.

TSO, of course, assigns a user name instead of user number. The ID option is an attempt to resolve this incompatibility.

VS APL under TSO did not permit shared variables across the boundaries of each individual user's address space, so it was not important to have unique numbers for each user. Each user was arbitrarily assigned the number 1001. That is what applications saw in ⎕*AI*, and what auxiliary processors used when sharing variables with the APL session.

APL2 still defaults to **ID(1001)**, but users taking that default cannot share variables with other users or with global auxiliary processors.[10] Thus it is important in many installations to ensure that each user invokes APL2 with a unique **ID** value.

In most cases no security checking is done based on the **ID** number, so the only requirement is uniqueness. An algorithm within the invoking CLIST, a CLIST parameter, or a a separate CLIST for each user may provide an adequate solution.

An alternative is for an installation exit to provide the number, as discussed in "Invocation and Termination" on page 17. This alternative would be required if the installation is using global auxiliary processors (including global servers written in APL) that do authorization checking based on partner number.

Allocating Space: AISIZE, FREESIZE, SHRSIZE, SVMAX, WSSIZE, and XA

FREESIZE, **SHRSIZE**, and **WSSIZE** are the three primary values that you need to worry about.

FREESIZE is a very elusive quantity. It really means only "24-bit addressable space that will be needed for anything else during the session." This may include a great deal of code, depending on what programs have been installed in LPA and whether yours is an MVS/370 or MVS/XA system. In

[10] This kind of sharing is not permitted anyway if the optional GSVP has not been installed.

particular, it sometimes includes the APL2 interpreter (AP2INTRP), Access Method Services, and GDDM programs. It will also include other programs that need to be loaded dynamically (below the line) in the user's address space. And it includes much of the dynamic storage needed during the APL2 session. If a FREESIZE value is specified, it is only used as a check during APL2 invocation. The system will verify that there is enough available storage to get, as three separate blocks, storage for FREESIZE, SHRSIZE, and WSSIZE. If not, the APL2 session will be terminated immediately. There is no actual FREESIZE block kept after initialization. Since there is rarely any way to make a reasonable estimate of the requirement, my normal recommendation is to omit this option.

SHRSIZE and **WSSIZE** represent blocks of storage that are allocated statically for the duration of the APL2 session. In an MVS/XA system they are normally allocated in extended storage (above the 16Meg line). SHRSIZE should be at least 10K larger than the size of the largest shared variable value that will be used during the session. WSSIZE should be large enough for the largest workspace that is to be loaded as well as the dynamic storage that its functions will need during processing. The symptom of WSSIZE being too small is *WS FULL*. The symptom of SHRSIZE being too small is *SYSTEM LIMIT* with ⎕*ET*=1 7, or a shared memory space error code returned by an auxiliary processor.

The IBM-supplied defaults of **SHRSIZE(32K) WSSIZE(25%)** may be reasonable for an MVS/370 system, but they are probably too small for an MVS/XA system. You should consider changing them in AP2TIOPT. Note that the 25% is based on the TSO **SIZE** parameter, which defines only storage below the 16Meg line. For MVS/XA systems, the IEFUSR system exit is used to set the limits for storage above the line, with a default of 32Meg. The APL WSSIZE default in AP2TIOPT should be chosen based on the rules used by IEFUSR. If, for example, IEFUSR makes the storage above the line five times as great as that below the line, you might set WSSIZE(400%) as the default. Or, if the MVS default of 32Meg is retained, you might set WSSIZE(25M) SHRSIZE(5M).

One warning is in order here. Large workspaces do increase the paging load on the system, sometimes rather dramatically. If you are having paging problems, one early correction to try is to reduce the default WSSIZE.

AISIZE and **SVMAX** are mere drops in the bucket in contrast to SHRSIZE and WSSIZE. The default 512 byte AISIZE is really too small, and likely to cause problems for applications that make much use of AP 101. You can increase this to 8K or more without much chance of causing storage problems elsewhere. SVMAX is expressed as number of variables, not space, but it does imply 12 bytes per variable. The default of 88 is probably quite adequate.

XA seems like a ringer in this group. It does not specify any storage at all. But, on an MVS/XA system, it does determine where WSSIZE, SHRSIZE, and the storage used during *)COPY* is all allocated. Its default is 31-bit storage, but that is ignored on an MVS/370 system. In most cases this is exactly what you want. The real reason for the option is that some FORTRAN programs called through processor 11 may not be able to tolerate parameter data above the line. Any session that is going to call such a program will have to specify XA(24). But the installation default should normally be left as XA(31).

Terminal Options: DBCS, DSOPEN, PROFILE, SMAPL, and TERMCODE

For most situations these parameters should be left alone, at least as installation defaults. The default actions are to try to use the session manager, but revert to normal TSO terminal I/O if that fails. The default is to let GDDM or VTAM determine the terminal type (depending on whether the APL session manager is being used). If you have non-IBM terminals, or want to fake one terminal type on another, you may need to specify DSOPEN (for GDDM) or TERMCODE (for VTAM), DBCS to obtain the special Asian-language character support available in Release 3.

PROFILE could be changed to to provide, as a default, a session manager profile different from that supplied by IBM, while retaining the IBM profile for optional use. The IBM default is stored as "pubqlfr.DEFAULT.VSAPLPR," where "pubqlfr" is specifiable in the AP2TITOP macro parameters in AP2TIOPT. If, for example, PROFILE(LOCAL) was included in the default invocation options in AP2TIOPT, then initial session manager setup would be controlled by "pubqlfr.LOCAL.VSAPLPR."[11]

Of course some backward installations may decide they don't want their users to experience the joy of working with the APL session manager.[12] Such installations may want to specify SMAPL(OFF) in default invocation options, or if they are truly fanatical, in the override options.

User Preference: CASE, DATEFORM, and HILIGHT

The term "user preference" probably says all the installation programmer needs to know for this group of options, except that you may want to default DATEFORM to the format which is most common in your country. For Americans, this means changing the DATEFORM(ISO) shipped with the product to be DATEFORM(US).

Just because you are likely to get complaints from confused users, a word on CASE may also be in order. Preferred character set case is really a *workspace* attribute, not a session attribute. Once it has been set for a given workspace it cannot be changed, short of copying that workspace into another one with a different attribute. Workspaces created prior to APL2 Release 2 have a CASE(0) attribute. Every new workspace begins life as a *CLEAR WS*. The CASE invocation option is merely a means of adding an implied parameter to the *)CLEAR* command, indicating what the case attribute of that new workspace will be.

Running Applications: INPUT, QUIET, RUN, and TERMCODE(-1)

RUN is an option that you may not have seen yet, unless you are reading this paper retrospectively. It provides a simple means of automatically starting an application which is located via ⎕*NA*.

INPUT can also be used to start an application, either through ⎕*NA* or by *)LOAD*ing a workspace. It is a bit more complex, since each APL statement needed must be provided as a character string. This becomes particularly messy when quote (') characters are involved. By the time you get through CLIST processing and APL2 parsing it may take half a dozen or so quotes to get one through to APL.[13] Worst of all, a number of APL characters won't make it at all through TSO PARSE, which has its own ideas of what characters are valid, and what kind of folding is best for the user.

TERMCODE(-1) provides another way to drive an application, which avoids the pitfalls of INPUT. In this case the input APL statements are in a file, allocated using the APLIN DDname. However it carries the idea too far to please some application writers. The file *is* the terminal, at least so far as ⎕ or ⍞ input, or standard APL prompting is concerned. Any application interaction with the user must be in fullscreen mode, using GDDM or ISPF.

For anyone used to the CMS stack, it is ***not*** possible under TSO to stack APL input before invoking APL2. It *is* possible within APL2 to use AP 101 to stack commands that will be processed after exit.

[11] The PROFILE description in Chapter 3 of "System Services Reference" states that "VSAPLPR" is installation modifiable. 'Tain't so for TSO.

[12] There may be a slight bias to this statement. It should be read with an overlay of lighthearted self-deflation.

[13] It usually takes me half a dozen or so tries, too, to get the right number of quotes.

The QUIET option is a means of suppressing APL chatter (such as responses to *)LOAD* commands) that the application writer does not want the user to see. Beginning with Release 3 there are two fairly significant enhancements affecting this option:

1. It may be specified as QUIET(ON) or QUIET(OFF). QUIET is still accepted without parentheses, and means, of course, QUIET(ON).

2. A new processor 11 function, *OPTION*, lets an application test and set the QUIET option (either ON or OFF) dynamically.[14]

None of these options should be specified as system defaults, unless your installation uses APL for only a single application. But but they would frequently be provided in CLISTs used to invoke applications.

The Boolean Options: DEBUG, SYSDEBUG, and TRACE

These options are mavericks. Although often expressed as single numbers, they really consist of a sum of integers, each of which is a power of 2. APL2 (but not the CLIST processor) will let you express them either way, so that *option*(1 2) means exactly the same thing as *option*(3). You can even say *option*(1 3) and it still means the same thing. (No, that is not the same as *option*(4).) All of the other options *replace* any previous occurrences of themselves, but this group *ors* them together. So *option*(1) *option*(2) also means the same thing as *option*(3).

But how can you reverse a previous flag setting? By using a negative number. (Either an APL ¯ or an ordinary - is acceptable.)[15] Negative numbers follow the same power-of-two decomposition rules as unsigned numbers, but are applied by turning the corresponding flags off.

The **DEBUG** options are, in general, provided to assist in debugging user-written workspaces and auxiliary processors. Their use is as described in "APL2 Programming: System Services Reference." None of them would normally be set by default. In particular, DEBUG(32) is somewhat of a religious issue. Traditional APL programmers become very upset if they see

```
AP2ISSS220   SYNTAX ERROR
```

when they were expecting to see *SYNTAX ERROR*.

I would also warn against the temptation to set DEBUG(4). This produces much bigger dumps, but in our experience, somewhat less useful ones. Without DEBUG(4) APL2 chooses the areas it thinks are important in solving the problem. With DEBUG(4) APL2 tells MVS to dump the areas that MVS considers important. More specifically, without DEBUG(4) APL2 dumps the first and last 4K of the workspace and shared memory, the installation options module, the area around the PSW and register 14, SDATA(CB), and PDATA(SA,SPLS). DEBUG(4) adds SDATA(LSQA,Q,TRT), PDATA(ALLPA), the entire workspace, and all of shared memory. But it omits specific dumps of the installation options module and the areas around the PSW and register 14.

The **SYSDEBUG** options are intended to provide assistance in debugging the APL2 product itself. Their use is described in "APL2 Diagnosis Reference." Paradoxically, you probably want one or two of these options all the time.

- **SYSDEBUG(1)** degrades the system very slightly by activating an in-storage wraparound trace. However that trace table is often worth its weight in gold when analyzing dumps. The system, as

[14] You might suspect that with a name as general as OPTION, the function could do more than QUIET. You might be right.

[15] No, you can't say DEBUG(+ 2).

distributed, includes SYSDEBUG(1) in the default options in AP2TIOPT, and you should probably leave it there.

- **SYSDEBUG(16)** Tells APL2 not to bother checking for hardware features that are available only on certain machines. If you have some of those features on your machine, their use can improve APL2 performance. But if you do not have them, the tests can be costly. Each test causes a program check and, depending on the level of APL2 you are running, they may be repeated on every *)LOAD* or *)CLEAR*. The features tested currently (or in the near future) include Square Root, E to the X, Natural Log, Base 10 Log, and Vector Facility. My recollection is that none of these instructions are currently implemented anywhere except on 4361, 4381, and 3090 processors.

In exceptional conditions you may need to use **SYSDEBUG(64)**. This will disable all APL abend handling. It would, for example, allow TSO TEST to gain control on program checks within an auxiliary processor. But you need to be aware that program checks may occur normally while APL2 is performing calculations on data. Setting SYSDEBUG(64) is likely to expose apparent "bugs" in the APL2 interpreter or elsewhere which are in fact not errors at all. In Releases 1 and 2 of APL2 it is not possible to modify SYSDEBUG(64) dynamically using *)CHECK SYSTEM*.

The **TRACE** options do not affect the wraparound trace table described earlier. Instead they activate trace output to the terminal or a trace file. Note, however, that SYSDEBUG(1) is a prerequisite to being able to produce any trace output. The individual trace options are described in "APL2 Diagnosis Reference."

TRACE(1) and TRACE(32) are special cases. TRACE(1) output is produced directly on the terminal using TPUT, independent of the session manager or any trace file. Most of the TRACE(32) output (which is quite voluminous) goes to GTF. It can be printed using AMDPRDMP, but only if USR=5A2 is specified on that program's EDIT command.

Here are some general tips in using trace options:

- TRACE(1) is the first thing to try in analyzing auxiliary processor problems.
- TRACE(2), TRACE(4), and TRACE(8) are much smaller if the APL session manager is not being used.
- TRACE(16) can provide a good feel for the overall flow of the system.
- TRACE(64) and TRACE(256) are useful in understanding problems with processor 11 routines.

The APNAMES, EXCLUDE, and LOADLIB quandary

These three options are grouped because, like the preceding set, a keyword may have multiple values associated with it, but unlike the preceding set, they still operate by complete replacement. Thus if you specify a system-wide LOADLIB in the defaults, and a user specifies a private LOADLIB for his session, the system-wide library will not be searched, even though the option supports concatenation in general. Therein lies the quandary, and it applies to all three of these options.

I will not have anything more to say about **LOADLIB** here. See "Accessing Modules in Private Libraries" on page 6.

Because of the quandary, you will not want to use APNAMES and EXCLUDE in the override list. Unless, that is, you want to force all of your users to run with exactly the same set of auxiliary processors. But you will in almost all cases want to include **APNAMES** in the default list. "Auxiliary Processors: ATASKS, RESAPS" on page 15 does discuss an alternative to the APNAMES parameter, but it involves linking the APs with APL2. For MVS/XA this means that the modules are moved from above the 16Meg line to below that line. For all systems it means that excluding the APs later will not recover the load module storage that they use.

The default options provided with the product list AP2X104 and AP2T127 in the APNAMES parameter. You will want to add to that list any locally written APs or APs provided with other products (such as ISPAPAUX, AP 317 for ISPF) that are in general use. If your installation does not have DB/2 installed, you will probably want to remove AP2T127 from the list. This AP requires over 64K and is only used to call DB/2. If only a few of your users need DB/2 you may want to remove it from the default list and provide a special CLIST for those users.

Whatever else you do, you almost certainly do ***not*** want to remove AP2X104 from the default APNAMES list. Without this AP the system cannot do *)COPY*, *)PCOPY*, or *)MCOPY*. (On the other hand, if you run a shop where users are only supposed to *)LOAD* applications, and no one needs *)COPY* except the system programmer, this is an easy way to disable it. Who would ever guess that specifying APNAMES(AP2X104) would reenable it?)

For the most part, EXCLUDE would be used only on the APL2 command to override individual APNAMES in the default list. (This is one way out of the quandary.) APs which are not in the APNAMES list are linked with APL2, so excluding them does not save much storage. But in exceptional cases you might want to provide an AP to *replace* one that is part of the product. It may be a FIXTEST version from Service, or a superset that you have written. If you assign a different entry point name to the replacement (perhaps just at linkedit time) you can EXCLUDE the standard version so that your version can use the standard AP number.

Changing Installation Options

This section discusses the AP2TITOP macro parameters that you can specify in AP2TIOPT. The DEFAULT and OVERIDE parameters were already discussed in "Invocation Options and their Defaults" on page 7 and will not be mentioned further here. Also not discussed here are the USERL macros which appeared in AP2TIOPT up through APL2 1.2.0. Finally, the module includes a table whose entry point is USERT. This table is used by the AP 100 APL USER command. It should be reviewed and corrected to match your system, but its fields are self-explanatory.

SAM Library Paraphernalia

Half of the AP2TITOP parameters deal specifically with tailoring the SAM library support. If you have decided to use VSAM libraries exclusively you need not worry about any of these except PUBQLFR. That parameter also determines the dataset names used for system wide APL session manager profiles.

SAM library support uses three-level dataset names.[18] There are three variations on these names depending on library type (private, public, or project), plus a fourth form that appears only as a catalog entry. See "The Nature of SAM Library Support" on page 1 for an explanation of the library types. The forms are:

Private	*prefix.aplid.wsname*
Public	*pubqlfr.aplidlib.wsname*
Project	*prefix.aplidlib.wsname*
Catalog	*libqlfr.aplidlib.prefix*

The values used in each of these forms are:

prefix	The TSO PROFILE PREFIX of the user who owns the library.
wsname	The simple workspace name.
pubqlfr	The value of the PUBQLFR parameter of AP2TITOP.
libqlfr	The value of the LIBQLFR parameter of AP2TITOP.
aplid	The value of the APLID parameter of AP2TITOP.
aplidlib	An eight character name beginning with *aplid* and ending with a library number. Zeroes are inserted at the beginning of the library number to pad the name to eight characters.

The special catalog entries are used as a project library index. As an example, if AP2TITOP APLID=V,LIBQLFR=APL2, and a user enters *)LOAD 1234 STOCKS*, APL2 will begin by doing a catalog search for *libqlfr.aplidlib* which is APL2.V0001234 in this case. It might find an entry APL2.V0001234.JOHNNY, which would indicate that JOHNNY is the owner of that library. APL2 would then know to read JOHNNY.V0001234.STOCKS to satisfy the *)LOAD* request.

You have probably already realized that APLID should be short, normally one or two characters at the most. Providing a three character APLID would restrict users to five digit library numbers, and longer APLID names would be progressively worse. In most cases the default of "V" is fine, but you might have a dataset naming convention that requires a different leading character in the second level name.

LIBQLFR requires some careful consideration, at least in a RACF shop In order to create a new library, APL2 adds a catalog entry with *libqlfr* as its first qualifier. RACF will prohibit that catalog update unless the user has either CREATE authority for the *libqlfr* group, or ALTER authority for the *libqlfr* generic prefix. If you want your users to be able to define new libraries on the fly, but don't

[18] Unless overridden by the new installation exit support.

want them to be able to clobber the public workspaces shipped with APL2. then you will have to make LIBQLFR different from PUBQLFR. On the other hand, if you change LIBQLFR any time after the system has been installed (even an earlier release), the change will make all project libraries seem to *disappear.*[17]

PUBQLFR is both more sensitive and less sensitive than LIBQLFR. It is more sensitive in a security sense, since RACF ALTER authority provides write access to system-controlled data. (No data is stored under the *libqlfr* prefix, only pointers to data which is controlled under other prefixes.) But PUBQLFR is less sensitive in that ***only*** system-controlled data is located using it. An installation can change PUBQLFR and at the same time recatalog the datasets stored under that prefix, and users will never notice. Perhaps even more important, an installation can migrate from one public library level to another by changing PUBQLFR, while still keeping the previous level online with its original names.

Future levels of the APL2 product will change the default PUBQLFR as a matter of course, while retaining LIBQLFR=APL2.

BLKSIZE, **LIBSER**, and **LIBUNIT** are parameters that APL2 uses when creating new workspace datasets. You can use your own judgement and knowledge of DASD devices in choosing an appropriate value for BLKSIZE. The value distributed with the system is 4240, which is somewhat on the low side for today's devices. Do remember that APL2 will reduce BLKSIZE to a multiple of 80.

As the product is shipped, LIBUNIT and LIBSER are both blank. This is equivalent to using ALLOCATE without a UNIT or VOLUME parameter, and (at least normally) means that APL workspaces are allocated on volumes with a PUBLIC use attribute.[18]

If your installation is one of those that routinely scratches private datasets on public volumes, you will have to do something about the LIBUNIT or LIBSER option. Specifying a LIBSER will, of course, force all APL workspaces to a single volume. This is great if you have only a few APL users creating datasets, and you want to limit the total space they can use. It can even work reasonably well if you have more users but keep HSM busy nibbling away at that volume. But for a shop that writes a lot of APL code, you will need to be able to spread the data out. The way to do this (and hang on to your public volume procedures) is to define an "esoteric" unit type using the UNITNAME macro in MVS SYSGEN. Then you can use that name in the APL2 LIBUNIT option.

LIBKEEP is a simple YES or NO indicating whether empty project libraries are to revert to an unowned status, or whether the system should hang on to the previous owner. We make you decide because we could never make up our minds which approach was more reasonable, so don't expect any sage advice from me. One factor to consider is that the owner of an empty library (LIBKEEP=YES in effect) can get rid of it manually by entering ***)DROP nnnn OWNERSHIP***. Since all workspace names are one to eight characters long we can treat "OWNERSHIP" as a special case.

PBLIBMX is a new option being introduced in Release 3 by popular demand. Its definition is a bit confusing. It is actually the smallest number which is ***not*** a public library number. So the default PBLIBMX=1000 is equivalent to the old rule that public libraries were 1-999. We expect a number of installations to change this value quickly to PBLIBMX=100, or perhaps even less. See "The Nature of SAM Library Support" on page 1 for the reasons.

[17] Don't worry, it's only an Ollie-loss. Change LIBQLFR back and the libraries will magically reappear. Of course if anyone has saved something new in the mean time, ...

[18] See SYS1.PARMLIB(VATLSTxx).

Session Variables: QNLT, QTZDEC, QTZINT

These are initial values during each APL2 session for the APL session variables ⎕*NLT* and ⎕*TZ*. (The initial value for ⎕*PW*, the remaining session variable, is based on terminal type.) The values to use may seem obvious, based on where you live, but there are a couple of surprises lurking here.

Most people don't live in fractional time zones, so you can probably leave QTZDEC=0 as it is. But the shipped value of QTZINT=-13 is a bit unsettling. Unless they have changed things since the last time I looked at a globe, no one lives 13 hours slower than GMT. Did we intentionally choose an invalid value to force you to change it? Not at all. The rule is that values outside the range of -12 to +12 mean APL2 should determine ⎕*TZ* based on the MVS clocks. In general this is better than specifying the offset. Unless you live some place without Daylight Savings Time you would have to reassemble AP2TIOPT twice a year if you gave a specific value.[19]

Well, at least QNLT should be easy enough. Set it to ENGLISH, right? Wrong. Actually, we did set the default to ENGLISH for the first two releases of APL2, more's the pity. But if you look closely at the the definition of ⎕*NLT*, the correct value for English is ⎕*NLT*←''. '' represents the "built-in" language. Any time an invalid language name is specified, the system reverts to the built-in language. So QNLT=ENGLISH is no better and no worse than QNLT=GRINGO.

But it works, doesn't it, so isn't this much ado about nothing? Not exactly, when you go to Release 3. In the first place, there is a *)MORE* message to warn you that you set ⎕*NLT* to an unknown language. If you say QNLT=ENGLISH your users will see things like *CLEAR WS*+ when they invoke APL2. In the second place, you or your users can actually create a language called ENGLISH, and "install" it without changing anything more than a single ALLOCATE statement. You might, for example, want to get all of your messages in lower case. You can do that, given the SAMPLE file shipped with the product, and half an hour or so of your own time.

Product Structure: CSVPID, INAME, OPTUSER

OPTUSER merely lets you have an installation exit routine that is not called AP2TIUSR. You might want this if you had rewritten the module from scratch, and did not want it confused with the sample we ship. No matter what you call it, the exit still has to be linked with the APL2 load module.

CSVPID is not a module name, but a subsystem name. This has to match the value used in the SSID= parameter used to start the Global Shared Variable Processor.

INAME is new for Release 3. It lets you have multiple levels of the system available concurrently, by matching the proper interpreter with the proper system support code.

Auxiliary Processors: ATASKS, RESAPS

Both of these parameters define auxiliary processors that are to be linked with the APL2 product. As noted earlier, doing this forces the APs below the line in an MVS/XA system. APs defined here must be distinguished as VS APL protocol (RESAPS) or APL2 protocol (ATASKS). For APs defined using the APNAMES invocation option that distinction is made by linking the VS APL routines with a compatibility stub, AP2TASVP.

[19] Installations that cheat by redefining GMT twice a year are too contemptible to be considered. Never trust a workspace from such a place.

Installation Exit Routines

There are two very different types of exit routines:

1. The OPTUSER installation option points to an exit routine module (Default AP2TIUSR) which is entered at APL2 invocation and termination, for all system commands, and for AP 100 commands. This exit executes in problem state as a part of a user's TSO session.
2. If the Global Shared Variable Processor is active, it also calls an installation exit during APL2 initialization. That exit is identified in the GSVP startup parameters as the ISECNAME. In some releases of APL2 the GSVP calls another exit whenever the APL2 session signs on to the GSVP. That second exit is named in the GSVP startup parameters as the GSECNAME, but is not discussed here since it is being removed from the product.

The OPTUSER exit has a new more formalized interface, together with full documentation, beginning with Release 3. A compatibility mapping has been provided for customers who have already rewritten the existing sample, or modified it heavily. The sample provided in Release 3 has been completely rewritten.

Invocation and Termination

There are two calls to the OPTUSER exit during invocation. The first occurs as early as possible, before any option parsing has been done. The exit is given the invocation option string (converted to EBCDIC) and can make limited changes to it. It can also set up an installation debugging exit. (Sample code is included, though disabled, to use a product called DBC which was developed by Yale University.)

The second OPTUSER exit occurs after option parsing has been done. The ID number, TERMCODE and terminal type are passed to this exit. It may return a list of SAM libraries that the user is permitted to save into.

Either exit may force termination of the APL2 session, and may return an error message to be displayed or queued.

The GSVP ISECNAME routine is entered after option parsing but before the second OPTUSER exit. It executes in supervisor state, key 0, in the GSVP address space. (Obviously it has to be stored in an authorized library.) The routine is given the TSO userid, the ASCB address for the TSO session address space, and the ID number which the APL2 session is proposing to use. The routine may approve or deny use of the GSVP by the APL2 session, and may change the ID number.

If a *CONTINUE* workspace is loaded automatically during invocation, the OPTUSER exits for *)LOAD* will be called.

The OPTUSER exit is also entered twice during APL2 session termination.[20] At the first exit the terminal support, SVP, and auxiliary processors are still active. The second exit occurs just before APL2 returns to its caller. These exits cannot exercise any control over the APL2 session, except that the first one could display a message. But the exits may want to do logging, and clean up any storage used used by the exit module.

[20] This is in addition to the *)SAVE* exits that will be entered if a *CONTINUE* workspace is saved.

Workspace Command Exits

OPTUSER exits are taken at the beginning and end of processing for all system commands dealing with workspace libraries. The first exit is taken after the appropriate library system has been chosen (SAM or VSAM) but before that subcomponent begins its work. The exit can inspect or modify the library number,[21] workspace name, password, or workspace size. It can also reject the command, specifying a return code which will trigger one of the standard system messages. Finally, like all OPTUSER exits it can also provide its own message text to be displayed or queued.

The second exit can specify a return code that triggers one of the standard system messages, or provide its own message text to be displayed or queued. It can also request processing to be restarted.

For SAM libraries, a number of other parameters can be controlled by the installation exits. These include prefix control, some dataset name control, and reclassification as private, public, or project. For Relase 3 there are special exits taken while the system is generating dataset names and looking up project library owners. These exits can be used to fine tune the standard processing or to replace it completely.

OPTUSER exits are also taken at the beginning and end of processing for *)CLEAR*. The first exit can prevent the operation or specify a size to be used for the *CLEAR WS*. Either exit may provide additional messages.

AP 100 and System Command Exits

OPTUSER exits are taken at the beginning and end of AP 100 command processing. The *)HOST* command also passes its text through AP 100, so the same exits are taken. The first exit can inspect the command about to be issued, and can abort processing of it if it chooses. It can also provide message text to be displayed or queued. The second exit can also provide a message.

Most of the OPTUSER exits previously described are entered for operations that originally arise as system commands. But in addition to those exits, there is a special OPTUSER exit that is taken for all system commands, even ones as innocent as *)VARS*. This exit is entered as soon as the command name has been parsed. If the command name is recognized, its command number is provided.[22] If the command name is not recognized, a command number of ¯1 is assigned and the exit is still called.

The exit can inspect the command and check its command number. On return it indicates that the command is to be executed, rejected, or ignored. And it can queue or display a message.

The "ignore" case is especially significant. The exit can in fact implement installation specific system commands. The system will call the exit indicating that the command is unknown. But the exit will actually execute the command and then return, indicating that it is to be "ignored."

[21] Changing the number (or FREEing a DDname) can be used to switch from a VSAM to SAM library, but not the other direction.

[22] If *⎕NLT≢''*, the command name passed may be in another language, but the command number will still be the same.