APL2 Version 2 Release 1
A Summary

October, 1991

Nancy Wheeler

IBM
APL Products
Santa Teresa Laboratory
San Jose, California, USA

Contents

Abstract

APL2 Version 2 Release 1 was announced on September 11, 1991. It contains significant enhancements over APL2 Version 1, including cooperative processing, access to files as APL2 arrays, improved space management, workstation compatibility items, and many other new features.

This document contains a summary description of the new function found in this version of APL2. Due to its short length, it is not possible to give all the details and complete syntax for each new feature presented. The manuals for APL2 Version 2 Release 1 should be consulted for more complete information. The appendix to this report contains a list of the manuals, with order numbers.

In addition to the author, the following individuals provided sections of this document:

Doug Aiton
Jim Brown
Erik Kane
David Liebtag
Ray Trimble

Cooperative Processing

APL2 Version 2 provides new facilities through which separate APL2 sessions can communicate either with each other or with other non-APL programs across a Transmission Control Protocol/Internet Protocol (TCP/IP) network. The facilities include interfaces at several different levels of both TCP/IP functional access and APL2 syntax.

There are four major facilities within APL2/370's support for cooperative processing:

- Cross-System Shared Variables

  This facility allows a user to share variables with other users on a TCP/IP network using normal APL2 shared variable techniques. It provides APL2's most convenient program-to-program cross network communication path.

- Shared Variable Interpreter Interface

  This interface provides a set of protocols whereby an APL2 interpreter can be controlled through a shared variable rather than through a terminal or file input. It provides a way for a program to control a remote session.

- Remote-Session Manager

  This function manages the protocols of the Shared Variable Interpreter Interface and allows a user to carry on an interactive dialogue with a remote interpreter just as if it was a normal local interpreter.

- TCP/IP Auxiliary Processor

  This processor allows users and applications to make direct requests to TCP/IP. It provides APL2's most flexible program-to-program cross network communication path. The interface can also be used for communication between APL2 and non-APL programs across a network.

Cross-System Shared Variables

APL2 Version 2 permits APL2 users to share variables with each other across systems connected by a TCP/IP network. The users use the $\square SVO$ system function just as they would to share variables with auxiliary processors or users on the same system.

The cross-system shared variable interface requires that both partners have access to TCP/IP to be fully functional.

It is possible to share variables with users on the same system through TCP/IP, although the performance will generally be poorer than using normal shared variables. As with normal shared variables, a session can not share variables with itself across a TCP/IP network.

APL2 Shared Variable Interpreter Interface

APL2's Shared Variable Interpreter Interface provides a set of protocols whereby an APL2 interpreter can be controlled through a shared variable rather than through a terminal or file input. The normal session input and output are replaced with a single shared variable over which communication occurs. This shared variable, and hence the interpreter, can then be managed by a user or program running under another user id.

The shared variable interpreter interface is started by use of the APL2 invocation keyword SMAPL. If the SMAPL parameter is numeric, the interpreter uses it as the processor ID with which it should share a variable. This variable is then used for all input and output to the interpreter. The variable is shared within the interpreter and is not available to, nor will it conflict with, variables and programs being run by the remote interpreter on behalf of the partner.

Input to the interpreter when using this interface is character vectors for terminal input and pairs of scalar integers for control signals. Output from the interpreter is nested arrays whose structure is the same as that produced by the system function ⎕*EC*. Array output is sent as unformatted arrays. Error messages are sent back line-by-line rather than as a ⎕*EM* array (as ⎕*EC* would do.) All other output is also sent as character vectors.

Using the shared variable interface to an interpreter has some impact on the use of system resources. For example, *WS FULL* can happen on any output as the resulting array is prepared for a shared variable assignment. In a directly controlled session, no space would be required.

Once an interpreter is running using the shared variable interface, it operates normally except that its input and output is through the shared variable. It is the responsibility of the interpreter's shared variable partner to manage the variable. The interpreter processes requests until instructed to shutdown either via a shutdown control signal or an *)OFF* or *)CONTINUE* command. When instructed to shutdown, the interpreter sends appropriate shutdown messages and retracts the shared variable.

The Remote-Session Manager

The Remote-Session Manager is an APL2 external function that allows a user to carry on an interactive session with a remote APL2 interpreter running under another user id, perhaps on another system.

RAPL2 establishes and manages a shared variable communication link with a remote APL2 interpreter, using the Shared Variable Interpreter Interface to control the remote interpreter. Once the link is established, the user can enter APL2 expressions and system commands and signal attention just as usual except that all input is passed to the remote interpreter.

⎕ and ⍞ inputs encountered during execution of the user's expressions, or any programs executed by the expressions, will be passed back and prompted for locally by *RAPL2*. Full screen interactions encountered during execution of the user's expressions, that is uses of AP100, AP124, or AP126, will occur at the remote interpreter's location.

When the user signals an interrupt, *RAPL2* will prompt the user for whether:

1. The interrupt should be sent on to the remote interpreter.
2. A local ⎕ prompt loop should be entered. (To exit this loop, signal interrupt again.
3. A shutdown signal should be sent to the remote interpreter (causing a *CONTINUE* workspace to be saved.)

RAPL2 relinquishes control of the terminal when the remote interpreter retracts its shared variable. This typically occurs when the remote interpreter receives an *)OFF* or *)CONTINUE* system command.

```
      rc←time RAPL2 proc_id
```

proc_id is the processor ID of the remote interpreter. This value is used as the left argument to *□SVO* in *RAPL2*'s offer to share a variable with the remote interpreter.

time is the number of seconds *RAPL2* should wait for the remote interpreter to match *RAPL2*'s share offer. If the remote interpreter does not match the offer within *time* seconds *RAPL2* issues an appropriate message and terminates. *time* is optional; the default amount is 30 seconds.

rc is an explicit result indicating whether connection was established, 1, or not, 0.

AP 119 - TCP/IP Processor

The TCP/IP processor, AP 119, is used to pass direct requests to the TCP/IP product. AP119 also provides commands to control how APL2's cross-system shared variable interface uses TCP/IP.

To use AP119, the user shares a variable with the AP and passes vectors of vectors that request various actions. The first element of the value assigned to the variable determines which of two types of commands is being issued:

- Commands to TCP/IP - *'TCPIP'*
- Commands to AP 119 - *'AP'*

The general form of the result is a three element vector:

- An AP 119 return code
- A TCP/IP return code
- Data returned by the command

For example, to issue the TCP/IP command GETHOSTID, you would assign to the shared variable:

```
      SV119←'TCPIP' 'GETHOSTID'
      (AP119_RC TCPIP_RC DATA)←SV119
```

Figure 1 on page 4 summarizes all of the AP 119 commands.

Command	Syntax
TCP/IP Commands	
ACCEPT	'TCPIP' 'ACCEPT' socket
BIND	'TCPIP' 'BIND' socket local_port local_addr
CLOSE	'TCPIP' 'CLOSE' socket
CONNECT	'TCPIP' 'CONNECT' socket remote_port remote_addr
FCNTL	'TCPIP' 'FCNTL' socket command data
GETCLIENTID	'TCPIP' 'GETCLIENTID'
GETHOSTID	'TCPIP' 'GETHOSTID'
GETHOSTNAME	'TCPIP' 'GETHOSTNAME'
GETPEERNAME	'TCPIP' 'GETPEERNAME' socket
GETSOCKNAME	'TCPIP' 'GETSOCKNAME' socket
GETSOCKOPT	'TCPIP' 'GETSOCKOPT' socket level option
GIVESOCKET	'TCPIP' 'GIVESOCKET' socket domain name subtask
LISTEN	'TCPIP' 'LISTEN' socket backlog
READ	'TCPIP' 'READ' socket type
RECV	'TCPIP' 'RECV' socket flags type
RECVFROM	'TCPIP' 'RECVFROM' socket flags type
SELECT	'TCPIP' 'SELECT' num_sockets read_mask write_mask exception_mask
SEND	'TCPIP' 'SEND' socket flags type data
SENDTO	'TCPIP' 'SENDTO' socket flags type data family remote_port remote_addr
SETSOCKOPT	'TCPIP' 'SETSOCKOPT' socket level option option_value
SHUTDOWN	'TCPIP' 'SHUTDOWN' socket how
SOCKET	'TCPIP' 'SOCKET'
TAKESOCKET	'TCPIP' 'TAKESOCKET' domain name subtask socket
WRITE	'TCPIP' 'WRITE' socket type data
AP Commands	
GETLPORT	'AP' 'GETLPORT'
SETLPORT	'AP' 'SETLPORT' processor_id listening_port

Figure 1. Auxiliary Processor 119 Commands

Managing the TCP/IP Interactions

In addition to AP 119 and the changes to the APL2 interpreter, two additional pieces make up the APL2 cooperative processing support: a user directory and a port server.

Identifying Share Partners

The numbers by which cross system shared partners are identified are specified using an APL2 TCP/IP profile file. Each user who wishes to share variables across systems must have this profile file, which defines numbers which will be used to refer to users on other systems with which variables will be shared. A sample profile is provided with APL2 and contains explanations of the file format.

In MVS/TSO, the TCP/IP profile file is a member in a partitioned data set allocated to ddname APL2PROF. Concatenated allocation is supported and can be used to support overriding profile files.

In VM/CMS, the TCP/IP profile file is a CMS file with filetype APL2PROF. The first file found in the normal CMS search order is used.

The APL2 Port Server

APL2 Version 2 includes a program called the *port server* which participates in the establishment of communication links across TCP/IP networks. Each system in the network should have a port server running.

Functions of the Port Server

The port server has three functions:

1. Accept requests to register users on the same system. This function tells the server which port number a given user will be using to accept connections from other users. This port number is arbitrarily assigned to the user by TCP/IP.
2. Accept requests to unregister users. This notifies the server that a given user is no longer accepting communication. This is automatically issued when the user's APL2 session ends.
3. Accept requests from remote users who want to know the port number which has been registered by a user.

When a user first attempts to use TCP/IP (either through cross system sharing or AP 119), TCP/IP assigns the user a TCP/IP port number. When a cross system share offer is made, APL2 contacts the port server at the partner's system to find out the partner's TCP/IP port number.

It is also possible to share variables across systems even if one or both of the systems do not have a port server running. The AP 119 command GETLPORT is used to find out what your own port number is. The command SETLPORT is used to inform the cross-system shared variable facility what your potential partner's port number is.

Running the Port Server

The port server is an external APL2 function which should be run in a started task on TSO or in a disconnected machine on CMS. The normal APL2 or APL2AE product can be used to run the server.

The port server is called *SERVER*. It is accessed as used as shown below.

```
      3 11 ⎕NA 'SERVER'
1
      SERVER
Enter server port number (default 31415):
Enter server password: SECRET
```

The server prompts for the port number it should use. If no response is given, it defaults to using 31415. If a port other than 31415 is given, then users on the same system need to start AP119 specifying the same port number, and users on remote systems will need to specify that port number in their TCP/IP profile files or use the AP 119 SETLPORT command.

The server also prompts for a password which will be required of users attempting to use restricted server commands. If no response is given, no restricted server commands can be used.

Note: Currently, no user server commands are implemented.

The APL2 invocation option RUN can also be used to start the port server. In this case, the INPUT option or APLIN would typically be used to supply the prompt responses.

Files As Variables

Processor 12 is a new Associated Processor which provides access to a variety of types of files by maintaining an image of the file as an array that appears to reside in the active workspace. This is analogous to the behavior of Processor 11 for functions. That processor can create an image of a program (written in any of a variety of languages) as a function which appears to reside in the active workspace. Neither the program (for Processor 11) nor the file (for Processor 12) is actually within the workspace. This has the following implications for Processor 12 files:

- Very large files can be accessed, files which may be many times larger than the active workspace. And yet the access can be done using normal APL constructs such as Compression (e.g. `bool/file`), Each (e.g. `process¨file`), selective assignment (e.g. `(recno⊃file)←value`), and catenation (e.g. `file←file,record`). These are only a few of many possible operations.
- Associations can be retained across `)SAVE` and `)LOAD` but the data is preserved in the file, and may be updated by other programs between uses.

  Note: In particular this should be contrasted with the Processor 11 definition for association with variables in namespaces. The general rule used by Processor 11 is that any time a variable is modified the new version is a private one known only to the workspace which was active at the time of modification.

  It should also be noted that files, even files newly created by Processor 12, have an existence independent of the workspace. Assigning a value to a Processor 12 variable causes (at least conceptually) an immediate and permanent change to the file. This is not affected by later expunging the variable, and is independent of whether the workspace containing it is later saved.

Processor 12 variables are also quite different from variables shared with file auxiliary processors:

- Processor 12 variables contain only the data, and (at least conceptually) all of the data at once. Shared variables contain both data and control information, and only relatively small pieces of the file data at a time.
- Processor 12 variables are really a path between the workspace and the actual file. Shared variables are a path between two programs, one of which in turn is capable of accessing files.
- Processor 12 associations can be retained across `)SAVE` and `)LOAD`. Shared variable associations must be reestablished explicitly.

⎕NA Syntax for Processor 12

The general syntax for name association through Processor 12 is:

```
('type' 'locator' 'format') 12 ⎕NA 'name'
```

name A name to be used within the APL workspace to refer to the file. The particular name used has no significance to Processor 12, and bears no required relationship to the name of the file with which it will be associated. Surrogate names are permitted, but have no functional significance.

type Two or more characters, the first specifying what class of file support is desired, and the others indicating how the file is to be accessed. The file classes supported in APL2 Version 2 Release 1 are APL files (as used by AP 121) and operating system sequential files. Read or write access is supported, along with automatic creation and/or deletion.

locator A character vector indicating where the file is located. For APL files, the locator consists of the library number and filename (as with AP 121 files). For sequential files, the locator is an operating system file name following the conventions of the operating system.

format A character vector which defines the format in which the data is to be viewed by the application. At present this vector must be empty for APL files and non-empty for sequential files.

The syntax of the format descriptor for an external variable is similar to that used by Processor 11. It describes the view of the data as it will be seen by the application, rather than the format of the data as it exists externally.

APL files are always viewed by the APL application as a vector of arbitrary arrays, with each item of the vector representing one object in the file. Each of the items may themselves be of any depth or shape. Sequential files are viewed by the APL application as a vector of arrays in which the sub-arrays are either character vectors or character matrices. Each character vector, or each row of a character matrix, represents one record in the file.

The explicit result of `⎕NA` is `1` if the association was successful or `0` if it failed. When `0` is returned, explanatory messages are usually queued. These may be seen by entering `)MORE` at the first terminal input opportunity or by running with `DEBUG 1`.

Supported Primitive Operations

Regardless of the file system in use, the following primitive operations are defined for external variables supported by Processor 12:

Default display	`file`

Each	`fun¨file`
	`file fun¨var`
	`file1 fun¨file2`

Outer product	`var∘.fun file`
	`file∘.fun var`
	`file1∘.fun file2`

Pick	`i⊃file`
Indexing	`file[i]` `i⌷file`
Pick assignment	`(i⊃file)←array`
Indexed assignment	`file[i]←array` `i⌷file←array`
Catenate	`file1←file1,⊂array`
Shape	`⍴file`

Compress	*i/file*
Take	*i↑file*
Drop	*i↓file*

Note: The functions referred to in Each and Outer product can be arbitrary primitive, defined, or derived functions. Since they are invoked repeatedly with one item of the array at a time, there is no immediate requirement that the entire array truly reside in the workspace. But if the invoked function produces a result, the full accumulated result returned by the derived function will be a normal variable stored in the workspace.

When using the above operations, only the portion of the file needed to perform the function is brought into the workspace. Operations other than those defined here will either attempt to bring the entire file into the workspace or give *DOMAIN ERROR*.

Space Management

The usage of memory by APL2 has been effected in several areas:

- Workspace storage management
- Page release management
- Location of the APL2 product in the VM virtual machine

Internal Memory Management

In APL2 Version 2, a new algorithm is used for management of memory within the workspace.

The primary purpose is to increase performance by reducing the amount of paging and garbage collection that is done. In particular, the larger the workspace, the better the performance improvement. Preliminary tests have shown as much as a 50% reduction in CPU time for an application. The improvement is expected to be greatest for applications manipulating a few large arrays as opposed to many small arrays.

One of the side effects that you will see is a small increase in the size of saved workspaces. In addition, the amount of storage in use while running APL2 will increase slightly. Some increase in workspace size may be necessary to avoid *WS FULL*. The performance benefit should offset this increase in the size of the workspace.

Page Release Performance

Some users of large workspaces on lightly loaded systems have in the past observed a performance problem whose symptom is a large total CPU time (and corresponding elapsed time) with a much smaller virtual CPU time. The problem has been traced to operating system overhead when APL2 releases real pages that are not currently needed. However this same operation has been very helpful on heavily loaded systems.

The new workspace storage management should in most cases address the root cause of this problem. But if you should experience it, you can run with `SYSDEBUG(8)` to completely disable page releases.

Use of Extended Address Space in VM

The APL2 product has been re-organized in Version 2 such that most of the product can run above the 16M line when under VM/XA or VM/ESA. The parts of the product that must run below the 16M line are packaged separately and total less than .25M in size.

Workstation Compatibility

Several new features have been added to APL2 Version 2 to provide increased compatibility with the workstation APL2 products. These include:

- The APL2 Object File Auxiliary Processor, AP 211
- The Fullscreen Auxiliary Processor, AP 124
- Changes in behavior of certain system commands and variables
- Support for new characters

AP 211: APL2 Object File Auxiliary Processor

AP 211 provides a facility for storing APL2 arrays in an object file. The objects may reside in a CMS file or TSO Sequential DASD file with unkeyed records. Fixed-length records are used in both operating systems.

AP 211 uses a single shared variable of any name to control access to a file. Up to 32767 variables may be shared with AP211, giving concurrent access to up to 32767 files. Implementations of AP211 on PC and RS/6000 platforms, however, have more restrictive limits. Portable applications should not use more than 255 concurrent variables.

Syntactically, the mainframe version of AP 211 is compatible with all the current workstation APL2 products. However, it uses a new internal form for its files. Files written in this new form can be identified by the ASCII characters "211B" in the first four bytes of the file, and are not compatible with the files written by the current APL2 for the PC.

The APL2/6000 product uses the new file format. Thus, in addition to source code portability, with APL2/6000 data portability is also possible. Files written by the APL2/6000 version of AP 211 may be uploaded to the mainframe and read directly by the mainframe version of AP 211. Datatype conversions from ASCII to EBCDIC and from IEEE to 370 floating point are done automatically.

Note: At present, APL2/6000 is unable to read the data in a file written by the mainframe AP 211 and downloaded to the RS/6000. It can issue all AP 211 commands against the downloaded file except *GET*.

Figure 2 on page 12 contains a summary of the AP 211 commands. The examples assume that a variable called *SHR211* has been shared with AP 211.

Description	Syntax
Create a file	*SHR211←'CREATE' 'Filename' [Rec_size]* *Return_code←SHR211*
Delete a file	*SHR211←'DROP' 'Filename'* *Return_code←SHR211*
Open a file	*SHR211←'USE' 'Filename' [User_id] ['READ'|'WRITE']* *(Return_code Rec_size)←SHR211*
Close a file	*SHR211←'RELEASE'* *Return_code←SHR211*
Save an object	*SHR211←'SET' 'Name' APL2_Object* *Return_code←SHR211*
Get an object	*SHR211←'GET' 'Name'* *(Return_code APL2_Object)←SHR211*
Delete an object	*SHR211←'ERASE' 'Name'* *Return_code←SHR211*
List the objects	*SHR211←'LIST' 'NAMES'|'ATTS'* *Object_info←SHR211*

Figure 2. AP 211 Operation Codes

AP 124 - Full Screen Management Auxiliary Processor

The Full Screen Management Auxiliary Processor allows you, through an APL application program, to control the screen format of an IBM 3270 Information Display System. In addition, it allows your application to:

- Define a logical screen
- Format the logical screen into screen fields
- Write to the formatted screen
- Read from the formatted screen
- Read program function and program attention keys

The AP 124 provided with APL2 Version 2 is upward-compatible with the VS APL version of AP 124. Some enhancements have been made, such as the addition of support for color. This AP 124 is also compatible with the workstation version of AP 124 wherever possible. However in certain circumstances it is not possible to provide the same abilities on a 3270-type screen that are available on a workstation.

Your APL application requests screen management services by assigning to the control variable a numeric scalar or vector that specifies the requested action. In response, the auxiliary processor issues a return code in the control variable indicating whether or not the requested action was successful.

Figure 3 on page 13 lists and describes the valid operation codes that may be specified to the control variable to request service from the Full Screen Management Auxiliary Processor. The table shows the values that should be specified in both the control and data variables.

CTL VAR	DAT VAR	Description
0 0 n		Delayed clear of screen
1	format	Format the screen
1,fieldnum	format	Reformat selected fields
2,fieldnum	data	Immediate write to screen
3 3 0		Read and wait
4,fieldnum	data	Buffered write to screen
5,fieldnum		Get Data
6,fieldnum	type	Change field type
7,fieldnum	0-255	Change field color or intensity
8 2		Return screen information
9		Read screen format
10		Print screen (not avail.)
11 11 0		Delayed alarm
11 1		Immediate alarm
11 2		Cancel delayed alarm
12	position	Set the cursor
16,fieldnum	attribute	Change input field attr
20		Erase the screen

Figure 3. Screen Management Operation Codes

System Variables and Commands

Domain of QUAD ET

The API.2 V1R3 system restricts values in □*ET* to positive integers between 0 and 32767. That limit is now changed to allow integers between ¯32767 and 32767.

This change also effects external routines in that the values they store in the field *ECVXCET* will now be treated as signed 15-bit integers.

Reference of Format Control

In APL2 Version 1, the result from a reference of ⎕*FC* is extended or truncated to 6 characters, regardless of the length of the vector specified by the user. This behavior is inconsistent with that of other system variables in the system, and with APL2/PC.

⎕*FC* has been modified to always return the user-specified value on reference, if a value has been specified. As before, if the user has not specified a value, the default 6-character value will be returned.

)COPY of System Variables

In previous releases of APL2, the)*COPY* and)*PCOPY* system commands did not copy any system variables from the source workspace.

For compatibility with the PC versions of APL2, and to enhance usability of the mainframe APL2, these commands have been enhanced to copy the following system variables: ⎕*CT*, ⎕*FC*, ⎕*IO*, ⎕*LX*, ⎕*PP*, ⎕*PR*, and ⎕*RL*.

As with other copied objects, only the global value will be copied from the saved workspace, and it will become the global value in the active workspace.

New Character Support

The following new characters can be entered with)*PBS ON* .

Character	*Entered As*	*Name*	⎕*AV*
◊	<_>	*diamond*	x'70'
⊢	[_-	*left tack*	x'76'
⊣	-_]	*right tack*	x'77'

The diamond, left tack and right tack characters have also been added to the symbol sets shipped with APL2.

Note: The additional support for these characters is for entry and display only. They still do not have syntactical meaning in the mainframe version of APL2, and *SYNTAX ERROR* will be reported if they are actually executed.

Other Enhancements

A number of additional enhancements have been made in APL2 Version 2. These include:

- The availability of Processor 10 (the APL2 REXX processor) under MVS.
- A restructured Processor 11, which includes a number of enhancements.
- New tools and utilities for calling programs in languages other than APL2, including C/370 and PL/1.
- A *)EDITOR* extension that allows editors to be APL2 external functions.
- An interface to the QMF (SAA Query) Callable Interface
- External functions to access ESA Data Window Services
- SQL Interface Enhancements
- A directory of commonly used APL2 phrases
- A function to access help information
- New APL2 fonts
- Various smaller usability enhancements

Processor 10 under MVS

A Processor 10 generally compatible with CMS is available under TSO in APL2 Version 2. This processor can be used to call REXX functions and access REXX variables and constants.

To call a REXX function you must first establish an association with dyadic ⎕*NA*. The function thus established is monadic, and its argument is either omitted (i.e. takes no arguments, indicated by ⍬), a character vector, or a vector of character vectors. REXX variables and constants can also be accessed when APL2 is itself invoked via a REXX EXEC.

Some examples, assuming APL2 is invoked from a REXX EXEC:

```
      3 10 ⎕NA 'DELWORD'
1
      DELWORD 'NOW IS THE TIME' '2' '2'
NOW TIME

      2 10 ⎕NA 'RC'
1
      RC
0

      1 10 ⎕NA 'VERSION'
1
      VERSION
REXX370 3.46 31 May 1988
```

Also provided through Processor 10 for TSO is the same set of built-in functions already supplied for CMS:

ΔEXEC to create and call a REXX EXEC
ΔFM to read and write files as matrices
ΔFV to read and write files as vectors of vectors
ΔF to return information about a dataset.

Restructured Processor 11

Processor 11 has been rewritten and restructured to provide new function, better reliability, and extensibility. Included with this new Processor 11 are the following extensions:

- Self-Describing Modules

  In past, any external routines (other than functions that exist in packaged workspaces) had to be described in a NAMES file. With the new Processor 11, external routines can be made self-describing, by prefixing the routine with the necessary NAMES file information.

  Self-describing modules can be accessed directly by specifying member or load library and member in the left argument of *⎕NA*:

  ```
  'MEMBER' 11 ⎕NA 'ROUTINE'
  ```

  or

  ```
  'LIBRARY.MEMBER' 11 ⎕NA 'ROUTINE'
  ```

  in which case the :LINK. and argument tags must appear in the self-describing module.

- Extensions to the :INIT. Tag

  The :INIT. tag in a NAMES file or a self-describing module may now also be specified with a member name or library.member.

- External Niladic Functions Supported

  External functions my now be niladic as well as monadic and dyadic. A new :VALENCE. tag has been added to allow specification of the valence.

- External Operators Supported.

  External operators written in languages other than APL2 are now supported. The :VALENCE. tag is used to specify the number of operands.

  External operators associated with Processor 11 must have :LINK. FUNCTION and be prepared to accept function linkage conventions as described in *APL2 Programming: System Service Reference* On entry, the operands are provided as tokens in ECVXTLF and ECVXTRF. No CDRs are created for the operands. The external operator routine, however, may use the XB service call to build CDR's if the operands are arrays.

- Enhancements for Routines with :LINK. FUNCTION

  :LINK. FUNCTION routines may have environment routines or be environment routines.

- :PARM. Tag

  A new tag, :PARM., may be specified in the NAMES file or in self-describing modules. It is effective only for environment routines which are automatically started. The operand of the :PARM. tag is a quoted character string (double quotes supported in the string). If the environment routine is automat-

ically started the character string, prefixed with a 2 byte length field, is provided to the external environment routine using OS linkage conventions.

This enhancements allows initialization parameters to be passed to automatically started environment routines.

- Additional Information in Parameter List

  The parameter lists to non-APL routines called by Processor 11 have been augmented with prefixes or suffixes with additional information. These enhancements provide a mechanism by which :LINK. OBJECT or :LINK. FUNCTION routines can issue APL service requests, including callback requests. Further, they allow specially designed external functions with access to the formats of APL control blocks to access Processor 11 control blocks or the APL PTH.

- CMS Relocatable Modules Supported.

  In the VM/CMS environment, relocatable load modules are now supported. Such modules can be created with the following CMS commands:

  ```
  LOAD routine (RLDSAVE
  GENMOD module
  ```

  When loading external routines in the VM/CMS environment, Processor 11 first searches for an existing CMS nucleus extension, then a module, then a TEXT file.

- New 'EZ' Service Request

  A new APL service is provided for external routines which are designed to stay active across replacement of the workspace. The 'EZ' service allows such routines to nominate a entry point which will be entered when APL is shut down. Since Processor 11 deletes all active external routines when the workspace is replaced (*)CLEAR,)LOAD,)OFF*), such routines must take special action to ensure that the specified entry point is still available at APL termination. This can be done by loading the necessary code as a CMS nucleus extension, or by issuing a LOAD (SVC 8) request for it. It is also the user's responsibility to delete such routines.

- Groups of Packaged Namespaces

  Packaged namespaces may be placed in a load module with an entry point header and thereby packaged together with other packaged namespaces or external routines. The names of objects which are to be accessed via *□NA* must appear in the routine list describing the collection.

Calls to Other Languages

Two new functions, a utility program, and two new EXECs are provided to help use Processor 11 to call non-APL programs.

- Processor 11 now supports self-describing routines. Routines are made self-describing by link-editing them with a routine description which contains names file information.

  The function *BUILDRD* can be used to build routine descriptions. *BUILDRD* itself can be accessed using Processor 11.

- Processor 11 supports packages of non-APL routines which are listed in a routine list. Such a routine list is required to call programs written in languages such as C/370 which require that the main routine that starts the run-time environment be link-edited with the subroutines. A routine list is also useful for grouping sets of related routines together.

The function *BUILDRL* can be used to construct an object file containing a routine list. *BUILDRL* can also be accessed using Processor 11.

- Processor 11 follows the FORTRAN convention of expecting routines to return scalar results in register 0. C/370 follows a different convention; it returns scalar results in register 1.

  Through judicious use of a routine list, which can be built with *BUILDRL*, it is possible to indicate to Processor 11 that a intermediate routine should be called which will in turn call the C/370 routine which is going to return a scalar result. The intermediate routine can make the call to the C/370 routine, and when it completes, it can copy the scalar result from register 1 to register 0.

  A object file is included in APL2 Version 2 which contains just such an intermediate routine. It is called **AP2XCMAP**.

- Two new execs, **AP2MP11L** and **AP2MP11M**, are provided to aid developers of non-APL routines. **AP2MP11L** link-edits a routine list, compiled non-APL routines, and routine descriptions into a member of a load library. It can be used on either CMS or TSO. **AP2MP11M** generates a module file from a routine list, compiled non-APL routines, and routine descriptions. It can be used on CMS.

Processor 11 Editor

User requests to edit APL objects can be passed to a Processor 11 function. In response to a ∇, APL2 will create an association to and call the Processor 11 function to handle the edit request.

The Processor 11 function is identified with `)EDITOR 2 name` and persists for the entire session unless changed. The Processor 11 function may either reside in an APL2 namespace or be a non-APL program.

The function is executed as if it had been called directly from the user's current namescope. However, it will not be associated in the current namescope so it's association will not cause name conflicts.

Guidelines for Writing a Processor 11 Editor

When the user enters an expression with a leading ∇, APL2 will attempt to establish an association with the function named in the `)EDITOR 2 name` command. APL2 will use `3 11` as the left argument to *⎕NA*. APL2 will then call the function.

The Processor 11 function will be passed a character vector containing the user's ∇ expression. It is the function's responsibility to parse the vector, interpret the user's request, and respond appropriately. APL2 does not ensure that the ∇ expression's syntax is valid. It is entirely the responsibility of the Processor 11 function to interpret the expression.

Note: There is one exception to that rule. If the expression indicates a valid request for display of all or part of a function's or operator's definition using `)EDITOR 1` rules, the request will be fulfilled by APL2; the Processor 11 function will not be called.

If the editor function resides in a namespace, it can use the *EXP* function to reach back into the user's current namescope to reference or specify object definition(s). If the function is a non-APL program, it may use the external services normally supported for Processor 11 functions to access the user's namescope.

QMF (SAA Query) Callable Interface

The SAA Query CPI is implemented in QMF Version 2 Release 4 as the QMF Callable Interface. This new interface to QMF allows a program to start QMF and issue QMF commands without requiring the QMF environment and ISPF to be present. In addition to regular QMF commands, three additional commands are available in this interface which start QMF (START) and allow the program to set and retrieve global QMF variables (GET GLOBAL and SET GLOBAL.)

The SAA Query CPI is supported in APL2 by a new external function interfacing to the QMF Callable Interface. The function is called **DSQCIA** and has the following syntax:

```
(rc handle data)←DSQCIA  handle  cmdstr  [names  values]
```

handle An integer identifying the instance of QMF to which the call refers.

cmdstr A character vector containing the QMF command to be executed.

names A vector of character vectors or scalars which are QMF keywords or variable names.

This parameter is required only for the SET GLOBAL and GET GLOBAL commands. It is optional for the START command.

values A vector of variable values. This can be a vector of character vectors or scalars, or it can be a vector of numbers. It cannot contain a mixture of numeric and character data.

This parameter is required only if the *names* parameter has also been specified.

rc A numeric return code.

data A value whose meaning is dependant on the value of *rc* and *cmdstr*.

- If *rc* is 0 and *cmdstr* contained the string 'GET GLOBAL', *data* will contain the values of the QMF variables requested. For any other QMF command *data* will be null.
- If *rc* is non-zero, *data* will be a four-item nested vector containing error diagnosis fields from the QMF Communications Area DSQCOMM.

ESA Data Window Services

APL2 Version 2 provides a low level direct mapping to the Data Window callable services as supported by other high level languages. The interface provides access to temporary hiperspaces as well as page formatted permanent files which can be viewed through a "window." Services provide for creating, opening, and closing data objects; opening and closing view windows; and committing or undoing changes.

The support is via a set of Processor 11 function routines with interfaces which are very similar to those defined in GC28-1843 *MVS/ESA Callable Services for High Level Languages*. The function routines use names that match those of the MVS/ESA callable services:

CSRIDAC Request or terminate access to a data object
CSRREFR Refresh an object

CSRSAVE Save changes made to a permanent object
CSRSCOT Save object changes in a scroll area
CSRVIEW View an object, or terminate an object view

Because of special MVS requirements, the actual buffer used for the interface is provided outside the workspace by the function routines, and a "mirror" is maintained in the workspace. The application interface is modified slightly to reflect this change.

SQL Interface Enhancements

- Isolation Level Specification

  AP 127 will now accept an additional APNAMES parameter, `ISOL(RR)` or `ISOL(CS)`. Use of this parameter sets the default isolation level for the entire APL2 session. (The AP 127 *ISOL* command may still be used to change the isolation level during program execution.)

- Subsystem Switching (MVS Only)

  A new AP 127 command, *SSID* is added to permit the setting of the DB2 subsystem ID under program control.

- PUT under VM

  The *PUT* SQL statement allows SQL/DS to block INSERT statements, giving a considerable performance improvement to them when inserting multiple rows at one time. It is used in a manner analogous to the *FETCH* command for SELECT statements.

```
                PREP 'NAME' 'INSERT INTO TABLE VALUES(:1,:2,:3,:4)'
0 0 0 0 0

                OPEN 'NAME'
0 0 0 0 0

                PUT 'NAME' DATA
0 0 0 0 0

                PUT 'NAME' DATA
0 0 0 0 0
                .
                .
                .
                CLOSE 'NAME'
0 0 0 0 0
```

- Matrix Input to AP 127

  Previously, value lists supplied as input data to AP 127 were required to be vectors. To process an INSERT, UPDATE, or DELETE statement multiple times, AP 127 had to be called once for each set of value substitutions. The *SQL* function in the *SQL* workspace accepted a matrix of data, but it then created one call to AP 127 for each row of the matrix.

  Now, a matrix of data may be passed directly to AP 127 on the *CALL* command (and in VM, the *PUT* command.) AP 127 will process each row of the matrix as one set of value substitutions. This reduces the number of calls to AP 127 required to process a matrix of data.

- Extended Connect for VM

  The extended form of the SQL CONNECT command, supported by SQL/DS Version 2 and later, will become the default form in APL2. To support the extended form of CONNECT, a third parameter to the AP 127 *CONNECT* verb is accepted, which specifies the database name. This new parameter eliminates the need to use the SQLINIT exec to switch databases, and provides the capability to connect to remote databases in SQL/DS Version 3 Release 3 or later.

  In addition, the *CONNECT* command now returns in the second item of the result a three-item vector containing the server identifier (from the SQLERRP field in the SQLCA control block), ID, and database name for the current connection.

- CONNECT on MVS

  The *CONNECT* command will now also be supported under MVS, when running DB2 Version 2 Release 3 or later.

  The *CONNECT* command in TSO has the following syntax:

```
DAT←'CONNECT' dbname              ⍝ Connect to remote server
(rc (server sqlid user))←DAT

DAT←'CONNECT' 'RESET'             ⍝ Reset to local server
(rc (server sqlid user))←DAT

DAT←'CONNECT'                     ⍝ Obtain connection information
(rc (server sqlid user))←DAT
```

  The *server* identifier is from SQLERRP as for VM/CMS. The *sqlid* and *user* will be obtained from the CURRENT SQLID and CURRENT USER special registers in DB2.

- DECLARE CURSOR with HOLD option

  The ability to declare a cursor such that its position is maintained across COMMIT is now a part of the SQL standard. DB2 Version 2 Release 3 supports this extension.

  To enable AP 127 for support of this ability, a new AP 127 command, *DECLARE*, has been defined.

```
          DECLARE 'NAME' 'HOLD'     ⍝ Declare with HOLD option
0 0 0 0 0

          DECLARE 'NAME'            ⍝ Declare without HOLD
0 0 0 0 0
```

  The *DECLARE* command precedes the *PREP* command in the sequence of AP 127 commands. If it is omitted, the cursor will not be marked for hold.

  When the *COMMIT* command is issued to AP 127 without the *RELEASE* option, cursors will no longer be implicitly be closed or purged by AP 127. If the database system you are connected to supports the HOLD option, cursors marked for HOLD will be maintained in position by the database across COMMITs. If it does not, all cursors will be implicitly closed by the database system, and SQL errors will result if commands are issued using cursor names marked for HOLD.

- SQLCA Control Block Format

  The SAA SQL standard specifies that a character status indicator called SQLSTATE be provided in addition to the numeric SQLCODE field. This new field will be provided by SQL/DS and DB2 in the SQLCA as a 5-element character field following the SQLWARN field, which is now 11 characters long.

AP 127 returns the SQLCA control block as part of the result of the *MSG* command. The format of the object returned to the AP 127 user has been changed to reflect the new structure of the SQLCA. Instead of two 8-byte character fields, the last 16 bytes of the SQLCA are returned as an 11-byte field and a 5-byte field.

In addition, two new AP 127 commands have been added to make it easier for application programs to obtain the contents of the SQLCA. The *SQLCA* command returns the current contents of the entire SQLCA. The *SQLSTATE* command returns just the SQLSTATE field. Unlike the *MSG* command, neither of these commands requires any arguments. New functions corresponding to these commands have also been added to the *SQL* workspace.

- Retrieving Help Text from SQL/DS

  In the SQL/DS product, help text is shipped as tables stored in the database system. A new function, *SQLHELP*, is included in the *SQL* workspace in APL2. This function takes a right argument of a character string keyword. The appropriate SQL tables are searched for the help text associated with the keyword, if any, and the text is returned as a result.

- Symptom Strings under VM

  With SQL/DS Version 2 Release 2, a new formatting program, ARISSMA, was shipped to format SQLCA control blocks into symptom strings suitable for use when calling SQL/DS Service.

  If the SQL/DS connection is successfully established, AP 127 will load this program and return its output as the third item of the result of the AP 127 *MSG* command.

APL2 Phrases

APL2 Phrases enhances the already proven productivity of APL2. With over 675 distinct APL phrases, sorted into 24 general categories, APL2 Phrases presents a fairly thorough list of one-line solutions to common application problems. By having a single repository for APL2 phrases, many of us can take advantage of algorithms that others have developed.

This list is in soft copy and can be accessed directly from your workspace. In addition, code can be dynamically inserted into your own code.

To use this utility simply type `3 11 ⎕NA 'IDIOMS'`, or `)COPY 1 SUPPLIED IDIOMS`, and invoke the *IDIOMS* function.

Once in the function a full screen gives you control over all the idioms. A flag for index origin is supplied and the display routines allow the user to select the origin of preference. A detailed description of each screen is available through a HELP function key.

To enable a quicker selection of idioms, 24 categories were created. These categories are as follows:

Assignment Algorithms
Boolean Selection Algorithms
Boolean Tests General Algorithms
Boolean Tests Numeric Algorithms
Computational Algorithms
Conversion Algorithms
Date and Time Algorithms
External Name Routine Algorithms

Financial Algorithms
Formatting Algorithms
Function Algorithms
Manipulating Characters Algorithms
Manipulating Numbers Algorithms
Numeric Range Algorithms
Numerical Geometry Algorithms
Selecting Positions Algorithms
Sorting Algorithms
Statistics Descriptive Algorithms
Statistics Distribution Algorithms
Structural Algorithms
Text Arrangement Algorithms
Text Change/Select Algorithms
Trigonometry Algorithms
Vectorizing Algorithms

HELP External Function

Frequently, applications need to present text to their users. This text may be more extensive than it is convenient to store in the application workspace. If the application resides in a namespace, maintenance of the text may be cumbersome if it is in the namespace. In addition, this text may need to be provided in several national languages. The *HELP* function allows applications to retrieve keyed text from an application dependent Help File. Help Files may be national language specific.

An APL2 Help File is a normal CMS file or TSO partitioned dataset member containing GML-like tags defining keys which delimit sections of free form text. A Help File may in turn refer to other files containing more text. The *HELP* function can be used to retrieve the list of keys available in a Help File or the text associated with a particular key.

Syntax for retrieving keys

```
keys←applid HELP ''
```

applid Character vector of length 1 to 8. *HELP* uses this as a DDname on TSO or filetype on CMS. If not supplied, a default value of APL2HELP is used. The current value of *⎕NLT* is used a the member name on TSO or the filename on CMS. If a file, or member, in the current national language is not available, the ENP file is used.

keys Character matrix containing available keys in the Help File.

Syntax for retrieving text

```
text←applid HELP key
```

key A character string of length 1 to 65. A *key* may contain imbedded blanks. Trailing blanks are ignored. The application's help file is searched for a record containing a help tag, :HELP., followed by the contents of *key*. All records following the tag up to the next help tag are returned.

text A character matrix containing the text found after the key.

Syntax for retrieving APL2 help

```
HELP key
```

key A character string containing the name of an APL2 public workspace or external function. *HELP* uses the IBM-supplied help file to retrieve the tutorial text for the specified workspace or function.

New APL2 Fonts

A set of All Points Addressable (APA) printer fonts for APL2 are included with APL2 Version 2. These fonts may be used with 3800-3, 3812, 3820, and similar printers. The fonts have a name of 'APL2 DOCUMENT FONT' and are designated as medium weight and medium width. The characters are available in point sizes 6 through 12, 14, 16, 18, 20, and 24. The code page and character set match that defined in Appendix A of *APL2 Programming: Language Reference*, and officially known as code page T1200293.

The following is an example of a DCF control statement to allow use of the new fonts in a SCRIPT document:

```
.df @APL type('APL2 DOCUMENT FONT') codepage T1200293
```

Miscellaneous Usability Enhancements

A number of smaller additions to the product have been made to make life a little easier for the APL2 programmer.

GRAPHPAK Functions for new file types

Although the *GRAPHPAK* workspace provides extensive capabilities for creating screen images, it's support for producing file output suitable for printing or transporting to other environments has been deficient. To ease this problem, three new functions are available in *GRAPHPAK*.

The first two functions, *GSSAVE* and *GSLOAD*, allow applications to build GDDM ADMGDF files. The third function, *PRINT38PP*, allows applications to produce GDDM LIST38PP files suitable for printing by imbedding them in Bookmaster documents.

External Function Directory

Included in the APL2 product are a wide variety of routines which can be accessed using ⎕*NA*. The product also includes NAMES files for each of these routines so they can be easily accessed; this allows users to not concern themselves with the location of the routines.

However, the user is still faced with the burden of specifying the name of the desired routine as an argument to ⎕*NA*. Since there are quite a few external routines in the product, it can be difficult to recall all that are available. Further, unlike defined functions whose purpose can generally be inferred by examining their code, external routines are provided without source code and so their purpose and usage can be unclear.

The *SUPPLIED* workspace contains associations to all the external routines in the APL2 product. They have all been accessed with ⎕*NA* and are ready to be used. Users can)*LOAD* the workspace or)*COPY* functions from it.

In addition, the *SUPPLIED* workspace contains a defined function, *LIST*, which can assist you in learning how to use the external routines.

The *LIST* function lists APL2's external routines and prompts the user to enter a routine name. In response, *LIST* displays tutorial information which describes the purpose, syntax, arguments, and results for the function. A null response to *LIST*'s prompt terminates the function.

DISPLAY as External Function

The *DISPLAY* and *DISPLAYG* functions are now available both as APL2 functions in the *DISPLAY* workspace and as Processor 11 external functions, accessible with ⎕*NA*.

Benefits of using the external versions include elimination of the need to keep a copy of the function in your workspace, and ability to use a surrogate name.

ATTN External Function

The *ATTN* routine allows applications to detect whether the user has signalled an attention.

Frequently applications need to protect themselves from interruption during critical sections of code. This ability is provided by the ⎕*EC* function and using the ignore attention execution attribute during function fixing. However, users also frequently need to signal these applications. The *ATTN* function allows an application to run without being interrupted by attentions and yet detect that the user has signalled.

The *ATTN* function can query whether an attention has been signalled, signal an attention, or remove an attention that has been signalled. It is provided as an external function available through Processor 11 and ⎕*NA*.

PBS External Function

The *PBS* routine allows applications to query and modify the user's current)*PBS* setting.

APL2 needs to be informed whether users' terminals support the seven new APL2 characters or whether printable backspaces are required for their entry. Users can indicate whether they can enter the new characters by use of the)*PBS* system command. Applications frequently also need to determine whether users can enter the APL2 characters. Using the *PBS* function, applications can query, and modify, the user's setting. It is provided as an external function available through Processor 11 and ⎕*NA*.

Host System Query

AP 100 in CMS and TSO has been extended to return a character string containing the name of the host system when it is passed a null character string.

APL NOMSG (TSO Only)

AP 100 under TSO will accept a new built-in command, *APL NOMSG command text*. The indicated *command* will be executed much as if *APL NOMSG* had not been specified, except that:

- Messages normally controlled by the CONTROL NOMSG command within a TSO CLIST are suppressed for the duration of the command. When the command completes message display is restored to its prior state.
- The command cannot be another built-in command.
- The command cannot be an ISPF EXEC. (But the ISPEXEC command can be used to invoke such an EXEC indirectly.)

This feature will aid is suppressing unwanted messages saying "FILE NOT FREED" and the like.

Lower Case Commands and Messages

It is now possible for users to enter system command keywords in any mixture of upper and lower case. Workspace names are also permitted in mixed case, as are operating system commands via *)HOST*.

Note: Names of objects within workspaces must still be entered in the proper case, since those names are case sensitive.

AP 100 for CMS has also been enhanced to translate commands as if they had been entered from the READY prompt at the keyboard. On TSO, the operating system itself will convert lower case letters in commands to upper case, so there is no need to enhance AP 100 for TSO in a similar way, except for the built-in commands, which are now also supported in mixed case.

A new message table is shipped with APL2 Version 2 which contains the product messages in mixed case. The default *⎕NLT* setting will point to this new message table.

DECODE Improvement

The performance of decode has been optimized for the following case:

Left argument all 2's

Right argument Boolean

Right argument vector of length 32 or greater

This change removes one of the performance penalties of migration from VS APL.

AP 121 Restriction Removed

Under CMS, a restriction existed that at most 15 AP 121 files could be open at one time. This restriction has been lifted.

Appendix A. APL2 Version 2 Manuals

Order Number	Title
GH21-1070	APL2 Licensed Program Specifications
GH21-1063	APL2 Application Environment Licensed Program Specifications
GH21-1051	APL2 General Information
SH21-1073	An Introduction to APL2
SH21-1072	APL2 Programming: Guide
SH21-1061	APL2 Programming: Language Reference
SH21-1054	APL2 Systems Service Reference
SH21-1056	APL2 Programming: Using the Supplied Routines
SH21-1057	APL2 Programming: Using Structured Query Language
SH21-1074	GRAPHPAK: User's Guide and Reference
SH21-1058	APL2 Processor Interface Reference
SH21-1059	APL2 Programming: Messages and Codes
SH21-1069	APL2 Migration Guide
SH21-1062	APL2 Installation and Customization under CMS
SH21-1055	APL2 Installation and Customization under TSO
SX26-3999	APL2 Reference Summary
SH21-1071	APL2 Reference Card
LY27-9601	APL2 Diagnosis