APL2

Migration Guide

Version 2 Release 2

SH21-1069-01

APL2

Migration Guide

Version 2 Release 2

SH21-1069-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

Second Edition (March 1994)

This edition replaces and makes obsolete the previous edition, SH21-1069-0. The technical changes for this edition are summarized under “Summary of Changes,” and are indicated by a vertical bar to the left of a change.

This edition applies to Version 2 Release 2 of APL2, 5688-228, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

Programming Interface Information

This migration guide is intended to help programmers code APL2 applications in APL2. This book documents General-Use Programming Interface and Associated Guidance Information provided by APL2.

General-use programming interfaces allow the customer to write programs that obtain the services of APL2.

Trademarks

About This Book

This migration guide is intended to assist you in migrating your IBM* APL2* applications.

Who Should Use This Book

Use this book if you are a current user of VS APL and want to convert applications and defined functions to APL2/370. Also use this book if you are a current user of APL2 Version 1 and want to consider how the changes made to APL2 Version 2 influence workspace migration.

This book describes two migration aids provided with APL2/370: the system command `)MCOPY`, for transferring workspaces, and the TRANSFER workspace, for locating and fixing differences between VS APL and APL2. It also describes the changes to APL2 for Version 2, as well as the workspace migration considerations for those changes.

This book can also be useful when migrating from VS APL to APL2 on workstations. Workstation APL2's supplied workspace MIGRATE contains tools for migration of VS APL workspaces directly to APL2 on workstations. Once migrated, the information in Chapter 3, "APL2 Compared with VS APL" on page 11 can be used to understand the language differences between VS APL and APL2.

APL2 Publications

Figure 1 lists the books in the APL2 library. This table shows the books and how they can help you with specific tasks.

Figure 1 (Page 1 of 2). APL2 Publications

Information	Book	Publication Number
General product	*APL2 Fact Sheet*	GH21-1090
Warranty	*APL2/370 Application Environment Licensed Program Specifications*	GH21-1063
	APL2/370 Licensed Program Specifications	GH21-1070
	APL2 for AIX/6000 Licensed Program Specifications	GC23-3058
	APL2 for Sun Solaris Licensed Program Specifications	GC26-3359
Introductory language material	*APL2 Programming: An Introduction to APL2*	SH21-1073
Common reference material	*APL2 Programming: Language Reference*	SH21-1061
	APL2 Reference Summary	SX26-3999

Figure 1 (Page 2 of 2). APL2 Publications

Information	Book	Publication Number
System interface	*APL2/370 Programming: System Services Reference*	SH21-1056
	APL2/370 Programming: Using the Supplied Routines	SH21-1054
	APL2/370 Programming: Processor Interface Reference	SH21-1058
	APL2 for OS/2: User's Guide	SH21-1091
	APL2 for Sun Solaris: User's Guide	SH21-1092
	APL2 for AIX/6000: User's Guide	SC23-3051
	APL2 GRAPHPAK: User's Guide and Reference	SH21-1074
	APL2 Programming: Using Structured Query Language	SH21-1057
	APL2 Migration Guide	SH21-1069
Mainframe system programming	*APL2/370 Installation and Customization under CMS*	SH21-1062
	APL2/370 Installation and Customization under TSO	SH21-1055
	APL2/370 Messages and Codes	SH21-1059
	APL2/370 Diagnosis Guide	LY27-9601

For the titles and order numbers of other related publications, see the "Bibliography" on page 46.

Conventions Used in This Library

This section discusses the conventions used in this library.

lower	Lowercase italicized words in syntax represent values you must provide.
UPPER	In syntax blocks, uppercase words in an APL character set represent keywords that you must enter exactly as shown.
[ ]	Usually, brackets are used to delimit optional portions of syntax; however, where APL2 function editor commands or fragments of code are shown, brackets are part of the syntax.
[*A* | *B* | *C***]**	A list of options separated by | and enclosed in brackets indicates that you can select one of the listed options. Here, for example, you could specify either *A*, *B*, *C*, or none of the options.
{*A* | *B* | *C***}**	Braces enclose a list of options (separated by |), one of which you must select. Here, for example, you would specify either *A*, *B*, or *C*.
...	An ellipsis indicates that the preceding syntactic item can be repeated.
{}...	An ellipsis following syntax that is enclosed in braces indicates that the enclosed syntactic item can be repeated.

The term *workstation* refers to all platforms where APL2 is implemented except those based on System/370* and System/390* architecture.

Throughout this book, the following product names apply:

Product Name	Platform
APL2/2	OS/2*
APL2 for Sun Solaris	Sun** Solaris**
APL2/6000*	AIX/6000*
APL2/370	MVS or VM
APL2/PC	DOS

Summary of Changes

Product

APL2/370, Version 2 Release 2

Date of Publication: March 1994

Form of Publication: Revision, SH21-1069-01

Document Changes

- Added information for migration from Version 2 Release 1 to Version 2 Release 2
- Added diamond information
- Added information on workstation APL2

Chapter 1. Overview of Migration

Migration is the process of transferring workspaces from one version of APL to another while ensuring that applications in the workspaces are processed correctly under the new version. You can transfer workspaces directly from VS APL to APL2, or from one APL2 version to another.

To migrate workspaces from VS APL, you need to:

- Transfer workspaces from VS APL to APL2.
- Make changes to defined functions, and test and debug them.
- Check data files, alternate input (stacked data), and user-written auxiliary processors.
- Test applications as a whole and put them into production.

APL2 provides a workspace, system functions, and system commands to aid in the migration procedure. APL2's extended debugging capabilities can also help you convert applications to APL2.

To migrate workspaces from the previous version of APL2, you need to load using the `)LOAD` command, or copy using the `)COPY` command, the workspace into APL2 Version 2 and then save it using the `)SAVE` command. For information on the changes to APL2 in Version 2 and their effect on existing workspaces, see Chapter 5, "Migration within APL2" on page 41.

Planning for Migration from VS APL

APL2 and VS APL can run concurrently at your installation, or APL2 can completely replace VS APL. In either case, you must make decisions concerning:

- Workspaces
- Files
- Auxiliary processors

Answering the following questions may help you make those decisions.

If you continue VS APL service, which workspaces remain VS APL workspaces, and which are migrated to APL2?

- *How long will you use the present applications?* If a current VS APL application is to be replaced in the future, you may decide to continue running it under VS APL.
- *Will the migration involve a workspace that is dependent on other workspaces?* Assess the impact of migrating all interdependent workspaces.
- *Must functions that use files be changed?* Determine the translation changes needed for functions in the workspace that read from or write to a file. Also, decide whether to convert the file to a different encoding. (Files created under VS APL are usually compatible with APL2. However, files created under APL2 may be incompatible with VS APL.)
- *Which user-written auxiliary processors must be adapted?* Determine which workspaces are dependent on user-written auxiliary processors.

- *What are the space requirements for maintaining both VS APL and APL2?* Calculate the amount of space required by the code, workspaces, files, and user-written auxiliary processors for both the VS APL and the APL2 systems.

If you do not continue VS APL service, what will you do with current VS APL workspaces and any files used by those workspaces?

- *Which workspaces are migrated and enhanced?*
- *Which workspaces are migrated and not enhanced?*
- *Which workspaces, if any, are dropped?*
- *Which files are converted to a new encoding?*
- *Which files are already compatible?*
- *Which files are dropped or replaced?*

Regardless of whether you continue VS APL service, you should plan for:

- Adequate APL2 training.
- Procedures and temporary space for maintaining a backup copy of all code, workspaces, files, and user-written auxiliary processors being converted from VS APL to APL2.

  Maintain backup copies until you are sure that your applications are working properly under APL2.

Preparing for Migration from VS APL

Read this guide to learn about the overall process of migrating to APL2.

When you are ready to migrate, follow the procedure in Chapter 2, "Transferring Workspaces from VS APL" to transfer workspaces from VS APL to APL2.

To familiarize yourself with the differences between VS APL and APL2, see Chapter 3, "APL2 Compared with VS APL" on page 11.

After you transfer your workspaces, refer to Chapter 4, "Testing and Debugging" on page 30 to modify functions in the transferred workspaces. It includes what to look for while you are debugging functions, and while you are checking data files, data on the alternate input stack, and user-written auxiliary processors.

Chapter 2. Transferring Workspaces from VS APL

This chapter discusses the transfer of workspaces from VS APL to APL2/370. For information about transferring workspaces from VS APL to workstation APL2, see the appropriate user's guide.

Preparation for Transferring Workspaces

Before transferring any workspaces do the following:

1. Make sure that APL2 has been installed and verified.
2. If you are migrating to APL2 under CMS, check for a LIBTAB APL2 file. If it does not exist, you can copy and modify the APLIBTAB file that you are using for VS APL. Do the following:
   a. Copy the VS APL APLIBTAB:

      ```
      COPY APLIBTAB APLIBTAB xxx LIBTAB APL2 yyy
      ```

      where:

      xxx is the disk where the APLIBTAB file currently resides.

      yyy is the same disk as that containing the module used to call APL2.

   b. For each public library, insert an @ before the filemode. If no filemode is shown, enter @ Y.
   c. Change each project library designation to public.

      Note: For more information, see *APL2/370 Programming: System Services Reference*.
3. Determine which workspaces to transfer. To find the names of the workspaces, start VS APL and issue the `)LIB` or `)LIB` *n* command, where *n* is the number of the library where the workspace resides.
4. Modify functions to adjust for `⎕AV` dependencies, as described in "Adjusting for `⎕AV` Dependencies" on page 4.
5. If VS APL resides on a system other than the host system for APL2, transfer your VS APL workspaces and files from that system to the host system—CMS or TSO. VS APL does not need to exist on the host system if your workspaces follow the VS APL naming conventions and format for the host system.

Choosing an Alphabet

VS APL used uppercase and underbarred uppercase letters in names. It allowed lowercase letters in comments, constants, and variables. APL2 uses uppercase and lowercase letters in names. The System/370 and System/390 implementation of APL2 allows underbarred letters in comments, constants, and variables. ASCII-based implementations of APL2 (such as APL2/PC, APL2/2, APL2 for Sun Solaris, and APL2/6000) do not define underbarred letters at all.

As a migration aid, the APL2/370 product provides some toleration of underbarred letters in names. The degree of toleration is controlled by a CASE option:

CASE(2) Underbarred letters in names result in *SYNTAX ERROR* whenever the names are encountered while processing an APL statement. No conversion is performed when the statement is accepted into an APL2 workspace.

CASE(1) Statements containing underbarred letters in names are accepted, but the letters are converted to lowercase before they are stored in the APL2 workspace. This includes labels, arguments, variable names, and names of defined functions. The conversion is applied to statements from sources such as typed input, APL editors, *)IN*, *)COPY*, and of immediate importance to this discussion, *)MCOPY*. It does not include constants (quoted character data in functions), comments, or the values within character arrays. If constants or arrays are later used as APL statements, for example by applying ⍎, *⎕EA*, or *⎕FX* to them, underbarred letters in names within the data being evaluated are converted at that time.

CASE(0) In addition to the processing done for CASE(1), lowercase letters in names are converted to underbarred letters whenever the names are displayed or returned by a function. This option gives the appearance that the system is behaving like VS APL, although in fact lowercase letters are being used internally.

CASE(2) is not recommended when converting workspaces from VS APL, because manual modification of all underbarred letters would be required. Either CASE(1) or CASE(0) provide semi-automatic translation of underbarred letters to lowercase. IBM recommends CASE(1) because it simplifies later migration to ASCII-based platforms, or use of certain ASCII-based terminal emulators. However some installations may choose CASE(0) so that the VS APL to APL2 migration appears as transparent as possible to their APL users.

Whatever CASE is chosen, it is important to note that the case of a workspace can never be changed except by copying it into another workspace having a different CASE. CASE is an APL2 invocation option, which can be changed dynamically by using the *OPTION* external function or the command:

```
      )CHECK SYSTEM CASE(n)
```

However that option does ***not*** affect the active workspace. It should be thought of instead as ***only*** an implicit parameter to the *)CLEAR* command. Migration is normally accomplished by using the sequence of APL commands *)CLEAR*, *)MCOPY*, *)SAVE*. The CASE in effect for the session at the time the *)CLEAR* is done in that sequence determines the case of the saved workspace.

Adjusting for ⎕AV Dependencies

You can translate character data from VS APL internal encoding (Z-codes) to APL2/370 internal encoding (EBCDIC).

The system cannot determine whether such translation is appropriate for all character data. If you want to preserve the ⎕AV position of data, use the following procedure to transfer data from VS APL to APL2 *without translation*:

Under VS APL:

1. Load the workspace that contains the character data.
2. Convert each array of character data to an array of indexes into ⎕AV. You can use an expression such as the following, where *CX* represents the array of character data and *IX* represents the array of indexes:

   ```
   IX←⎕AV⍳CX
   ```

3. Save the workspace.

Under APL2:

1. Use the)*MCOPY* command (as described in "Transfer Procedure") to transfer the VS APL workspace into the APL2 active workspace.
2. To convert the data back to character format, use the same index origin that you used under VS APL. For example:

   ```
   CX ← ⎕AV [IX]
   ```

   Note: If no object names are specified)*MCOPY* copies the index origin from the VS APL workspace.

3. Erase *IX* and then save the workspace.

You can also use the *TRANSFER* workspace distributed with APL2. This contains two functions to help you migrate character data whose application requires the same ⎕AV positions:

- *CHARIND*, typed or transferred into the VS APL workspace, modifies each variable named in its argument to be a vector of ⎕AV indexes. The variables named in the argument should not be numeric variables. (*DESCRIBE* in the *TRANSFER* workspace explains how to use the session manager to put this function in your VS APL workspace without typing it.)
- *INDCHAR*, used in the transferred workspace under APL2, rebuilds the variables encoded with *CHARIND*.

  Warning: Running *INDCHAR* against a variable *not* encoded by *CHARIND* converts the variable improperly.

Transfer Procedure

To transfer workspaces from VS APL to APL2, start APL2. Next, follow the steps described in Figure 2. During the entire migration process, maintain your original VS APL workspaces so that, if necessary, you can return to any previous stage of the migration process.

Figure 2 (Page 1 of 2). Steps for Transferring a Workspace from VS APL to APL2

Step	Your Entry under APL2	Explanation and Comments
1	)*CLEAR*	Clear the active workspace (sets CASE attribute).

Figure 2 (Page 2 of 2). Steps for Transferring a Workspace from VS APL to APL2

Step	Your Entry under APL2	Explanation and Comments
2	)*MCOPY* *[libno] wsname*	Specify the name (and library, if necessary) of the VS APL workspace that you want to migrate. The)*MCOPY* command copies the contents of that workspace into the active workspace. If you want to copy only selected objects from the VS APL workspace, refer to "Transferring Selected Objects" on page 9.
3	)*SAVE* *[libno] wsname*	Specify the name of the APL2 workspace into which you want to save the contents of the active workspace. This name can be the same as the VS APL workspace name. (The VS APL workspace remains intact.) When a workspace is saved, its time stamp changes. The time stamps of functions in the workspace do not change.

Repeat these steps for each VS APL workspace that you want to migrate.

The *TRANSFER* workspace distributed with APL2 contains the *MASSMCOPY*_ function to simplify the procedure when you want to transfer many workspaces. *MASSMCOPY*_ transfers multiple workspaces from VS APL to APL2.

The)*MCOPY* Command

To transfer a workspace from VS APL to APL2/370, copy a workspace from your VS APL library into the APL2 active workspace. Then, save the workspace in your APL2 library, as shown in Figure 3.

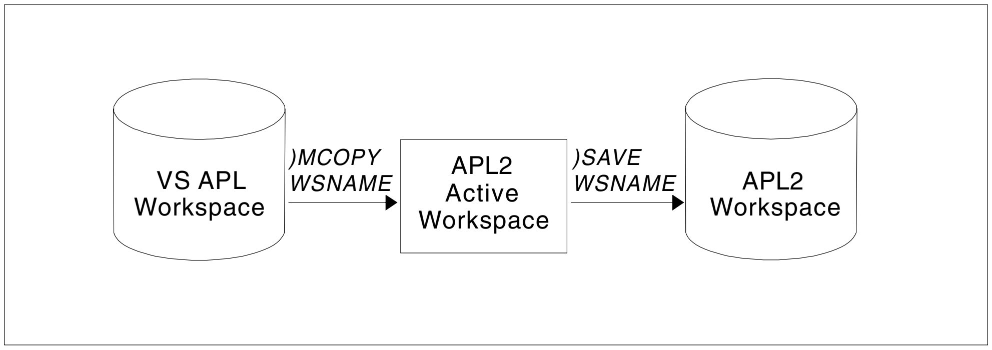

Figure 3. Transferring a Workspace from VS APL to APL2

The)*MCOPY* command copies the contents of a workspace from your VS APL library into the APL2 active workspace.

System Requirements for Using)*MCOPY*: To use the)*MCOPY* command, your workspaces must reside as VS APL workspaces on the same CMS or MVS system in which you use the)*MCOPY* command.

If your workspaces reside on another system, you must first transfer your workspaces to the host system. Information about how to transfer workspaces is available in *VS APL for CMS: Installation Reference Manual*, *VS APL for TSO: Installation Reference Manual*, *VS APL for CICS/VS: Installation Reference Manual*, and *VS APL for VSPC: Installation Reference Manual*.

Syntax of the)*MCOPY* **Command:** Figure 4 shows the sytax of the)*MCOPY* command.

```
)MCOPY [libno] wsname [:[password]] [names]
```

Figure 4. Syntax of the)MCOPY Command

To copy a VS APL workspace not in your private library, you must specify the library number before the workspace name when you issue the)*MCOPY* command:

```
      )MCOPY 6 MESSAGES
```

To copy a VS APL workspace that is password protected, you must specify the workspace name, colon, and password when you issue the)*MCOPY* command:

```
      )MCOPY PASSAGES:HUSH
```

Effects of the)*MCOPY* **Command:** When you issue the)*MCOPY* command, it has the following effects:

- The specified objects are copied from the VS APL workspace into the APL2 active workspace. If no objects are specified by name, all objects in the VS APL workspace are copied.
- Any objects copied from the VS APL workspace replace objects of the same name in the active workspace.
- If no object names are specified, the system variables ⎕*LX*, ⎕*IO*, ⎕*RL*, ⎕*CT*, and ⎕*PP* are also copied from the VS APL workspace into the active workspace. Otherwise, you must specify the names of the desired system variables with other object names when you issue the)*MCOPY* command.

  Note: Only the five system variables ⎕*LX*, ⎕*IO*, ⎕*RL*, ⎕*CT*, and ⎕*PP* can be copied. An attempt to copy other system variables (for example, ⎕*PW*) results in a *NOT FOUND* message.
- Locked functions in a VS APL workspace are copied and remain locked. Under APL2, their execution properties are 1 1 1 1, as shown below for the locked function named *SEARCH*:

  ```
        3 ⎕AT 'SEARCH'
  1 1 1 1
  ```

  For information on function locking and attributes under APL2, see *APL2 Programming: Language Reference*.
- The time stamps of functions in the migrated workspace do not change. Older functions that do not have time stamps take the time stamp of the VS APL workspace.
- Indexed numeric vector constants are enclosed in parentheses when they are copied. For example, the expression 4 5 6[3] under VS APL becomes (4 5 6)[3] under APL2.

  Note:)*MCOPY* does *not* convert these expressions if they are contained in character data or constants, because such data might later cause an error if it is executed (using ⍎), transferred into a function (using ⎕*FX*), or stacked using AP 101.

- Any names in the local list in a VS APL defined function that match the names of the result or either argument of the function are deleted. See Figure 22 on page 21 for more information on the evaluation of expressions under VS APL.
- Each group in the VS APL workspace is represented in the active workspace as a simple character matrix, as shown in Figure 5. The matrix has the same name as the group name. Each row in the matrix is composed of the name of an object in the group. The matrix also contains its own name.

Group Created under VS APL:

```
   )GROUP MYGROUP SEARCH DATA1 DATA2
```

Matrix Created By)*MCOPY* **under APL2:**

```
      MYGROUP
MYGROUP
SEARCH
DATA1
DATA2

      ⍴ MYGROUP
4 7
```

Figure 5. Migrating a Group of Objects from VS APL to APL2

Workspace Location for the)*MCOPY* **Command:** When locating a workspace, the)*MCOPY* command uses the VS APL library definition rather than the APL2 one.

Under CMS The)*MCOPY* command uses the file APLIBTAB APLIBTAB—not the file LIBTAB APL2.

If no APLIBTAB APLIBTAB file is found, a dummy one is created. This dummy file allows you to access only the library with the number ⎕*AI*[1].

Under TSO The)*MCOPY* command can access the same workspaces available under VS APL, as long as the TSO user ID and PROFILE PREFIX are unchanged.

Installation Options and the)*MCOPY* **Command:** If your VS APL installation has changed parameters in the options module that you want carried forward by)*MCOPY* to APL2, then:

Under CMS If the PRIVWS parameter of the APLSCOPT module was changed, the modified APLSCOPT module should have been link-edited with APL2. See *APL2/370 Installation and Customization under CMS*.

Under TSO APLYUOPT should be available as a separate load module if any of the following parameters were changed: APLID (from @W), PUBQLFR (from @PL), or LIBQLFR (from @). See *APL2/370 Installation and Customization under TSO*.

Transferring Selected Objects

Using the `)MCOPY` command, you can copy selected objects from a VS APL workspace into the APL2 active workspace. You can copy:

- **Individual objects.** Specify the object names when you issue the `)MCOPY` command:

  ```
          )MCOPY MISSIONS SEARCH DATA1 DATA ⎕LX ⎕IO
  ```

- **Groups of Objects.** Specify the group name in parentheses when you issue the `)MCOPY` command:

  ```
          )MCOPY MISSIONS (MYGROUP)
  ```

  If parentheses are not used, the group of objects is not copied; only the object *MYGROUP* is copied.

- **Both individual objects and groups of objects.** Specify the names of groups, in parentheses, with the names of individual objects:

  ```
          )MCOPY MISSIONS (MYGROUP) DATA (STGP) ⎕LX ⎕IO
  ```

Error Messages When Using)*MCOPY*

Figure 6 lists and describes possible `)MCOPY` error messages.

Figure 6. Error Messages

Message	Meaning
IMPROPER LIBRARY REFERENCE	The specified library is inaccessible or does not exist.
LIBRARY I/O ERROR	An internal error is preventing the successful copying of the workspace.
LIBRARY NOT AVAILABLE	Another user temporarily has control of a shared library. This prevents making a successful copy of the workspace at this time.
NOT COPIED:	The listed objects cannot be copied because insufficient space is available for copying the objects. The names of objects not copied are listed.
NOT FOUND:	The named objects do not exist or cannot be copied from the specified workspace.
SYSTEM LIMIT	This message can indicate any of the following: • Your virtual machine size or region size is not large enough. Increase the size. • Your workspace size is not large enough to load the specified workspace. Change that size by using the WSSIZE option when APL2 is started. • The freespace size is not large enough. Change that size by using the FREESIZE option when APL2 is started. • In CMS, insufficient space exists on the disk where your private library is located. Free up space on that disk or change the LIBTAB APL2 file to point to another disk. In TSO, the library file definition or files defined by ddnames CPYSWAP or CPYSPILL are either not allocated or not large enough. Ensure that they have sufficient space.
WS FULL	Sufficient space is not available for copying all or some of the objects specified with the `)MCOPY` command. When possible, `)MCOPY` lists the names of objects not copied.
WS LOCKED	The password specified with the `)MCOPY` command is incorrect, or a necessary password is missing from the command.
WS NOT FOUND	The specified VS APL workspace does not exist in the specified VS APL library.

For more information on correcting these errors, see *APL2/370 Messages and Codes*.

Chapter 3. APL2 Compared with VS APL

This chapter lists features that are new, extended, or no longer supported under APL2.

The extended features are organized into those that are compatible and those that are incompatible. A feature is *compatible* if it allows an application that ran under VS APL to produce the expected results under APL2. Many changes in the editors and other facilities are compatible because they do not affect applications being migrated into APL2. However, they can cause errors if used improperly under APL2.

The material in each section is separated into features pertaining to the APL2 language and features pertaining to system services.

New APL2 Features

This section lists new language and system features.

New Language Features

For information on features listed in Figure 7, see *APL2 Programming: Language Reference*.

Figure 7 (Page 1 of 3). New Language Features under APL2

Category	New Feature
Data Types and Structures Allowed	**Complex numbers:** Numbers can have real and imaginary parts. **Mixed data:** Both characters and numbers can be in the same array. **Nested arrays:** An array can contain another array.
Primitive Functions (Boolean and Relational)	Match (*L*≡*R*)
Primitive Functions (Structural)	Depth (≡*R*) Enclose (⊂*R*) Enclose with Axis (⊂[*X*]*R*) Disclose (⊃*R*) Disclose with Axis (⊃[*X*]*R*) Ravel with Axis (,[*X*]*R*) Enlist (∊*R*)
Primitive Functions (Selection)	First (↑*R*) Pick (*L*⊃*R*) Take with Axis (*L*↑[*X*]*R*) Drop with Axis (*L*↓[*X*]*R*) Without (*L*~*R*) Index (*L*⌷*R*) Index with Axis (*L*⌷[*X*]*R*)
Primitive Functions (Position)	Find (*L*⍷*R*) Dyadic Grade Up (*L*⍋*R*) Dyadic Grade Down (*L*⍒*R*) Partition (*L*⊂*R*) Partition with Axis (*L*⊂[*X*]*R*)
Primitive Operators	Each (*LO*¨)

Figure 7 (Page 2 of 3). New Language Features under APL2

Category	New Feature
Defined Operators	In addition to functions, you can define operators. The operand(s) of defined operators can be arrays or they can be primitive, defined, or derived functions.
Derived Functions	Derived functions are new functions resulting from operators. Operands of operators can be primitive functions, defined functions, well-formed derived functions, or arrays. Thus, many derived functions are possible. Two new derived functions are: • Replicate (*LO/R*) • N-wise Reduce (*L LO/ R*)
Separators	Diamond (◊)
System Functions and Variables (Event Handling and Debugging)	Event Message ⎕*EM* Event Simulation ⎕*ES* Event Type ⎕*ET* Execute (Process) Alternate ⎕*EA* Execute (Process) Controlled ⎕*EC* Left Argument ⎕*L* Right Argument ⎕*R*
System Functions and Variables (Sharing)	Shared Variable Event ⎕*SVE* Shared Variable State ⎕*SVS*
National Language Translation for System Commands and Messages (⎕*NLT*)	System commands can be entered and system messages can be displayed in English or in several other national languages. See *APL2 Programming: Language Reference* for a list of the messages and commands in the supported national languages.
Other System Functions and Variables	Atomic Function ⎕*AF* Attributes ⎕*AT* Format Control Characters ⎕*FC* Name Association ⎕*NA* Prompt Replacement ⎕*PR* Time Zone ⎕*TZ* Transfer Form ⎕*TF*
System Commands (For Storing and Retrieving)	)*IN* Read objects from a transfer file.)*MCOPY* Copy a VS APL workspace.)*OUT* Write objects to a transfer file.)*PIN* Read objects from a transfer file, protecting like-named objects
System Commands (For the Active Workspace)	)*CS* Sets APL2 to insert parentheses around indexed numeric constants and to generate errors, when vector notation is used for expressions other than simple numeric vector constants.)*EDITOR* Identify or specify the function editor.)*NMS* List objects in the active workspace.)*OPS* List defined operators in the active workspace.)*PBS* Turn the printable backspace character on or off, or report its setting.)*RESET* Reset the state indicator.)*SIC* Reset the state indicator.)*SIS* Query the state indicator, showing statements.
System Commands (For System Services and Information)	)*HOST* Submit command to the host system.)*MORE* Display additional system messages (new under CMS).

Figure 7 (Page 3 of 3). New Language Features under APL2

Category	**New Feature**
System Messages	*AXIS ERROR* *SYSTEM LIMIT* *VALENCE ERROR*

New System Features

For more information on features listed in Figure 8, see *APL2/370 Programming: System Services Reference*. For more information on installation, see *APL2/370 Installation and Customization under CMS* or *APL2/370 Installation and Customization under TSO*.

Figure 8 (Page 1 of 2). New System Features under APL2

Category	**New Feature**	
Invocation options	CASE	Specify alphabet convention.
	DATEFORM	Change the time and date stamp format.
	DSOPEN	Override GDDM* default terminal characteristics.
	EXCLUDE	Exclude the listed auxiliary processors from those normally available upon invocation.
	ID	Specify your numeric user ID—the default library number. This replaces the positional ID parameter previously available in VS APL under CMS.
	INPUT	Specify lines of APL2 input to be processed upon invocation (new under CMS).
	SVMAX	Specify the maximum number of shared variables that can be concurrently handled by the system.
	SYSDEBUG	Specify system programming debug settings.
	TRACE	Specify system programming trace settings.
	XA	Specify location of working storage.
	RUN	Specify name of an external function to be processed upon invocation.
	DBCS	Use double-byte character set.
Session manager commands	*FIND* allows searching for a character string in the session log.	
Auxiliary processors distributed with APL2	AP 102 Main Storage Access Processor (new under CMS)	
	AP 119 TCP/IP Socket Interface Auxiliary Processor	
	AP 127 DB2* or SQL/DS* Auxiliary Processor	
	AP 211 APL2 Object File Auxiliary Processor	
Facilities for user-written auxiliary processors	A return code from the shared variable processor allows auxiliary processors written according to VS APL conventions to detect when they are given data that cannot be represented in VS APL format. If you wish to modify them to handle APL2 data format, one new executable macro and two new mapping macros are provided for this purpose. A set of services is also provided for writing auxiliary processors according to APL2 conventions.	

Figure 8 (Page 2 of 2). New System Features under APL2

Category	**New Feature**
Associated processors	A new type of processor is supported in APL2. Associated processors allow names in a workspace to be associated with objects outside the workspace using the new system function ⎕*NA*. Once associated, these names can be used with normal APL2 syntax. Three associated processors are provided: • Processor 10 supports association of APL2 names with REXX execs and variables. • Processor 11 supports association of names with routines written in languages other than APL2. Processor 11 also supports association of names with APL2 objects residing in other workspaces. • Processor 12 supports associating names with either sequential operating system files or auxiliary processor 121 APL object files.
Editors	The line editor,)*EDITOR* 1, allows selective display and selective deletion of lines through new edit commands. The full-screen editor,)*EDITOR* 2, has the features of the line editor, additional editing commands, and function key settings to make editing easier. It also allows editing of character matrixes and split-screen editing (segmenting the screen to edit more than one function at a time). Functions and APL2 expressions can also be processed during a full-screen editing session. The full-screen editor requires the Graphical Data Display Manager (GDDM) Licensed Program. The)*EDITOR* 2 *name* command can be used to specify the name of an editor written either in APL2 or another language. The editor must be accessible through associated processor 11. When the user invokes the editor, through the use of ∇, APL2 will create an association to the editor and pass it the ∇ expression. The editor can then extract the object to be edited from the workspace, display it for edit, and reestablish the new definition. System editors such as XEDIT and ISPF can also be accessed from APL2. The command)*EDITOR* *xxxx* specifies the name of the CMS command or TSO CLIST that is used for editing. APL2 writes the function or character matrix to be edited into a CMS file or TSO data set and invokes the command or CLIST with the name of the file or data set as its argument. When editing ends, the object is brought back into the APL2 workspace.
Performance Analysis	An external function, *TIME*, is accessible through processor 11 and allows users to gather performance statistics on their applications. *TIME* can return line-by-line statistics on an application indicating number of times the line was processed and the relative and absolute amount of CPU time consumed in the line. This facility can be used to isolate bottlenecks in applications so that they can be recoded using more efficient techniques either in APL2 or other languages.

Extended-Compatible Features

Figure 9 through Figure 21 on page 20 compare features of VS APL with similar features of APL2. The extended features listed in this section are compatible—they allow an application that runs under VS APL to produce the expected results under APL2.

Extended-Compatible Language Features

For more information on features listed in Figure 9 through Figure 18 on page 18, see *APL2 Programming: Language Reference*.

Figure 9. VS APL Compared with APL2—Data Representation (Compatible)

Data Representation under VS APL	Data Representation under APL2
An array can contain only character or only numeric data. Each item in an array must be a single number or a single character.	An array can contain a mixture of character and numeric data. It can contain an item that is not a single number or character (for example, vector or matrix).

A vector constant can be entered as adjacent numbers separated by blanks or as a string of characters enclosed in single quotation marks. A vector constant cannot be a mixture of these representations.
`R←9 13 48 27`
`L←'SHOE'`
	A vector can be entered as a list of arrays. These arrays can be represented by numbers, one or more characters in single quotation marks, an object name representing the value of an array, or an APL2 expression producing the value of an array. Array representations can be separated by either blanks or parentheses. A vector can be a mixture of these representations.
	`OBJNAME←4 6⍴⍳24`
	`R←9 13 'S' 'SHOE' OBJNAME`

Assignment can be to only a single name.
`A←2`
`B←2`
`C←'ANC'`
`D←2 3 4`
`E←2 3 4`
`F←2 3 4`
	Assignment can take a simple list of names enclosed in parentheses on the left. If the right argument is a vector of the same length, then each value from the right is assigned to the corresponding name on the left. If the right argument is a scalar, then each name is assigned to be the scalar item.
	`      (A B C)←2 3 'ANC'`
	`      A`
	`2`
	`      ⍴C`
	`3`
	`      (D E F)←⊂2 3 4`
	`      D`
	`2 3 4`
	`      ⍴E`
	`3`

Figure 10. VS APL Compared with APL2—Evaluation of Expressions (Compatible)

Evaluation of Expressions under VS APL	Evaluation of Expressions under APL2

Parentheses are used for grouping an expression to control the order of evaluation.
`      2×3+5`
`16`
`      (2×3)+5`
`11`
	Parentheses are used for grouping. They are correct if properly paired and if the contents within the parentheses evaluates to an array, function, or operator.
	Correct
	`      (2 4 5) 3`
	` 2 4 5  3`
	Correct, but parentheses around `∘.+` are redundant:
	`      (2 3⍴⍳6) (∘.+).× 3 2⍴-⍳6`
	`¯22 ¯28`
	`¯49 ¯64`

Figure 11. VS APL Compared with APL2—Display of Output (Compatible)

Display of Output under VS APL	Display of Output under APL2

In numeric output, multiple spaces on a line are compressed to a single space because they are redundant.
`     4    6    9    2`
`4 6 9 2`
	In numeric output, multiple spaces can display on a line to indicate the structure of items in a nested array.
	`      1 2 (3 4)`
	` 1 2  3 4`

Figure 12. VS APL Compared with APL2—Object Names (Compatible)

Object Names under VS APL	Object Names under APL2
The characters ¯ and _ are not valid in object names.	The characters ¯ and _ are valid in object names, provided they are not the first character.

Figure 13 (Page 1 of 2). VS APL Compared with APL2—Primitive Functions (Compatible)

Primitive Functions under VS APL	Primitive Functions under APL2

Compression (*L*/*R*) is a primitive function that requires a Boolean left argument.
`      1 1 0 1/'SHOE'`
`SHE`
	Replicate (*L*/*R*) is a derived function. Its left operand is a vector of Boolean, negative, or positive integers.
	`      1 0 3 2/'SHOE'`
	`SOOOEE`
	With a Boolean left argument, the derived function is called compress and is similar to VS APL compression:
	`      1 1 0 1/'SHOE'`
	`SHE`

Expansion (*L**R*) is a primitive function that requires a Boolean left argument.
`      1 1 0 1\'SHE'`
`SH E`
`      1 0 1\2 2⍴'SWIM'`
`S W`
`I M`
	Expand (*L**R*) is a derived function. It is similar to the VS APL expansion. Its left operand is a simple Boolean vector.
	`      1 0 1\2 2⍴'SWIM'`
	`S W`
	`I M`
	`      1 0 1\(2 3)(4 5)`
	`  2 3  0 0  4 5`

Figure 13 (Page 2 of 2). VS APL Compared with APL2—Primitive Functions (Compatible)

Primitive Functions under VS APL	Primitive Functions under APL2

Format (*L*⍕*R*) requires a numeric left argument.	Format (*L*⍕*R*) by specification behaves similarly to format under VS APL. Format (*L*⍕*R*), by example, takes a character left argument that serves as a model for the format of the corresponding column.
	`      ' 55@ $53.50 EA' ⍕ 32 9.17`
	` 32@  $9.17 EA`

Figure 14. VS APL Compared with APL2—Selective Specification (Compatible)

Selective Specification under VS APL	Selective Specification under APL2

Bracket Indexing (*A*[*I*]) is the only primitive function that allows you to assign a value to selected items in an array.
`      A←23 44 97`
`      A[2]←80`
`      A`
`23 80 97`
	Bracket Indexing (*A*[*I*]), and other functions that select positions from a named array, allow you to assign a value to selected items in arrays.
	`      A←1 2 ¯1 5`
	`      ((A=¯1)/A)←0`
	`      A`
	`1 2 0 5`

Figure 15. VS APL Compared with APL2—Primitive Operators (Compatible)

Primitive Operators under VS APL	Primitive Operators under APL2
Derived functions can be created by applying a primitive operator to a primitive function.	Derived functions can be created by applying a primitive or defined operator to one or two of the following operands: primitive functions, defined functions, derived functions, or arrays.

Figure 16. VS APL Compared with APL2—System Functions and Variables (Compatible)

System Functions and Variables under VS APL	System Functions and Variables under APL2
Name List (⎕*NL*) accepts as its argument any integer 1 through 3.	Name List (⎕*NL*) accepts as its argument any integer 1 through 4 (with 4 meaning a defined operator).
Printing Precision (⎕*PP*) can be as great as 16.	Printing Precision (⎕*PP*) can be as great as 18.

Figure 17. VS APL Compared with APL2—Debugging and Processing (Compatible)

Debugging and Processing under VS APL	Debugging and Processing under APL2
Trace *T*∆ and Stop *S*∆ controls cannot be referenced.	Trace *T*∆ and Stop *S*∆ controls can be referenced.
Interrupted processing of statements entered in immediate execution mode cannot be resumed.	Interrupted processing of a statement can be resumed at the point where the processing has halted by using →⍳0.
The state indicator, listed by the)*SI* command, contains a list of the calling sequence of defined functions (with their pertinent line numbers) that led to the current state. To indicate suspended immediate processing statements, the state indicator includes an asterisk '⋆' on the line containing the name and function line number of the first function called.	As with VS APL, the state indicator, listed by the)*SI* command, contains a list of the calling sequence of defined functions and defined operators that led to the current state. To indicate each suspended immediate execution statement, the state indicator includes an asterisk '⋆' on a line by itself. Asterisks can indicate suspended immediate execution statements that do not call a function. The)*SIS* command lists the state indicator with the function name, pertinent line number, and corresponding statement in the definition. Suspended immediate execution statements are listed, and each is preceded by an asterisk. Two carets below each statement indicate where the processing has halted and where any error has occurred.)*SI*,)*SINL*, and)*SIS* commands also allow an optional numeric argument to indicate the number of levels of the state indicator to be displayed, for instance)*SI* 4.
The right arrow '→' clears the most recent line(s) placed in the state indicator.	As with VS APL, the right arrow '→' clears the most recent line(s) placed in the state indicator. The system command)*RESET* *n* or)*SIC* *n* clears *n* lines from the state indicator.)*RESET* or)*SIC* clears all lines.

Figure 18. VS APL Compared with APL2—System Commands (Compatible)

System Commands under VS APL	System Commands under APL2
)*FNS* and)*VARS* allow you to specify the beginning letter or set of letters for listing functions and variables in the active workspace.	)*FNS* and)*VARS* allow you to specify the beginning and ending letter or set of letters for listing functions and variables in the active workspace.
)*SYMBOLS* allows you to increase the maximum size of the symbol table, but can only be used in an empty workspace.	)*SYMBOLS* is usually unnecessary because the symbol table dynamically expands as the number of symbols increase. For efficiency, you can specify a symbol table size, and can do so at any time.)*SYMBOLS* also causes the unused symbols in a workspace to be removed.
)*LIB* allows you to specify alphabetic letter(s) for beginning the listing of workspace names.	)*LIB* allows you to specify alphabetic letters for beginning and ending the listing of workspace names.
)*OFF*[*HOLD*] and)*CONTINUE*[*HOLD*] provide an optional *HOLD* parameter that allows you to return to the host system. If you do not specify the *HOLD* parameter, you are logged off the host system.	)*OFF*[*HOLD*] and)*CONTINUE*[*HOLD*] always return control to the host system. The optional *HOLD* parameter has no effect.

Extended-Compatible System Features

For more information on most features listed in Figure 19 through Figure 21 on page 20, see *APL2/370 Programming: System Services Reference*. For more information on installation features, see *APL2/370 Installation and Customization under CMS* or *APL2/370 Installation and Customization under TSO*.

Figure 19. VS APL Compared with APL2—Invocation Options (Compatible)

Invocation Options under VS APL	**Invocation Options under APL2**
The DEBUG option provides the following debug settings: msg (1) echo (2) abend (64) micro (128)	APL2 retains the following DEBUG settings from VS APL: msg (1) echo (2) Abend (64) has been dropped, but a similar facility is available under TSO through SYSDEBUG, a new invocation option described in *APL2/370 Diagnosis Guide*. Micro (128) has been dropped. The following DEBUG settings have been added, or replace VS APL settings: **xdump (4)** Supply additional information in dumps **estimate (8)** Give estimates of times for selected long-running tasks **msgid (32)** Prefix messages with a message identifier **nolx (64)** Do not process ⎕*LX* during)*LOAD* **noquemsg (128)** Discards secondary messages rather than queuing them. **Note:** While 1-MSG is turned on, secondary messages are immediately displayed rather than queued, so the setting of this flag is irrelevant.
The HILIGHT option default results in no highlighting in CMS and highlighted output in TSO.	The HILIGHT option default results in highlighted input in CMS and highlighted output in TSO.
The SMAPL option indicates whether the local session manager should be used: `SMAPL(ON | OFF | TRY)` (TRY underlined) where `TRY` is treated as `ON` if possible, else as `OFF`.	The SMAPL option adds support for a remote session manager: `SMAPL(ON | OFF | TRY | nnnn)` where *nnnn* is a processor number. The APL2 interpreter shares a variable with that processor and handles all session input and output (⎕, ⍞, Editor 1, and immediate execution) through that variable rather than directly at the user's terminal. That processor number could be resolved as another APL session or as a session manager, and the process could reside on the same system or on another system. The communication protocol is defined in *APL2/370 Programming: System Services Reference*.
The TERMCODE option identifies the type of terminal you are using in TSO.	TERMCODE (-1) can also be used in either CMS or TSO to tell APL2 to redirect APL2 input and output to files instead of to the terminal.

Figure 20. VS APL Compared with APL2—Editors (Compatible)

Editing under VS APL	**Editing under APL2**
The licensed program provides a line editor for function editing. A full-screen editor that edits functions and character variables is available as an IUP to VS APL.	The licensed program provides a line editor, a full-screen editor, and access to user-written and system editors. Both the full-screen and system editors edit functions, operators, and simple character variables. See "New System Features" on page 13 for more details.
The editor command `[`*n*`⎕]` displays line *n* of the function definition.	The editor command `[⎕`*n*`]` displays line *n* of the function or operator definition. `[`*n*`⎕]` results in a *DEFN ERROR*.
The editor command `[⎕`*n*`]` displays line *n* through the last line of the definition.	The editor command `[⎕`*n*`-]` displays line *n* through the last line of the definition.
Editing the function name in a definition replaces the current name of the function.	Editing the function or operator name in a definition creates a new function or operator with the edited name. The existing function or operator remains as it was.
In VS APL local functions cannot be edited.	With the APL2 editors, local functions or operators can be edited. When local and global functions with the same name exist, only the local function can be accessed by the APL2 editors.

Figure 21. VS APL Compared with APL2—Auxiliary Processors (Compatible)

Auxiliary Processors Distributed with VS APL	**Auxiliary Processors Distributed with APL2**
Auxiliary processors distributed with VS APL support only VS APL data types.	AP 110, AP 111, AP 119, AP 121, AP 127, AP 210, and AP 211 support the new APL2 data types. AP 110, AP 111, AP 123, and AP 210 have new options that provide mapping between the VS APL internal character encoding and APL2 internal character encoding. Changes in default translation and initialization options for auxiliary processors are discussed on page 35. Several of the VS APL APs have been enhanced to support additional function. For example, AP 100 can be used to determine the name of the host system, and AP 124 supports color.

Extended-Incompatible Features

Figure 22 through Figure 27 on page 24 compare features of VS APL to similar features of APL2. These extended features of APL2 are incompatible—they can prevent an application that runs under VS APL from producing the expected results under APL2.

Extended-Incompatible Language Features

For more information on features listed in the following figures, see *APL2 Programming: Language Reference*.

Figure 22. VS APL Compared with APL2—Evaluation of Expressions (Incompatible)

Evaluation of Expressions under VS APL	Evaluation of Expressions under APL2

VS APL's evaluation of brackets allows you to select an item from a vector constant.
`      4 5 6[3]`
`6`
	APL2's evaluation of brackets results in a *RANK ERROR*, if you attempt to select an item from a numeric vector constant. An expression such as `4 5 6[3]` is interpreted as:
	`      4 5 (6[3])`
	`RANK ERROR`
	`      4 5(6[3])`
	`          ^^`
	When processing VS APL defined functions that contain such expressions, the `)MCOPY` and `)IN` system commands insert parentheses around numeric vector constants, so that the expressions are evaluated correctly under APL2. Parentheses are not inserted into character vectors. See also "If the Function Is Interrupted" on page 32.

A suspended function can be restarted at the suspended line with any of the following expressions:
`→⎕LC`
`→''`
`→⍳0`
	A suspended function or operator can be restarted at the suspended line with the following expression:
	`→⎕LC`
	It can be restarted at the point of suspension with either of the following expressions:
	`→''`
	`→⍳0`

Dyadic defined functions cause a syntax error if invoked monadically.	All functions are ambi-valent. Dyadic functions can be invoked monadically.
Stop vectors in an unlocked function invoked by a locked function are honored.	Stop vectors in an unlocked function invoked by a locked function are ignored. Stop vectors in functions are also ignored if the function is run under `⎕EC`. However, they are honored in a function invoked from `⎕EA`.
Functions containing duplicate labels use the first definition of the label.	Functions or operators containing duplicate labels use the last definition of the label.
VS APL ignores local names in a defined function that duplicates the names of the arguments or the result.	APL2 permits duplication of names in the header of a defined function or operator. The rightmost occurrence of a duplicate name in the header is taken as its definition.
VS APL provides blanks where it expects them to be, for example, `⍎ '1F 2'` is processed as `⍎ '1 F 2'`.	APL2 does not always insert blanks. It returns a *SYNTAX ERROR* for `⍎ '1F 2'`.

Figure 23. VS APL Compared with APL2—Default Output

VS APL Default Output	APL2 Default Output
Default output of arrays with rank ≥2 folds the output on a line-by-line basis where the width of ⎕*PW* is exceeded.	Default output of arrays with rank ≥2 folds the output on a plane-by-plane basis where the width of a plane exceeds ⎕*PW*.
Default output of empty arrays always produces one line.	Default output of empty arrays produces as many lines as there are rows in the array.

Figure 24. VS APL Compared with APL2—Primitive Functions (Incompatible)

Primitive Functions under VS APL	Primitive Functions under APL2

Power (*L***R*) returns the odd root of a negative number in the form of a real number. The even root of a negative number results in a *DOMAIN ERROR*.
`      ¯8*÷3`
`¯2`
`      ¯8*÷2`
`DOMAIN ERROR`
`      ¯8*÷2`
`      ^`
	Power (*L***R*) returns the odd root of a negative number (a complex number) as its principal value. Power also returns the even root of a negative number.
	`      ¯8*÷3`
	`1J1.732050808`
	`      ¯8*÷2`
	`0J2.828427125`

Monadic Format (⍕*R*) formats columns of a numeric array so that each column has the same width. It includes a column of leading blanks for arrays with rank 2 or greater.	Format (Default) (⍕*R*) formats each column according to the item with the greatest width in the column. It does not include a column of leading blanks. In APL2, a column of leading blanks indicates nesting.
Dyadic Format (*L*⍕*R*) includes a blank for the unit's place if any number in the right argument *R* is less than 1, and the digits part of the left argument *L* is a nonzero integer.	Format by specification (*L*⍕*R*) does not include a blank column for the units digit if all numbers in a column of the right argument are less than 1.

Circle (*L*○*R*) accepts the integers ¯7 through 7 as valid left arguments. The result of ¯4○*R* is a positive square root.
`     ¯4 ○ 2`
`1.732050808`
`     ¯4 ○ ¯2`
`1.732050808`
	Circle (*L*○*R*) accepts the integers ¯12 through 12 as valid left arguments. The result of ¯4○*R* is a negative square root if *R* is negative.
	`      ¯4 ○ 2`
	`1.732050808`
	`      ¯4 ○ ¯2`
	`¯1.732050808`

Residue (*L* | *R*) has no implicit arguments.
`      ⎕PP←16`
`      ⎕CT←1E¯13`
`      1|.99999999999999`
`0.99999999999999`
	Residue (*L* | *R*) uses ⎕*CT* as an implicit argument.
	`      ⎕PP←16`
	`      ⎕CT←1E¯13`
	`      1|.99999999999999`
	`0`

One-element arrays are extended in primitive dyadic scalar functions.	Only scalars and one-element vectors are extended in primitive dyadic scalar functions.

Figure 25. VS APL Compared with APL2—Inner Product (Incompatible)

Inner Product under VS APL	Inner Product under APL2

The shape of the arguments for the general case of `P f.g Q` are conformable if the last axis of P is equal to the first axis of Q. The following example produces a length error:
`A ← 15 1ρ 'A'`
`A∧.= 'AA'`
	The shape of the arguments for the general case of `P f.g Q` are conformable if the shape of the arguments `P` and `Q` are conformable for the function `g`.

Figure 26 (Page 1 of 2). VS APL Compared with APL2—System Functions and Variables (Incompatible)

System Functions and Variables under VS APL	System Functions and Variables under APL2
A character output (⍞) assignment immediately followed by a character input prompt (⍞) returns a vector containing the prompt and response, except that the prompt can be partially or entirely replaced by blanks depending upon the device type, VS APL release, and whether GDDM is used.	A character output (⍞) assignment immediately followed by a character input prompt (⍞) returns a vector composed of the response preceded by one of the following: • The characters composing the prompt, if the Prompt Replacement system variable is specified as `⎕PR←''`. • A repeated character, specified in `⎕PR`, which replaces the prompt portion of the resulting vector. A blank is the default setting of `⎕PR`.
If the character input prompt (⍞) is on a different line from the character output, entering a single-character response results in a scalar.	If the character input prompt (⍞) is on a different line from the character output, entering a single-character response results in a one-item vector.
The Atomic Vector (`⎕AV`) uses an ordering unique to VS APL. The alphabet is contiguous.	The Atomic Vector (`⎕AV`) uses an ordering that conforms with EBCDIC and is different from the ordering under VS APL. The alphabet is not contiguous.
When Canonical Representation (`⎕CR`) returns the character representation of a defined function, it limits the precision of any numeric constants to 17 significant digits—the same precision allowed by the line editor.	Canonical Representation (`⎕CR`) returns the character representation of a defined function or operator, not limiting the precision of any numeric constants.
Expunge (`⎕EX`) does not erase defined functions that are suspended or waiting to complete processing.	Expunge (`⎕EX`) erases defined functions that are suspended or waiting to complete processing. However, erasing such functions does not affect their definitions in the state indicator.
Fix (`⎕FX`) can be applied to any character matrix. It can be used to fix the definition of a function, including a function that replaces another function that is neither suspended nor waiting to complete processing.	Fix (`⎕FX`) can be applied to a character matrix or a vector of character scalars or vectors. It can be used to fix the definition of a function or operator. The function or operator being defined can replace any existing operation, including one that is suspended or waiting to complete processing. `⎕FX` has also been extended to accept a left argument, which enables you to set execution properties for a defined function or defined operator.
Name Class (`⎕NC`) returns a class of `0` through `4`, with `4` indicating an invalid name for an object.	Name Class (`⎕NC`) returns a class of `¯1` through `4`, with `¯1` indicating an invalid name or unused distinguished name for an object. `4` indicates a defined operator name. `⎕NC` can also be applied to system functions and system variables.

Figure 26 (Page 2 of 2). VS APL Compared with APL2—System Functions and Variables (Incompatible)

System Functions and Variables under VS APL	System Functions and Variables under APL2
Shared Variable Offer (⎕*SVO*) extends any left argument that is a one-item vector or scalar, so that it is used as the processor number for each name represented in the right argument.	Shared Variable Offer (⎕*SVO*) extends any left argument that is a scalar, so that it is used as the processor number for each name represented in the right argument. A left argument that is a one-item vector corresponds with a right argument of only one name.
The result of ⎕*EX*, ⎕*NC*, ⎕*SVO*, or ⎕*SVR* applied to a scalar or vector is a one-item vector.	The result of ⎕*EX*, ⎕*NC*, ⎕*SVO*, or ⎕*SVR* applied to a scalar or vector is a scalar.
The value of ⎕*WA* depends on the internal format of VS APL objects.	The value of ⎕*WA* depends on the internal format of APL2 objects and often differs substantially from the value obtained in a VS APL environment.
The)*COPY* command, when used to copy an entire workspace, copies all user objects in the workspace, but not system variables.	The)*COPY* command, when used to copy an entire workspace, includes the system variables ⎕*CT*, ⎕*FC*, ⎕*IO*, ⎕*LX*, ⎕*PP*, ⎕*PR*, and ⎕*RL* as well as all user objects in the workspace.

Extended-Incompatible System Features

For more information on most features listed in the following figures, see *APL2/370 Programming: System Services Reference*.

Figure 27 (Page 1 of 2). VS APL Compared with APL2—Auxiliary Processor Options (Incompatible)

Auxiliary Processor Translation Options under VS APL	Auxiliary Processor Translation Options under APL2
Auxiliary processors distributed with VS APL provide default translation options.	Certain auxiliary processors distributed with APL2 provide default translation options that differ from those provided under VS APL. For a description of these changes, see “Changes in APL2 Auxiliary Processor Translation Options” on page 35.
AP 110, AP 111, and AP 210 provide a BYTE option that allows for no translation regardless of the encoding of the file.	The BYTE option works only for some applications that use it under VS APL. See “Changes in APL2 Auxiliary Processor Translation Options” on page 35 for a discussion on files and auxiliary processors.
AP 101 provides the options 370 and APL.	AP 101 no longer provides the options 370 and APL. If they are specified in the initial value of the shared variable, that value is considered invalid. Valid translation options in APL2 are 192 or EBCD.

AP 101 stacked input cancels the session manager *SUPPRESS* command.	AP 101 stacked input no longer cancels the session manager *SUPPRESS* command. Entering the following lines cancels the *SUPPRESS* command:
	`3 11 ⎕NA 'OPTION'`
	`'OFF' OPTION 'QUIET'`

Figure 27 (Page 2 of 2). VS APL Compared with APL2—Auxiliary Processor Options (Incompatible)

Auxiliary Processor Translation Options under VS APL	**Auxiliary Processor Translation Options under APL2**
User-written auxiliary processors do not handle the new APL2 data types.	A new return code `12-68` from ASVPREF or ASVPCPY enables user-written auxiliary processors to detect data that cannot be represented in VS APL data format. To allow VS APL auxiliary processors to handle all APL2 data types, the VS APL SVP interface has been extended in APL2 to support new APL2 data types through the ASVDFORM executable macro and the AP2SDF mapping macro that defines the parameter block used by ASVDFORM.
ASVPQRY can return information about multiple variables or multiple partners in a single request.	ASVPQRY only supports queries of a single variable or processor for each call.
Under TSO, CLISTs invoked through AP 100 are processed by APL itself. APL provides support for the special statements $EXIT, $REPEAT, and $RETURN, and CLIST processing is deferred under certain circumstances. When not deferred, the CLIST used to invoke APL is resumed as a part of the AP 100 processing if one of the special statements is not used.	CLISTs invoked through AP 100 are now handled by the TSOLNK facility rather than by APL. There is no support for $EXIT, $REPEAT, and $RETURN. &LASTCC is always returned to the AP 100 user. All CLISTs are processed immediately, and the CLIST used to invoke APL is never resumed until APL2 termination.

VS APL Features No Longer Supported

Figure 28 and Figure 29 on page 26 list VS APL features not supported under APL2 and any replacements.

Language Features No Longer Supported

For more information on features listed in Figure 28, see *APL2 Programming: Language Reference*.

Figure 28. Language Features No Longer Supported under APL2

Dropped from VS APL	Replacement in APL2
Dyadic Shared Variable Query ⎕*SVQ*	No replacement.
)*GROUP*,)*GRP*,)*GRPS*	)*COPY*,)*MCOPY*,)*PCOPY*, and)*ERASE* can copy or erase indirectly. When issuing one of these commands, you specify in parentheses the name of a matrix containing the names of objects to be processed.
)*STACK*	No replacement. Handled automatically.
)*WSSIZE*	No replacement.
WAS lib wsname, which is displayed with loading of the *CONTINUE* workspace under TSO	No replacement.
Horizontal Tabs ⎕*HT*	No replacement. APL2 accepts the name ⎕*HT*. However, referencing ⎕*HT* yields ⍳0 no matter how it is specified.

System Features No Longer Supported

For more information on features listed in Figure 29, see *APL2/370 Programming: System Services Reference*.

Figure 29. Facilities No Longer Supported under APL2

Dropped from VS APL	Replacement in APL2
A numeric ID could be specified as a first positional parameter when invoking VS APL under CMS.	The ID keyword provides equivalent function.

Workspaces

The workspaces distributed with VS APL and APL2 under CMS and TSO are summarized in Figure 30.

Figure 30. Workspaces Distributed with VS APL and APL2

Description of Workspace	VS APL CMS	VS APL TSO	APL2 CMS	APL2 TSO
APL file access	APLDATA	APLDATA	APLDATA	APLDATA
Interactive Chart Utility interface			CHARTX	CHARTX
Environment-dependent auxiliary processors	CMS	TSO	CMS	TSO
Conversion-migration	CONVERT	CONVERT	TRANSFER	TRANSFER
Data structure display			DISPLAY	DISPLAY
Usage examples	EXAMPLES	EXAMPLES	EXAMPLES	EXAMPLES
File transferring (TSO)	.	FILESERV	.	FILESERV
Data formatting	FORMAT	FORMAT		
AP 124 full-screen facilities	FSC124	FSC124		
AP 126 facilities (FSC124 compatible)	FSC126	FSC126	FSC126	FSC126
Panel design	FSDESIGN	FSDESIGN		
AP 126 (GDDM full-screen facilities)	FSM	FSM	FSM	FSM
AP 126 (GDDM cover function)			GDMX	GDMX
Graphics	GRAPHPAK	GRAPHPAK	GRAPHPAK	GRAPHPAK
Edit descriptions	HOWEDITS	HOWEDITS		
Mathematical functions			MATHFNS	MATHFNS
Text editing	MEDIT	MEDIT	MEDIT	MEDIT
News bulletins	NEWS	NEWS		
Nondisplay graphics	PLOT	PLOT	.	.
Printing	PRINTCMS	PRINTTSO	PRINTWS	PRINTWS
Example	SBIC	SBIC		
Text editing	SEDIT	SEDIT		
Access to SQL database			SQL	SQL
External function directory			SUPPLIED	SUPPLIED
Miscellaneous functions	UTILITY	UTILITY	UTILITY	UTILITY
Simple database	VAPLFILE	VAPLFILE	VAPLFILE	VAPLFILE
VSAM data access	VSAMDATA	VSAMDATA	VSAMDATA	VSAMDATA
Workspace descriptions	WSINFO	WSINFO	WSINFO	WSINFO

External Functions Distributed with APL2

Figure 31 lists the external functions distributed with APL2 under CMS and TSO. See *APL2/370 Programming: Using the Supplied Routines* for more information about the external functions.

Figure 31 (Page 1 of 2). APL2/370 External Routines

External Routine	Function
Data Conversion	
ATR	Convert an APL array to a record with mixed data types
CTK	Convert extended character data to mixed DBCS data
CTN	Convert character data to numeric data
DFMT	Format an array of extended character data
KTC	Convert mixed DBCS data to extended character data
PFA	Generate a pattern for *ATR* or *RTA*
RTA	Convert a record to an APL array
CAN[1]	Compress and Nest
DAN[1]	Delete and Nest
SAN[1]	Slice and Nest
External Routine Support	
APL2PI	A niladic form of *APL2PIE*
APL2PIE	Interface with non-APL programs that call APL2.
ATP	Update parameters passed by a non-APL program
BUILDRD	Build a routine description for an external routine
BUILDRL	Build a routine list for a module containing external routines
EXP	Request APL evaluation in the previous name scope
PACKAGE	Convert a workspace to a namespace
PTA	Extract parameters passed by a non-APL program
QNS	Query the current name scope
APL Object Access	
EDITOR2[2]	A program interface to Editor 2
EDITORX[2]	A program interface to a named system editor
IN[2]	Program access to system command)*IN*
OUT[2]	Program access to system command)*OUT*
PIN[2]	Program access to system command)*PIN*
REXX Access (Processor 10)	
Δ*EXEC*	Execute a REXX program
Δ*F*	Obtain information about a CMS or MVS file
Δ*FM*	Read or write a file as a matrix
Δ*FV*	Read or write a file as a vector of vectors
System Data Access	
CSRIDAC	Access an MVS/ESA* virtual data object
CSRREFR	Refresh an MVS/ESA virtual data object
CSRSAVE	Save changes to a permanent MVS/ESA virtual data object
CSRSCOT	Save MVS/ESA virtual data object changes in a scroll area
CSRVIEW	Define a view on an MVS/ESA virtual data object
DSQCIA	Interact with the database Query facility

Figure 31 (Page 2 of 2). APL2/370 External Routines

External Routine	Function
Environment Control	
ATTN	Query or reset the attention flag
MSG	Use APL2 message facilities from an application
OPTION	Query or set APL2 invocation options
PBS	Query or set the)*PBS* state
RAPL2[2]	Run the remote-session manager
SERVER	Start a TCP/IP port server
SVI	Determine shared variable processor numbers or user IDs.
Usage and Debugging Aids	
CMSIVP	Installation verification under CMS
DISPLAY	Display an array in a form that shows nesting and data types
DISPLAYC	The same as *DISPLAY*.
DISPLAYG	The same as *DISPLAY*, but using box characters
FED	Diagnostic tool for IBM service usage
HELP	Obtain information from APL2HELP files
IDIOMS[2]	Search the APL2 phrase collection
TIME	Performance monitoring within a workspace
TSOIVP	Installation verification under TSO

Notes:

1. The Partition primitive (⊂) should be used instead of these three functions.
2. Not supplied with Application Environment.

Chapter 4. Testing and Debugging

After transferring workspaces to APL2, you can verify that their functions continue to run properly by doing the following:

- Inspect and correct statements for the effects of changes, and test the altered functions under APL2
- Check data files used by functions in the transferred workspaces
- Check alternate input on the AP 101 stack
- Check user-written auxiliary processors used by functions in the transferred workspaces
- Test the application as a whole

Inspecting, Correcting, and Testing Functions

"Extended-Compatible Features" on page 15 lists features of APL2 that can prevent an application that runs under VS APL from producing the expected results under APL2. Inspect and correct statements in functions that use those features. Pay particular attention to the following:

- Primitive functions:
  - Format (default) (⍕*R*) and format by specification (*L*⍕*R*)
  - Residue (*L* | *R*)
- System functions and variables:
  - Name class (⎕*NC*)
  - Data received from a character input/output (⍞) request for user response
  - Fix (⎕*FX*)
- High numerical precision
- Arguments as local names

The problems that can occur with each of these are discussed later in this section. Additionally, the *TRANSFER* workspace distributed with APL2 contains functions to help you locate these incompatible features in your functions and to fix them.

Using the TRANSFER Workspace

DESCRIBE in the *TRANSFER* workspace explains the functions and variables in the workspace. To use the *TRANSFER* workspace:

1. Use)*LOAD* *transws*, where *transws* is the name of the transferred workspace.
2. Use)*PCOPY* 2 *TRANSFER* (*GPTRANSFER*) to add the *TRANSFER* workspace contents.

   When you use)*PCOPY* you are alerted to name conflicts you must resolve. If an object in *transws* and an object in the *TRANSFER* workspace have the same name, consider renaming one of them.
3. Use)*PCOPY* 2 *TRANSFER* *AV_VSAPL* or)*COPY* 2 *TRANSFER* *AV_APLSV* if necessary for this workspace.
4. Use)*SAVE* *testws*, where *testws* is a new name for the workspace.

   Some of the functions in the *TRANSFER* workspace modify workspace contents. Setting up a new workspace in which to make corrections keeps the original transferred workspace intact as a backup.

5. Make your corrections after using functions in the *TRANSFER* workspace.
6. Use)*ERASE* (*GPTRANSFER*) to eliminate *TRANSFER* workspace functions and variables.

   Warning: If you had name conflicts when copying in Step 2, you may not want to use the indirect erase unless you have changed the names of your functions.
7.)*SAVE*.

When you are satisfied that the corrected functions run properly, you can rename your test workspace.

Some of the *TRANSFER* workspace functions and variables are described below. Others are described with the specific problem they are designed to remedy.

*ALL_* creates a list of names of all defined functions in the workspace you are debugging. It excludes all *TRANSFER* workspace functions from the list. The result of *ALL_* can then be an argument for any function that requires a list of function names as an argument.

*FLAG_* searches for given character strings and returns a list of all the statements that contain those strings. Each statement is preceded by the function name and line number. *FLAG_* takes two arguments: the right argument is a list of functions to examine; and the left argument is a list of character strings to be searched for. For example, the following expression searches for all occurrences of residue and ⎕*FX*:

```
'|' '⎕FX' FLAG_  ALL_
```

If you specify no search argument, *FLAG_* prompts for character strings, which you enter one at a time.

The *TRANSFER* workspace includes the variable *FLAGMVSAPL_*; this is a prepared list of character strings, which includes all known migration problem areas. *FLAGMVSAPL_* is a convenient argument for *FLAG_*. For example, the following expression searches for all known migration problem areas in all functions in the workspace:

```
FLAGMVSAPL_  FLAG_  ALL_
```

*FIX_* is used to make changes that require simple string replacements. *FIX_* takes two arguments: the right argument is a list of functions requiring change, and the left argument is a set of old and new pairs, nested together.

The *TRANSFER* workspace includes the variable *FIXMIUP_*; this a prepared list of old and new pairs.

Warning: *FIX_* modifies the workspace. Be sure that you want to make the changes you have indicated in all examples in all functions listed. *FLAG_* can be used to identify all examples of the old character strings for inspection before you use *FIX_*. Also, although *FLAGMVSAPL_*, *FIXMIUP_*, and *ALL_* are available arguments to *FIX_*, you may want to enter the arguments more selectively.

What to Look For

In many examples, the APL2 extensions do not affect the processing of your functions. Each should be flagged, however, and examined in the context of the application to determine whether a problem exists. Appropriate corrections should then be made. The extensions listed earlier in this section and discussed below are known to cause problems.

Residue (*L*|*R*): Results of primitive function residue (*L*|*R*) can be affected by ⎕*CT* (comparison tolerance) under APL2 but not under VS APL. Use *FLAG*_ to locate uses of residue. Carefully check functions that use residue under conditions of high-level precision. The least significant digits in the results under APL2 can differ from those in the results under VS APL.

Use of ⎕*NC*: A defined function can depend on ⎕*NC* (name class) to test for an invalid object name. Under VS APL, ⎕*NC* returns 4 for an invalid object name. Under APL2, ⎕*NC* returns ¯1. (4 indicates a defined operator name.)

Use *FLAG*_ to identify functions that use ⎕*NC* to test for invalid object names.

Local Variables: VS APL ignores localization of the name of an argument in the function header statement:

```
      ∇Z←FN A;A
```

APL2 permits localization. If the argument is referenced before it is specified as a local variable, a *VALUE ERROR* is generated. If the VS APL workspace was transferred with)*MCOPY*, these duplicate local names have been deleted; otherwise, use *CHKHDRS*_ to identify localized arguments. Then, delete them from the list of local names in the function headers.

Testing Functions under APL2

After you have checked and modified functions for the effects of changes and are ready to test them, migrate any test workspaces and data that you may have used previously under VS APL.

During testing, problems can surface in two different ways: they can interrupt processing of a function, or they can allow a function to complete but produce unexpected results.

If the Function Is Interrupted

| Processing of a function can be interrupted for many reasons. Two common
| causes of interruptions are unexpected APL2 data types and indexed numeric constants.

APL2 Data Types: A function can receive data that could not have existed in VS APL, either entered by a user or read from a file. For example, ⎕ input prompts until valid data is received, but no longer rejects '*AB*' 2.

To guard against accidental entry of the new APL2 data types, such as nested or mixed arrays, you can run the application under the control of ⎕*EA* (execute alternate). With ⎕*EA*, you can either branch to an error handling routine if an error results from new data types, or you can continue normal processing if no error occurs.

Indexed Numeric Constants: A function under VS APL can index an item in a numeric vector constant:

```
      4 5 6[3]
6
```

Under APL2, the expression is equivalent to the following one, which results in a *RANK ERROR* because the brackets are tightly bound to the name on their left.

```
      4 5 (6[3])
RANK ERROR
      4 5(6[3])
          ^^
```

In functions transferred from VS APL to APL2 with the `)MCOPY` command, such vector constants must be parenthesized:

```
      (4 5 6)[3]
6            ^
```

You must add parentheses yourself or use the command `)CS 1` (compatibility setting) to have indexed numeric constants enclosed in parentheses in the following situations:

- If the function was not transferred using `)MCOPY` or `)IN`
- If the function comes from a file other than a transfer file (using an auxiliary processor)
- If the function uses execute (⍎) on a character vector that contains an indexed numeric constant vector, for example `⍎'4 5 6[3]'`

Use of `)CS`: The system command `)CS` (compatibility setting) provides a temporary solution to the source of two common migration problems:

- Indexed numeric constants in functions not transferred by `)MCOPY` or `)IN`
- Inadvertent use of vector notation for other than numeric vector constants

The syntax of `)CS` is `)CS` *n*, where *n* can have one of the following values:

0	APL2 (the default).
1	Indexed numeric constant vectors produce results as they do in VS APL. When this setting is used, APL2 inserts the parentheses as it does when `)MCOPY` or `)IN` is used to transfer workspaces. Thus, functions are displayed with parentheses enclosing numeric constant vectors.
2	Vector notation is restricted to numeric vector constants, so that expressions like `(1 2)(3 4)`, `'A' 'B' 'C'`, and `4 3 FDS` (where *FDS* is a variable) generate a *SYNTAX ERROR* as they do under VS APL.
3	Produces the combined effects of settings 1 and 2.

Use `)CS` without a parameter to query the current setting.

The compatibility setting is saved and loaded with the workspace; however, it is not copied if the workspace is copied.

Warning:)*CS* should be regarded as a temporary measure, particularly in disabling vector notation. Functions should be changed as described in “APL2 Data Types” on page 32.

Also, you should never copy an APL2 workspace into a workspace whose compatibility setting is not 0, because of the chances of failure or inappropriate results.

If the Function Is Not Interrupted

Problems that do not stop a function from processing can be detected from examining the output and comparing it to the VS APL output of the function run against the same data. In the case of a report that does not align properly, the problem is evident immediately. Sometimes problems, especially those involving numerical differences between APL2 and VS APL, are detected only after considerable testing or actual use of a function under APL2. If your function is giving erroneous results, check the following:

- Use of format (⍕) functions
- Use of ⎕*AV*
- ⎕*PR* setting
- Numerical precision

Formatting: Reports or results that you produce with format (default) (⍕*R*) or format by specification (*L*⍕*R*) can differ under APL2 because:

- Format (default) does not add a column of leading blanks in front of numeric arrays of rank 2 or greater.
- Format (default) and, in some cases, format by specification determine the width of each column according to the item of the greatest width. Because of this independent column formatting, the widths of columns in the result can vary from VS APL to APL2.

⎕*AV* ***Dependencies:*** If you did not adjust your functions for ⎕*AV* dependencies before transferring them to APL2, you should now either modify them, or rewrite the functions to avoid ⎕*AV* dependencies. For more information, see “Adjusting for ⎕*AV* Dependencies” on page 4.

Using Character Input/Output: The new system variable ⎕*PR* (prompt replacement) affects the behavior of ⍞. VS APL functions that use character input/output ⍞ to request and receive input on the same line are usually compatible under APL2 as long as ⎕*PR* is set to the default (a blank). Consider editing your functions to include ⎕*PR* as a local variable and specify it to be a blank (⎕*PR*←' ').

Numerical Precision: APL2 provides different numerical precision than does VS APL. If your defined functions carry results to a high level of precision, check the least significant digits in those results for possible variations between the results produced by VS APL and those produced by APL2. To display full precision in APL2, set ⎕*PP* to 18.

Checking Data Files Used by a Function

The most common problems with data files result from using the encoding for reading and writing. For example, a function that writes a file without conversion produces Z-code under VS APL and EBCDIC under APL2.

Two alternatives for making defined functions and data files compatible are:

- Using the character translation options provided by auxiliary processors distributed with APL2
- Rewriting applications to take advantage of APL2 and, possibly, to improve the efficiency of the applications

The first alternative provides the simplest means of correcting any translation problems. The second alternative can be considered as a long range solution. This section focuses on the first alternative–using translation options provided by auxiliary processors. See *APL2 Programming: Language Reference* for information on how to rewrite functions, and see *APL2/370 Programming: System Services Reference* for a description of APL2's capabilities.

Changes in APL2 Auxiliary Processor Translation Options

Character translation options offered by auxiliary processors distributed with APL2 can prevent many problems in reading from or writing to files in internal or other encoding. Problems can arise because the auxiliary processor and translation option used are no longer compatible with the encoding in a file.

Figure 32 summarizes the changes in default translation options for each auxiliary processor distributed with APL2.

Figure 32 (Page 1 of 2). Changes in Translation Options for APL2 Auxiliary Processors

AP Environment	VS APL Option	VS APL Conversion	APL2 Option and Notes
100 CMS	none	*A̲* to a . . . ^ to & . . .	370(default) – equivalent
			EBCD – no conversion
100 TSO	none	Full 256 EBCDIC	*EBCD*(default) – equivalent
			370 – like CMS 370
101 CMS	*APL*	*A̲* to *A*(bsp)_ . . .	Not supported
	370(default)	*A̲* to a . . . ^ to & . . .	Not supported
	192	Full 256 EBCDIC	192(default)–equivalent
101 TSO	none	Full 256 EBCDIC	none – equivalent

Figure 32 (Page 2 of 2). Changes in Translation Options for APL2 Auxiliary Processors

AP	Environment	VS APL Option	VS APL Conversion	APL2 Option and Notes
110	CMS	*APL*	*A̲* to *A*(bsp)_ . . .	*APL* – equivalent
		192	Full 256 EBCDIC	*EBCD* (or 192) – equivalent
		370	*A̲* to a . . . ^ to & . . .	370 (or *BCD*) – equivalent
		BIT	11000001 to *A* . . .	*BIT* – equivalent
		BYTE	Unchanged for copying or ⎕*AV* indexing	*BYTE* – no converted
		BYTE	Unchanged for storing APL character data	*COD*1 – EBCDIC to VS APL
		BYTE	With translate table	*BYTE* – equivalent with conversion table
111	ALL	*APL*		
210	TSO	192(or *EBCD*)	See 110 CMS	See 110 CMS
		370(or *BCD*)		
		BIT		
		BYTE		
111	TSO	*TN*	Superscripts, plotting characters	*TN* – equivalent
210	TSO			
123	ALL	none	Unchanged for copying or ⎕*AV* indexing	*T*(default) – equivalent *T*1 – equivalent or
		none	Unchanged for storing character data	*T*2 – store in EBCDIC *T* – equivalent with converted table
		none	With translate table	
126	ALL	¯4 1(default).	256 EBCDIC on data, tables/symbol sets unchanged.	¯4 0(default), if tables/symbol sets unconverted
		¯4 0	Nothing changed	¯4 1 if tables/symbol sets converted **Note:** For AP 126, the meanings of the conversion options have changed. In VS APL, ¯4 1 converted *data* between ZCODES and EBCDIC. In APL2, ¯4 1 converts *tables* between scrambled EBCDIC and EBCDIC. Either of the old options may need to be converted to either of the new options, depending on what conversion has been applied to the tables.

Possible Translation Problems

The following summarizes the possible problem areas for each auxiliary processor:

AP 100 (CMS Command Processor): Functions that used AP 100 under VS APL should operate under APL2.

AP 101 (Alternate Input Processor): Problems involving AP 101 are discussed in "Checking Alternate Input" on page 39.

AP 110 (CMS File Processor): This processor is compatible, except for the *BYTE* option. Data can be read or written compatibly, using the *BYTE* option, if:

- The file **does not** contain character data. On either system, the *BYTE* option does no translation. (For example, packed decimal numbers remain as packed decimal numbers.)
- Character data is not being displayed, converted, tested or modified within APL. (For example, it can successfully be rearranged and written to another file.)
- Character data is being generated or decoded using ⎕*AV*. (This is functionally equivalent to the *BIT* option.)
- An application is using the *BYTE* option and a translation table. The translation may have been between a particular encoding, such as EBCDIC or ASCII, and VS APL internal encoding. Normally, the translate table is converted properly by)*MCOPY*. If not, use the *Z̲C̲* table in 1 *UTILITY* and apply the following conversion:

```
NEWTAB←⎕AV[zc⍳OLDTAB]
```

Functions using the *BYTE* option, which do not meet any of the above criteria must be modified.

- As a short-term solution, the *BYTE* option can be changed to *COD*1.
- An efficient long-term solution is to convert the files and option to *EBCD*.

AP 111 (QSAM Processor): This processor can produce the same problems described for AP 110. Most options provided for AP 110 are also provided for AP 111. In the CMS environment, AP 111 was changed to enforce the rule that the block size for variable length records must be at least 8 bytes larger than the largest record. This change was made because of a change in CMS/SP2.

AP 120 (Session Manager Command Processor): Functions that used AP 120 under VS APL continue to work under APL2.

AP 121 (APL File Processor): This processor allows files to be completely compatible between VS APL and APL2, provided that files continue to be processed in the same way under APL2.

To write nested or mixed arrays, complex numbers, or extended character data to an AP 121 file, use service requests *SWC* or *DUC*. These new service requests allow any valid APL2 data to be written to an AP 121 file.

AP 123 (VSAM Processor): This processor transfers VSAM keys and data without translation. Two problem situations can occur:

- Under VS APL, a function can use AP 123 and no other functions to write a file in Z-codes. Under APL2, the function can access that file through AP 123.

However, the keys no longer identify the correct records and data is incomprehensible. Change the APL 2 function to use the T1 option, which can access data in Z-code format.

An alternative and more permanent resolution of this problem is to do a one-time conversion of the file to EBCDIC. After the data is converted, the function can run under APL2 without change. See *APL2/370 Programming: System Services Reference* for more information on the translation options for AP 123.

- Under VS APL, a function can use AP 123 with the ECO and ECI functions in the workspaces distributed with VS APL. These functions allow AP 123 to write or read data in EBCDIC format.

  Under APL2, these functions are no longer needed to translate from Z-codes to EBCDIC. Copy the new ECO and ECI functions into your workspace; they are provided with the APL2 Licensed Program in the workspace UTILITY. ECO and ECI do no translation. To increase the efficiency of a program, consider eventually removing calls to ECO and ECI from your applications.

  Note: Files **not** containing character data can be read from and written to without change.

AP 126 (GDDM Processor): When a VS APL workspace is transferred to APL2, `)MCOPY` translates all character data in the workspace from Z-codes to EBCDIC. This data is compatible with default conditions under APL2. (The default 4 service request is off.) APL2 character data is already written internally in EBCDIC.

However, some character data in a workspace transferred from VS APL by `)MCOPY` cannot be properly converted to EBCDIC. Under VS APL, AP 126, the symbol set values, and translate table values are usually in EBCDIC encoding. Such data is scrambled by `)MCOPY` when the workspace is transferred.

You can correct or circumvent this problem by one of the following methods:

- Save the symbol sets on auxiliary storage while you are under VS APL. Transfer the workspace. Then, read in the symbol sets to replace the scrambled data.
- Back-translate the scrambled tables, using the `zc` table in `1 UTILITY`:

  `GOODTAB←zc[⎕AV⍳BADTAB]`
- Change your application to use the AP 126 `¯4` service request, which has the following *trans values*:

  0 Special EBCDIC translation off (APL2 default)
  1 Special EBCDIC translation on (VS APL default)

  A trans value of `1` (instead of the APL2 default) enables you to compensate for the scrambled symbol sets.

AP 210 (BDAM Processor): This processor can have translation problems similar to those of AP 110. Refer to the explanation for AP 110.

Checking Alternate Input

If you use AP 101, the alternate input (stack) auxiliary processor, two problems can occur during migration:

- Use of *)COPY*, *)PCOPY*, or *)ERASE* to copy or erase groups
- Use of an inappropriate translation option

Groups: APL2 does not support groups. Instead, the *)COPY*, *)PCOPY*, and *)ERASE* commands are extended to process objects indirectly. APL2 replaces groups with character matrixes that contain the names of objects in the group. If you use the *)MCOPY* command to transfer workspaces, any groups that exist under VS APL are defined by character matrixes in your workspaces under APL2. (See Figure 5 on page 8 for more information on migrating a group of objects from VS APL to APL2.) Check stack commands for group dependencies and change them for indirect copying or erasing. For example, the following VS APL expression copies objects in the group *MYGROUP*:

```
      )COPY MISSIONS MYGROUP
```

Under APL2, groups of objects are copied by enclosing the name of the character matrix in parentheses:

```
      )COPY MISSIONS (MYGROUP)
```

For more information on indirect copying, see *APL2 Programming: Language Reference*.

Translation Option: In APL2, AP 101 defaults to EBCD–no translation. The 192 option from VS APL is equivalent and is permitted. The APL and 370 options are not supported, and are rejected with return code 1. In most cases, the APL option can simply be removed. If the function was using the 370 option under CMS, check especially for dependencies on the following character mappings:

APL characters	∧	¨	÷	∆	≠	×
Passed to CMS	&	"	%	#	$	@

Checking User-Written Auxiliary Processors Used by a Function

The APL2 shared variable processor (SVP) allows user-written auxiliary processors to be compatible with APL2. They do not need to be reassembled unless:

- You want the user-written auxiliary processors to handle the new APL2 data types.
- You want the user-written auxiliary processors to handle the new return code. (ASVPREF and ASVPCPY return code is 12-68 if data cannot be represented in VS APL format.)

- The user-written auxiliary processors access VS APL internal tables, workspaces, or executor control blocks. The user-written auxiliary processors need to be modified because the internal structure and workspace formats are different under APL2.

Under TSO, module AP2TASVP replaces module APLYURVC. In CMS, module AP2VASVP replaces ASVPSRVC. In both environments, user-written auxiliary processors must be link-edited again for use with APL2. For more information, see *APL2/370 Installation and Customization under TSO* or *APL2/370 Installation and Customization under CMS*.

APL2/370 Programming: Processor Interface Reference discusses modifying user-written auxiliary processors to handle APL2 data.

Note: The Shared Variable Processor (SVP) does not support certain forms of the query request (ASVPQRY) macro instruction. It only supports queries of a single variable or a single processor. Queries of multiple variables or multiple partners must be accomplished through multiple query requests.

Testing the Application As a Whole

After you have checked the functions in the workspace, and the files and auxiliary processors they use, you are ready to test the application as a whole. You may find that some runtime instructions need to be modified. For example, if groups of functions or variables are copied from other workspaces, the directions for copying groups must be changed to specify indirect copying.

When you are satisfied that the application is running properly in production, you can drop the unmodified backup workspaces.

Performance Analysis of the Application

Once you have verified that the entire application is operating properly, you may find that the application can be significantly improved by conducting performance analysis.

The APL2 external function *TIME* can be used to gather performance statistics and identify “hotspots” in the application that are using most of the CPU time. By focusing your effort on these hotspots, you can have the greatest effect in reducing overall application CPU time.

Recoding hotspots either using more efficient APL2 algorithms, exploiting APL2's nested array processing capabilities to avoid sequential code with a high degree of interpretive overhead, or even rewriting sections in compiled languages are also techniques that can be used to improve performance sensitive code.

The *TIME* function is discussed in greater detail in *APL2/370 Programming: Using the Supplied Routines*.

Chapter 5. Migration within APL2

This chapter provides a summary of changes in migrating to APL2 Version 2 from other APL2 environments.

Migrating between Mainframe APL2 Systems

This section discusses migration between mainframe systems.

Functional Changes—Version 1 to Version 2

This section discusses the functional changes in Version 2 that can affect migration from Version 1.

There are no major incompatibilities, but minor changes in the behavior of messages, ⎕*NLT*,)*COPY*, CASE, and error signalling may need to be considered.

Mixed-Case and National Language Support

The product is now shipped with a default of mixed-case messages. This is implemented as a “national language” called DEFAULT. APL2 language-defined messages that can appear in ⎕*EM* have been left in uppercase, so in most cases APL applications should be unaffected by the change. However applications that inspect or set ⎕*NLT* may behave differently.

- In Version 1, if ⎕*NLT* was set to an unsupported value the system reset it to be empty, which indicates that uppercase American English is to be used. If it is set to an unsupported value in Version 2, the system resets it to its last valid value. In particular, programs or users may have previously set ⎕*NLT*←'*ENGLISH*' to request English messages. The name '*ENGLISH*' never has been defined as a part of the APL2 product, and this assignment no longer has any effect. You are left with the previous value of ⎕*NLT* rather than having it reset to empty.
- It was previously possible for a user or program to modify a national language table (an APL2LANG file), then respecify ⎕*NLT* to have it take effect. In order to improve performance, the system no longer physically reads an APL2LANG file unless ⎕*NLT* is changed to point to a different file from the one currently active. The recommended procedure for using a newly-changed language file is to first set ⎕*NLT*←'' and then set it to the desired name.
- APL2 now supports IBM's standardized set of three-character language codes as synonyms for the spelled out names it previously recognized. If an installation or user had defined a private language file with a three-character name, the system no longer honors that name if it conflicts with any standard IBM code. This is true whether or not a language file is available for that code. If you want to provide a file for some additional language, and want to access it by its three-character code, define it using the fully-spelled out form of the language name. (See ⎕*NLT* in *APL2 Programming: Language Reference* for details.)
- Since APL2 recognizes the three-character codes as synonyms for the full names, it replaces the code with the full name in ⎕*NLT*. This could confuse an application that set ⎕*NLT* to the code value and then checked it to see if the value was accepted.

Copying System Variables

The)*COPY* command, when used to copy an entire workspace, has been changed to include the system variables ⎕*CT*, ⎕*FC*, ⎕*IO*, ⎕*LX*, ⎕*PP*, ⎕*PR*, and ⎕*RL* along with all user objects in the workspace. This makes its behavior compatible with APL2 workstation products, but different from that of previous mainframe APL systems. The change is a clear improvement for users who are trying to clean up and compress workspaces, or who want to change the workspace CASE. But it can create surprises when merging two workspaces into one. The recommended approaches to combining applications include use of namespaces or of indirect copy lists.

Workspace CASE Attribute

The lowercase alphabet has replaced the underbarred alphabet for use in APL names. As in APL2 Version 1 each workspace has an associated case attribute that controls the format used to enter and display APL names. The Version 1 default was CASE(0), but that now has been changed. Workspaces distributed with the Version 2 product are in CASE(1), and that is also the product default as distributed for newly-created user workspaces. (This default may be overridden by the installation or as an invocation option.) The definitions of supported cases remain as before:

CASE(0)	Lowercase and underbarred characters can be used interchangeably when entering names into the system, though underbarred characters are converted to lowercase internally. Primitives that return names as results (that is, ⎕*NL*, ⎕*CR*, ⎕*FX*, ⎕*SVQ*, and ⎕*TF*), and system commands and messages that display names, convert lowercase letters to underbarred letters for their output.
CASE(1)	Lowercase and underbarred characters can be used interchangeably when entering names into the system, though underbarred characters are converted to lowercase internally. Primitives that return names as results, and system commands and messages that display names, do no conversion of lowercase letters for their output.
CASE(2)	Underbarred characters are treated as invalid in names and are not accepted or produced in system functions, commands, or messages.

The invocation option CASE(*n*) determines the convention to be used for new workspaces created during the APL2 session. This option does not, however, apply to all work done by the user during that session. It is instead interpreted as an implicit parameter to all subsequent)*CLEAR* commands. Note that)*CLEAR* is the only way in APL to create a new workspace.

The CASE attribute assigned to a workspace during)*CLEAR* cannot be changed later. The only way to change the CASE attribute of a workspace is to transfer its contents to a different workspace. Using)*COPY*,)*PCOPY*, or)*IN* does not affect the CASE attribute of the workspace into which the objects are copied. A *CASE* function has been provided in the *UTILITY* workspace that returns the case of the active workspace.

When in a CASE(0) or CASE(1) workspace,)*IN* can be used to access objects with underbarred names or containing references to other objects with underbarred names. The names of the copied objects, as well as names referred to by copied functions or defined operators, are converted to lowercase as appropriate. Note, however, that literal strings and comments within functions, and the content of variables, are not converted.

Caution: *)IN* should not be used in a CASE(2) workspace to access a transfer file written from a CASE(0) workspace. Attempts to do so can fail because names in transfer files created from CASE(0) workspaces frequently contain underbarred letters, and CASE(2) does not convert underbarred letters to lowercase. CASE(1) provides a bridge between CASE(0) and CASE(2) in this context.

)COPY and *)PCOPY* can be used with no problem between CASE(0) and CASE(2) workspaces, because names in APL2 workspaces never actually contain underbarred letters internally.

Migrating Workspaces

This section discusses how you can migrate your workspaces.

Version 1 to Version 2

Migration of workspaces from APL2 Version 1 to Version 2 is automatic on any *)LOAD* or *)COPY* of the workspace. A message is issued indicating that the internal conversion has been done, and if the workspace is then saved, it is a Version 2 workspace. If the workspace was loaded, it has the CASE it had when it was saved under the previous release. If it was copied, it has the case of the active workspace at the time of the *)COPY*.

Version 2 to Version 1

Because of internal changes to the structure of the workspace, Version 2 workspaces cannot be loaded or copied into Version 1. If a *)LOAD* or *)COPY* is attempted, the message *WS INVALID* is issued.

To migrate a workspace backward from Version 2 to Version 1, use *)OUT* in Version 2 to create a transfer form file, and use *)IN* in Version 1 to receive it.

Workspaces that take advantage of new Version 2 features, of course, do not migrate directly to Version 1. For example, existing external associations with new Processor 11 external functions or with Processor 12 files cause errors during the *)IN*, and uses of auxiliary processors 119 or 211 cause execution-time errors.

Version 2 Release 1 to Version 2 Release 2

The workspace formats of the two releases of Version 2 are compatible, so workspaces can be transferred between the releases in either direction with *)LOAD* and *)SAVE*.

If the new diamond statement separator has been used in a Version 2 Release 2 workspace, errors occur when functions containing the diamond are executed in Version 2 Release 1.

Coexistence with Version 1

This section discusses how to coexist with the previous version of APL2.

Shared Variable Processor Considerations

APL2 provides an optional Global Shared Variable Processor (GSVP) for connections between APL2 sessions on the same system, and between an APL2 session and a server written as a global auxiliary processor. The GSVP provided with APL2 Version 2 can be used by the previous version of APL2, but APL2 Version 2 cannot use earlier versions of the GSVP.

Migrating between Mainframe and Workstations

This section discusses migration between the mainframe and workstation environments, and between the different workstation environments.

Transferring Workspaces

Workspaces are easily transferred between APL2 systems. Transfer file formats have been defined to permit exchange of workspace objects among all IBM APL2 implementations.

Workspace Transfer between APL2 Systems

In general, APL2 workspaces must be sent to other APL2 systems as *transfer form* files. Transfer forms have the following default file naming conventions:

CMS	filename APLTF *
TSO	prefix.APLTF.filename
OS/2 or DOS	path\filename.ATF
AIX* or UNIX**	path/filename.atf

The APL2 commands used to create and read transfer form files are)*OUT*,)*IN*, and)*PIN*. To transfer a workspace, start APL2 on the system where the workspace resides, and issue the following commands:

```
)LOAD  wsid
)SIC   (or )RESET)
)OUT  filename
```

A transfer file is created by the)*OUT* command.

Once the transfer file is created, it then must be moved to the target APL2 system, and can be saved with a name following the conventions of the target system. The techniques for physically moving files from one system to another can vary depending on the types of systems and what connections exist between them.

- One key issue is that some systems (for example MVS/TSO and VM/CMS) use an EBCDIC character encoding, while others (for example OS/2 and AIX/6000) use an ASCII encoding. Both ASCII and EBCDIC transfer file formats are defined, and all IBM APL2 systems accept both formats. No data conversion should be attempted within the file itself when transferring it from one system to another. The receiving APL2 system performs any necessary conversion. If the transfer is done electronically through a network connection, the programs controlling that transfer must be told that this is a “binary” rather than “character” file. (The exact terminology used may vary depending on the system and network control programs being used.)
- Some systems use “record-oriented” files while others use stream files. If stream files are being transferred to a system that expects record-oriented files,

an arbitrary record length may be used, but the existing record separators ("LF" or "CR/LF") must be retained. Conversely, separators should not be inserted when record-oriented files are being transferred to a system that expects stream files. Again, the receiving APL2 system adjusts to these differences.

- Within these constraints, standard data transmission commands appropriate to the system such as "ftp put," "SEND," "SENDFILE," or "TRANSMIT" can be used for network transmission, with corresponding commands such as "ftp get" or "RECEIVE" as appropriate to the receiving system.
- Because the receiving APL2 system performs all necessary conversions, it is also possible to use shared DASD, remote file systems, removable media, or other such facilities to transport the data.

When the file has been transferred to the target system, it can then be read into APL2 and saved as a workspace:

```
      )CLEAR
      )IN  filename
      )SAVE  wsid
```

Migration of TryAPL2 Workspaces

Workspaces saved under TryAPL2 can be read by APL2/2, APL2/6000, and APL2 for Sun Solaris. The function *TRYLOAD* in the FILE workspace can be used to read these files. Once migrated to one of the workstations, the)*OUT* and)*IN* processes can be used to migrate to the mainframe.

Transferring AP 211 Files

Files created by AP 211 are portable between APL2/370, APL2/2, APL2/6000, and APL2 for Sun Solaris. The files must be transferred in binary mode. The receiving APL2 system performs all necessary conversions of data. Files to be uploaded to the mainframe must be uploaded as fixed format files, with a record length equal to the AP 211 blocksize for the file. The blocksize can be obtained by issuing an AP 211 'USE' command against the file.

In addition, files created by AP 211 on APL2/PC can be read by APL2/2, APL2/6000, and APL2 for Sun Solaris. Writing back to these files is not allowed. The function *REBUILD*211 in the public workspace 2 FILE can be used to permanently convert the APL2/PC file to the new format if desired. Once migrated to one of the workstations, the file can then be uploaded to the mainframe.

Bibliography

APL2 Publications

- *APL2 Fact Sheet,* GH21-1090
- *APL2/370 Application Environment Licensed Program Specifications,* GH21-1063
- *APL2/370 Licensed Program Specifications,* GH21-1070
- *APL2 for AIX/6000 Licensed Program Specifications,* GC23-3058
- *APL2 for Sun Solaris Licensed Program Specifications,* GC26-3359
- *APL2/370 Installation and Customization under CMS,* SH21-1062
- *APL2/370 Installation and Customization under TSO,* SH21-1055
- *APL2 Migration Guide,* SH21-1069
- *APL2 Programming: Language Reference,* SH21-1061
- *APL2/370 Programming: Processor Interface Reference,* SH21-1058
- *APL2 Reference Summary,* SX26-3999
- *APL2 Programming: An Introduction to APL2,* SH21-1073
- *APL2 for AIX/6000: User's Guide,* SC23-3051
- *APL2 for OS/2: User's Guide,* SH21-1091
- *APL2 for Sun Solaris: User's Guide,* SH21-1092
- *APL2 for the IBM PC: User's Guide,* SC33-0600
- *APL2 GRAPHPAK: User's Guide and Reference,* SH21-1074
- *APL2 Programming: Using Structured Query Language,* SH21-1057
- *APL2/370 Programming: Using the Supplied Routines,* SH21-1056
- *APL2/370 Programming: System Services Reference,* SH21-1054
- *APL2/370 Diagnosis Guide,* LY27-9601
- *APL2/370 Messages and Codes,* SH21-1059

Other Books You Might Need

The following book is recommended:

- *APL2 at a Glance*, by James Brown, Sandra Pakin, and Raymond Polivka, published by Prentice-Hall, ISBN 0-13-038670-7 (1988). (Copies can be ordered from IBM as SC26-4676.)

Plastic replacement keyboard keycaps are included with this product. Additional sets of keyboard keycaps are available from IBM as:

- *APL2 Keycaps (US and UK base set)*, SX80-0270
- *APL2 Keycaps, German upgrade to SX80-0270*, SX23-0452
- *APL2 Keycaps, French upgrade to SX80-0270*, SX23-0453
- *APL2 Keycaps, Italian upgrade to SX80-0270*, SX23-0454.

Two sets of *APL2 Keyboard Decals*, SC33-0604, are included with this product. Additional sets of these decal sheets can be ordered from IBM.

Index

We'd Like to Hear from You

APL2
Migration Guide
Version 2 Release 2

Publication No. SH21-1069-01

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: (408) 463-4488.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIB6JN8
  - Internet: apl2@vnet.ibm.com

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

APL2
Migration Guide
Version 2 Release 2

Publication No. SH21-1069-01

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments
SH21-1069-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department M46/D12
PO Box 49023
San Jose, CA 95161-9023

Fold and Tape **Please do not staple** Fold and Tape

SH21-1069-01

Cut or Fold
Along Line

File Number: S370-34
Program Number: 5688-228
5688-229

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

The APL2 Library

GH21-1090	APL2 Family of Products (fact sheet)
SH21-1073	APL2 Programming: An Introduction to APL2
SH21-1061	APL2 Programming: Language Reference
SX26-3999	APL2 Reference Summary
SH21-1074	APL2 GRAPHPAK: User's Guide and Reference
SH21-1057	APL2 Programming: Using Structured Query Language
SH21-1069	APL2 Migration Guide
SC33-0600	APL2 for the IBM PC: User's Guide
SC33-0601	APL2 for the IBM PC: Reference Summary
SC33-0851	APL2 for the IBM PC: Reference Card
SH21-1091	APL2 for OS/2: User's Guide
GC23-3058	APL2 for AIX/6000 Licensed Program Specifications
SC23-3051	APL2 for AIX/6000: User's Guide
GC26-3359	APL2 for Sun Solaris Licensed Program Specifications
SH21-1092	APL2 for Sun Solaris: User's Guide
GH21-1063	APL2/370 Application Environment Licensed Program Specifications
GH21-1070	APL2/370 Licensed Program Specifications
SH21-1062	APL2/370 Installation and Customization under CMS
SH21-1055	APL2/370 Installation and Customization under TSO
SH21-1054	APL2/370 Programming: System Services Reference
SH21-1056	APL2/370 Programming: Using the Supplied Routines
SH21-1058	APL2/370 Programming: Processor Interface Reference
LY27-9601	APL2/370 Diagnosis Guide
SH21-1059	APL2/370 Messages and Codes

SH21-1069-01