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.

© Copyright International Business Machines Corporation 1984, 1994. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user’s responsibility.

IBM may have patents or pending patent applications covering subject material in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Corporation, IBM Director of Licensing, 208 Harbor Drive, Stamford, Connecticut, United States 06904.

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

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- AIX/6000
- APL2
- APL2/6000
- DB2
- GDDM
- IBM
- MVS/ESA
- OS/2
- SQL/DS
- System/370
- System/390

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies:

- Sun
- Solaris
- UNIX
- Sun Microsystems, Inc.
- Sun Microsystems, Inc.
- AT&T Corporation
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.

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

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL2/2</td>
<td>OS/2*</td>
</tr>
<tr>
<td>APL2 for Sun Solaris</td>
<td>Sun** Solaris**</td>
</tr>
<tr>
<td>APL2/6000*</td>
<td>AIX/6000*</td>
</tr>
<tr>
<td>APL2/370</td>
<td>MVS or VM</td>
</tr>
<tr>
<td>APL2/PC</td>
<td>DOS</td>
</tr>
</tbody>
</table>
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 /quadAV dependencies, as described in "Adjusting for /quadAV 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 )s, )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:
   \[ IX + AV \cdot CX \]
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:
   \[ CX \leftarrow AV [IX] \]
   **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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Your Entry under APL2</th>
<th>Explanation and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>)CLEAR</td>
<td>Clear the active workspace (sets CASE attribute).</td>
</tr>
</tbody>
</table>
Figure 2 (Page 2 of 2). Steps for Transferring a Workspace from VS APL to APL2

<table>
<thead>
<tr>
<th>Step</th>
<th>Your Entry under APL2</th>
<th>Explanation and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>)MCOPY [libno] wsname</td>
<td>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 &quot;Transferring Selected Objects&quot; on page 9.</td>
</tr>
<tr>
<td>3</td>
<td>)SAVE [libno] wsname</td>
<td>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.</td>
</tr>
</tbody>
</table>

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](image)

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.

Syntax of the \texttt{MCOPY} Command: Figure 4 shows the syntax of the \texttt{MCOPY} command.

\begin{verbatim}
\texttt{MCOPY [libno] wsname [:[password]] [names]}
\end{verbatim}

\textit{Figure 4. Syntax of the \texttt{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 \texttt{MCOPY} command:

\texttt{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 \texttt{MCOPY} command:

\texttt{MCOPY PASSAGES:HUSH}

Effects of the \texttt{MCOPY} Command: When you issue the \texttt{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 \texttt{/quadLX}, \texttt{/quadIO}, \texttt{/quadRL}, \texttt{/quadCT}, and \texttt{/quadPP} 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 \texttt{MCOPY} command.

\textbf{Note:} Only the five system variables \texttt{/quadLX}, \texttt{/quadIO}, \texttt{/quadRL}, \texttt{/quadCT}, and \texttt{/quadPP} can be copied. An attempt to copy other system variables (for example, \texttt{/quadPW}) results in a \texttt{NOT FOUND} message.

- Locked functions in a VS APL workspace are copied and remain locked. Under APL2, their execution properties are \texttt{1 1 1 1}, as shown below for the locked function named \texttt{SEARCH}:

  \begin{verbatim}
  3 /quadAT /quoteSEARCH/quote
  1 1 1 1
  \end{verbatim}

  For information on function locking and attributes under APL2, see \textit{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 \texttt{4 5 6[3]} under VS APL becomes \texttt{(4 5 6)[3]} under APL2.

\textbf{Note:} \texttt{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 \texttt{AP 101}), transferred into a function (using \texttt{AP 101}), or stacked using \texttt{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.

![Figure 5. Migrating a Group of Objects from VS APL to APL2](image)

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

**Under CMS**

The \texttt{\textasciitilde MCOPY} command uses the file APLIBTAB—not the file LIBTAB APL2.

If no APLIBTAB file is found, a dummy one is created. This dummy file allows you to access only the library with the number \texttt{\textasciitilde A[1]}.

**Under TSO**

The \texttt{\textasciitilde 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 \texttt{\textasciitilde MCOPY} Command:** If your VS APL installation has changed parameters in the options module that you want carried forward by \texttt{\textasciitilde 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 \texttt{\textasciitilde W}), PUBQLFR (from \texttt{\textasciitilde PL}), or LIBQLFR (from \texttt{\textasciitilde }). 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.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPROPER LIBRARY REFERENCE</td>
<td>The specified library is inaccessible or does not exist.</td>
</tr>
<tr>
<td>LIBRARY I/O ERROR</td>
<td>An internal error is preventing the successful copying of the workspace.</td>
</tr>
<tr>
<td>LIBRARY NOT AVAILABLE</td>
<td>Another user temporarily has control of a shared library. This prevents making a successful copy of the workspace at this time.</td>
</tr>
<tr>
<td>NOT COPIED:</td>
<td>The listed objects cannot be copied because insufficient space is available for copying the objects. The names of objects not copied are listed.</td>
</tr>
<tr>
<td>NOT FOUND:</td>
<td>The named objects do not exist or cannot be copied from the specified workspace.</td>
</tr>
<tr>
<td>SYSTEM LIMIT</td>
<td>This message can indicate any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Your virtual machine size or region size is not large enough. Increase the size.</td>
</tr>
<tr>
<td></td>
<td>• Your workspace size is not large enough to load the specified workspace. Change that size by using the WSSIZE option when APL2 is started.</td>
</tr>
<tr>
<td></td>
<td>• The freespace size is not large enough. Change that size by using the FREESIZE option when APL2 is started.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>WS FULL</td>
<td>Sufficient space is not available for copying all or some of the objects specified with the <code>MCOPY</code> command. When possible, <code>MCOPY</code> lists the names of objects not copied.</td>
</tr>
<tr>
<td>WS LOCKED</td>
<td>The password specified with the <code>MCOPY</code> command is incorrect, or a necessary password is missing from the command.</td>
</tr>
<tr>
<td>WS NOT FOUND</td>
<td>The specified VS APL workspace does not exist in the specified VS APL library.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Category</th>
<th>New Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Types and Structures Allowed</td>
<td><strong>Complex numbers</strong>: Numbers can have real and imaginary parts.</td>
</tr>
<tr>
<td></td>
<td><strong>Mixed data</strong>: Both characters and numbers can be in the same array.</td>
</tr>
<tr>
<td></td>
<td><strong>Nested arrays</strong>: An array can contain another array.</td>
</tr>
<tr>
<td>Primitive Functions (Boolean and</td>
<td>Match ($L \equiv R$)</td>
</tr>
<tr>
<td>Relational)</td>
<td></td>
</tr>
<tr>
<td>Primitive Functions (Structural)</td>
<td>Depth ($\equiv R$)</td>
</tr>
<tr>
<td></td>
<td>Enclose ($\in R$)</td>
</tr>
<tr>
<td></td>
<td>Enclose with Axis ($\in [X] R$)</td>
</tr>
<tr>
<td></td>
<td>Disclose ($\geq R$)</td>
</tr>
<tr>
<td></td>
<td>Disclose with Axis ($\geq [X] R$)</td>
</tr>
<tr>
<td></td>
<td>Ravel with Axis ($, [X] R$)</td>
</tr>
<tr>
<td></td>
<td>Enlist ($\epsilon R$)</td>
</tr>
<tr>
<td>Primitive Functions (Selection)</td>
<td>First ($+ R$)</td>
</tr>
<tr>
<td></td>
<td>Pick ($L \rightarrow R$)</td>
</tr>
<tr>
<td></td>
<td>Take with Axis ($L + [X] R$)</td>
</tr>
<tr>
<td></td>
<td>Drop with Axis ($L + [X] R$)</td>
</tr>
<tr>
<td></td>
<td>Without ($L \sim R$)</td>
</tr>
<tr>
<td></td>
<td>Index ($L [] R$)</td>
</tr>
<tr>
<td></td>
<td>Index with Axis ($L [] [X] R$)</td>
</tr>
<tr>
<td>Primitive Functions (Position)</td>
<td>Find ($L \in R$)</td>
</tr>
<tr>
<td></td>
<td>Dyadic Grade Up ($L \uparrow R$)</td>
</tr>
<tr>
<td></td>
<td>Dyadic Grade Down ($L \downarrow R$)</td>
</tr>
<tr>
<td></td>
<td>Partition ($L \leftarrow R$)</td>
</tr>
<tr>
<td></td>
<td>Partition with Axis ($L \leftarrow [X] R$)</td>
</tr>
<tr>
<td>Primitive Operators</td>
<td>Each ($L O^{+}$)</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1984, 1994
### New Language Features under APL2

<table>
<thead>
<tr>
<th>Category</th>
<th>New Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined Operators</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 $L O / R$  
  - N-wise Reduce $L L O / R$ |
| Separators                       | Diamond (•)                                                                 |
| System Functions and Variables   | Event Message $EM$  
  Event Simulation $ES$  
  Event Type $ET$  
  Execute (Process) Alternate $EA$  
  Execute (Process) Controlled $EC$  
  Left Argument $L$  
  Right Argument $R$ |
| (Event Handling and Debugging)   | 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. |
| System Functions and Variables   | Shared Variable Event $SVE$  
  Shared Variable State $SVS$ |
| (Sharing)                        | 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 2 of 3). New Language Features under APL2
## 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*.

### 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*.

#### 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.
### New System Features under APL2

<table>
<thead>
<tr>
<th>Category</th>
<th>New Feature</th>
</tr>
</thead>
</table>
| 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 `/quadNA`. 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.

<table>
<thead>
<tr>
<th>Data Representation under VS APL</th>
<th>Data Representation under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>An array can contain only character or only numeric data. Each item in an array must be a single number or a single character.</td>
<td>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).</td>
</tr>
<tr>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>[ R \leftarrow 9 \ 13 \ 48 \ 27 ] [ L \leftarrow 'SHOE' ]</td>
<td>[ OBJNAME \leftarrow 4 \ 6 \ 12 \ 4 ] [ R \leftarrow 9 \ 13 \ 'S' \ 'SHOE' \ OBJNAME ]</td>
</tr>
<tr>
<td>Assignment can be to only a single name.</td>
<td>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.</td>
</tr>
<tr>
<td>[ A \leftarrow 2 ] [ B \leftarrow 2 ] [ C \leftarrow 'ANC' ] [ D \leftarrow 2 \ 3 \ 4 ] [ E \leftarrow 2 \ 3 \ 4 ] [ F \leftarrow 2 \ 3 \ 4 ]</td>
<td>( (A \ B \ C) \leftarrow 2 \ 3 \ 'ANC' ) [ A ] [ 2 \ \rho C ] [ 3 ] [ (D \ E \ F) \leftarrow 2 \ 3 \ 4 ] [ D ] [ 2 \ 3 \ \rho E ] [ 3 ]</td>
</tr>
</tbody>
</table>
### Figure 10. VS APL Compared with APL2—Evaluation of Expressions (Compatible)

<table>
<thead>
<tr>
<th>Evaluation of Expressions under VS APL</th>
<th>Evaluation of Expressions under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses are used for grouping an expression to control the order of evaluation.</td>
<td>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.</td>
</tr>
<tr>
<td>2×3+5</td>
<td>Correct: (2 4 5) 3</td>
</tr>
<tr>
<td>16</td>
<td>2 4 5 3</td>
</tr>
<tr>
<td>(2×3)+5</td>
<td>Correct, but parentheses around ⋅+ are redundant:</td>
</tr>
<tr>
<td>11</td>
<td>(2 3ρι6) ⋅+ 3 2ρ−16</td>
</tr>
<tr>
<td></td>
<td>&quot;22 -28&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;49 -64&quot;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Display of Output under VS APL</th>
<th>Display of Output under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>In numeric output, multiple spaces on a line are compressed to a single space because they are redundant.</td>
<td>In numeric output, multiple spaces can display on a line to indicate the structure of items in a nested array.</td>
</tr>
<tr>
<td>4 6 9 2</td>
<td>1 2 (3 4)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Object Names under VS APL</th>
<th>Object Names under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>The characters ⍴ and ⍾ are not valid in object names.</td>
<td>The characters ⍴ and ⍾ are valid in object names, provided they are not the first character.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Primitive Functions under VS APL</th>
<th>Primitive Functions under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression (L/R) is a primitive function that requires a Boolean left argument.</td>
<td>Replicate (L/R) is a derived function. Its left operand is a vector of Boolean, negative, or positive integers.</td>
</tr>
<tr>
<td>SHE</td>
<td>1 0 3 2/'SHOE'</td>
</tr>
<tr>
<td></td>
<td>SOOEE</td>
</tr>
<tr>
<td>With a Boolean left argument, the derived function is called compress and is similar to VS APL compression:</td>
<td></td>
</tr>
<tr>
<td>SHE</td>
<td>1 1 0 1/'SHOE'</td>
</tr>
<tr>
<td>Expansion (L\R) is a primitive function that requires a Boolean left argument.</td>
<td>Expand (L\R) is a derived function. It is similar to the VS APL expansion. Its left operand is a simple Boolean vector.</td>
</tr>
<tr>
<td>SHE</td>
<td>1 1 0 1/'SHOE'</td>
</tr>
<tr>
<td>S W</td>
<td>1 0 1 2 2ρ'SWIM'</td>
</tr>
<tr>
<td>I M</td>
<td>1 0 1 (2 3)(4 5)</td>
</tr>
<tr>
<td></td>
<td>2 3 0 0 4 5</td>
</tr>
</tbody>
</table>
**Figure 13.** VS APL Compared with APL2—Primitive Functions (Compatible)

<table>
<thead>
<tr>
<th>Primitive Functions under VS APL</th>
<th>Primitive Functions under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format (L ⌽ R) requires a numeric left argument.</td>
<td>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.</td>
</tr>
<tr>
<td>55@ $53.50 EA</td>
<td>32 9.17 EA</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Selective Specification under VS APL</th>
<th>Selective Specification under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bracket Indexing (A[I]) is the only primitive function that allows you to assign a value to selected items in an array.</td>
<td>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.</td>
</tr>
<tr>
<td>A←23 44 97</td>
<td>A←2 1 5</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>23 80 97</td>
<td>1 2 0 5</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Primitive Operators under VS APL</th>
<th>Primitive Operators under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived functions can be created by applying a primitive operator to a primitive function.</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>System Functions and Variables under VS APL</th>
<th>System Functions and Variables under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name List (NL) accepts as its argument any integer 1 through 3.</td>
<td>Name List (NL) accepts as its argument any integer 1 through 4 (with 4 meaning a defined operator).</td>
</tr>
<tr>
<td>Printing Precision (PP) can be as great as 16.</td>
<td>Printing Precision (PP) can be as great as 18.</td>
</tr>
</tbody>
</table>
### Figure 17. VS APL Compared with APL2—Debugging and Processing (Compatible)

<table>
<thead>
<tr>
<th>Debugging and Processing under VS APL</th>
<th>Debugging and Processing under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace (T_{\delta}) and Stop (S_{\delta}) controls cannot be referenced.</td>
<td>Interrupted processing of a statement can be resumed at the point where the processing has halted by using (\rightarrow 10).</td>
</tr>
<tr>
<td>Interrupted processing of statements entered in immediate execution mode cannot be resumed.</td>
<td>The state indicator, listed by the (\text{}SI\text{)}) command, contains a list of the calling sequence of defined functions (with their pertinent line numbers) that led to the current state.</td>
</tr>
<tr>
<td>To indicate suspended immediate processing statements, the state indicator includes an asterisk (\ast) on the line containing the name and function line number of the first function called.</td>
<td>As with VS APL, the state indicator, listed by the (\text{}SI\text{)}) command, contains a list of the calling sequence of defined functions and defined operators that led to the current state.</td>
</tr>
<tr>
<td>The state indicator, listed by the (\text{}SI\text{)}) command, contains a list of the calling sequence of defined functions (with their pertinent line numbers) that led to the current state.</td>
<td>To indicate each suspended immediate execution statement, the state indicator includes an asterisk (\ast) on a line by itself. Asterisks can indicate suspended immediate execution statements that do not call a function.</td>
</tr>
<tr>
<td>The system command (\text{}SI\text{)}), (\text{}SINL\text{)}), and (\text{}SIS\text{)}) commands also allow an optional numeric argument to indicate the number of levels of the state indicator to be displayed, for instance (\text{}SI 4\text{)}).</td>
<td>The system command (\text{}RESET\text{)}) (n) or (\text{}SIC\text{)}) (n) clears (n) lines from the state indicator. (\text{}RESET\text{)}) or (\text{}SIC\text{)}) clears all lines.</td>
</tr>
</tbody>
</table>

The right arrow \(\rightarrow\) clears the most recent line(s) placed in the state indicator.

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

<table>
<thead>
<tr>
<th>System Commands under VS APL</th>
<th>System Commands under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{}FNS\text{)}) and (\text{}VARS\text{)}) allow you to specify the beginning letter or set of letters for listing functions and variables in the active workspace.</td>
<td>(\text{}FNS\text{)}) and (\text{}VARS\text{)}) allow you to specify the beginning and ending letter or set of letters for listing functions and variables in the active workspace.</td>
</tr>
<tr>
<td>(\text{}SYMBOLS\text{)}) allows you to increase the maximum size of the symbol table, but can only be used in an empty workspace.</td>
<td>(\text{}SYMBOLS\text{)}) 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. (\text{}SYMBOLS\text{)}) also causes the unused symbols in a workspace to be removed.</td>
</tr>
<tr>
<td>(\text{}LIB\text{)}) allows you to specify alphabetic letter(s) for beginning the listing of workspace names.</td>
<td>(\text{}LIB\text{)}) allows you to specify alphabetic letters for beginning and ending the listing of workspace names.</td>
</tr>
<tr>
<td>(\text{}OFF[\text{HOLD}]\text{)}) and (\text{}CONTINUE[\text{HOLD}]\text{)}) provide an optional (\text{HOLD}\text{)}) parameter that allows you to return to the host system. If you do not specify the (\text{HOLD}\text{)}) parameter, you are logged off the host system.</td>
<td>(\text{\text{OFF[\text{HOLD}]}\text{)})) and (\text{\text{CONTINUE[\text{HOLD}]}\text{)})) always return control to the host system. The optional (\text{HOLD}\text{)}) parameter has no effect.</td>
</tr>
</tbody>
</table>
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: | APL2 retains the following DEBUG settings from VS APL:
  - msg (1) |  - msg (1)
  - echo (2) |  - echo (2)
  - abend (64) |  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) |  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 \quadLX during \quadLOAD
- **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 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 (\quad, 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)

<table>
<thead>
<tr>
<th>Editing under VS APL</th>
<th>Editing under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>The editor command [n] displays line (n) of the function definition.</td>
<td>The editor command [n] displays line (n) of the function or operator definition. [n] results in a DEFN ERROR.</td>
</tr>
<tr>
<td>The editor command [[n] displays line (n) through the last line of the definition.</td>
<td>The editor command [[n] displays line (n) through the last line of the definition.</td>
</tr>
<tr>
<td>Editing the function name in a definition replaces the current name of the function.</td>
<td>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.</td>
</tr>
<tr>
<td>In VS APL local functions cannot be edited.</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

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

<table>
<thead>
<tr>
<th>Auxiliary Processors Distributed with VS APL</th>
<th>Auxiliary Processors Distributed with APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auxiliary processors distributed with VS APL support only VS APL data types.</td>
<td>AP 110, AP 111, AP 119, AP 121, AP 127, AP 210, and AP 211 support the new APL2 data types.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Changes in default translation and initialization options for auxiliary processors are discussed on page 35.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
## Extended-Incompatible Features

Figure 22 through Figure 27 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)

<table>
<thead>
<tr>
<th>Evaluation of Expressions under VS APL</th>
<th>Evaluation of Expressions under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS APL's evaluation of brackets allows you to select an item from a vector constant.</td>
<td>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:</td>
</tr>
<tr>
<td>4 5 6[3]</td>
<td>4 5 (6[3])</td>
</tr>
<tr>
<td>6</td>
<td>RANK ERROR</td>
</tr>
<tr>
<td></td>
<td>4 5(6[3])</td>
</tr>
</tbody>
</table>

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:

- →/quadLC
- →/quote/quote
- →ι0

A suspended function or operator can be restarted at the suspended line with the following expression:

- →/quadLC

It can be restarted at the point of suspension with either of the following expressions:

- →/quote/quote
- →ι0

Dyadic defined functions cause a syntax error if invoked monadically.

All functions are ambivalent. 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 /quadEC. However, they are honored in a function invoked from /quadEA.

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'.
<table>
<thead>
<tr>
<th>VS APL Default Output</th>
<th>APL2 Default Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default output of arrays with rank (\geq 2) folds the output on a line-by-line basis where the width of (PW) is exceeded.</td>
<td>Default output of arrays with rank (\geq 2) folds the output on a plane-by-plane basis where the width of a plane exceeds (PW).</td>
</tr>
<tr>
<td>Default output of empty arrays always produces one line.</td>
<td>Default output of empty arrays produces as many lines as there are rows in the array.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primitive Functions under VS APL</th>
<th>Primitive Functions under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power</strong> ((L , \ast , 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 <strong>DOMAIN ERROR</strong>.</td>
<td><strong>Power</strong> ((L , \ast , 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.</td>
</tr>
<tr>
<td>(-8 , \ast , 3)</td>
<td>(-8 , \ast , 3)</td>
</tr>
<tr>
<td>(-2)</td>
<td>(1J1.732050808)</td>
</tr>
<tr>
<td>(-8 , \ast , 2)</td>
<td>(-8 , \ast , 2)</td>
</tr>
<tr>
<td><strong>DOMAIN ERROR</strong></td>
<td>0J2.828427125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monadic Format ((\ast , 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.</th>
<th>Format (Default) ((\ast , 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.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dyadic Format</strong> ((L , \ast , 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.</td>
<td>Format by specification ((L , \ast , 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.</td>
</tr>
<tr>
<td><strong>Circle</strong> ((L , \circ , R)) accepts the integers (-7) through (7) as valid left arguments. The result of (-4 , \circ , R) is a positive square root.</td>
<td><strong>Circle</strong> ((L , \circ , R)) accepts the integers (-12) through (12) as valid left arguments. The result of (-4 , \circ , R) is a negative square root if (R) is negative.</td>
</tr>
<tr>
<td>(-4 , \circ , 2)</td>
<td>(-4 , \circ , 2)</td>
</tr>
<tr>
<td>1.732050808</td>
<td>1.732050808</td>
</tr>
<tr>
<td>(-4 , \circ , -2)</td>
<td>(-4 , \circ , -2)</td>
</tr>
<tr>
<td>1.732050808</td>
<td>1.732050808</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Residue</strong> ((L , \div , R)) has no implicit arguments.</th>
<th><strong>Residue</strong> ((L , \div , R)) uses (CT) as an implicit argument.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Residue</strong> ((L , \div , R)) uses (CT) as an implicit argument.</td>
<td><strong>Residue</strong> ((L , \div , R)) uses (CT) as an implicit argument.</td>
</tr>
<tr>
<td>(\Box PP , + , 16)</td>
<td>(\Box PP , + , 16)</td>
</tr>
<tr>
<td>(\Box CT , + , 1E^{-13})</td>
<td>(\Box CT , + , 1E^{-13})</td>
</tr>
<tr>
<td>1.9999999999999999</td>
<td>1.9999999999999999</td>
</tr>
<tr>
<td>0.9999999999999999</td>
<td>0.9999999999999999</td>
</tr>
</tbody>
</table>

| 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

The shape of the arguments for the general case of $P \cdot g \cdot Q$ 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 1p 'A'
A^.= 'AA'
```

Inner Product under APL2

The shape of the arguments for the general case of $P \cdot g \cdot Q$ 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

A character output (\l) assignment immediately followed by a character input prompt (\l) 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.

If the character input prompt (\l) is on a different line from the character output, entering a single-character response results in a scalar.

The Atomic Vector (\AV) uses an ordering unique to VS APL. The alphabet is 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.

Expunge (\EX) does not erase defined functions that are suspended or waiting to complete processing.

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.

Name Class (\NC) returns a class of 0 through 4, with 4 indicating an invalid name for an object.

System Functions and Variables under APL2

A character output (\l) assignment immediately followed by a character input prompt (\l) 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 \overline{PR}←'.
- A repeated character, specified in \overline{PR}, which replaces the prompt portion of the resulting vector. A blank is the default setting of \overline{PR}.

If the character input prompt (\l) 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 that conforms with EBCDIC and is different from the ordering under VS APL. The alphabet is not contiguous.

Canonical Representation (\CR) returns the character representation of a defined function or operator, not limiting the precision of any numeric constants.

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 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 \_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)**

<table>
<thead>
<tr>
<th>System Functions and Variables under VS APL</th>
<th>System Functions and Variables under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>The result of EX, NC, SVO, or SVR applied to a scalar or vector is a scalar.</td>
<td>The result of EX, NC, SVO, or SVR applied to a scalar or vector is a one-item vector.</td>
</tr>
<tr>
<td>The value of WA depends on the internal format of VS APL objects.</td>
<td>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.</td>
</tr>
<tr>
<td>The )COPY command, when used to copy an entire workspace, copies all user objects in the workspace, but not system variables.</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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)**

<table>
<thead>
<tr>
<th>Auxiliary Processor Translation Options under VS APL</th>
<th>Auxiliary Processor Translation Options under APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auxiliary processors distributed with VS APL provide default translation options.</td>
<td>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 &quot;Changes in APL2 Auxiliary Processor Translation Options&quot; on page 35.</td>
</tr>
<tr>
<td>AP 110, AP 111, and AP 210 provide a BYTE option that allows for no translation regardless of the encoding of the file.</td>
<td>The BYTE option works only for some applications that use it under VS APL. See &quot;Changes in APL2 Auxiliary Processor Translation Options&quot; on page 35 for a discussion on files and auxiliary processors.</td>
</tr>
<tr>
<td>AP 101 provides the options 370 and APL.</td>
<td>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.</td>
</tr>
<tr>
<td>AP 101 stacked input cancels the session manager SUPPRESS command.</td>
<td>AP 101 stacked input no longer cancels the session manager SUPPRESS command. Entering the following lines cancels the SUPPRESS command:</td>
</tr>
</tbody>
</table>

```
3 11 NA 'OPTION'
'OFF' OPTION 'QUIET'
```
### Auxiliary Processor Translation Options under VS APL

User-written auxiliary processors do not handle the new APL2 data types.

### Auxiliary Processor Translation Options under APL2

A new return code 1 2 - 6 8 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

ASVPQRY can return information about multiple variables or multiple partners in a single request.

### ASVPQRY

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](image)

<table>
<thead>
<tr>
<th>Dropped from VS APL</th>
<th>Replacement in APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dyadic Shared Variable Query $SVQ</td>
<td>No replacement.</td>
</tr>
<tr>
<td>$GROUP, $GRP, $GRPS</td>
<td>$COPY, $MCOPY, $PCOPY, and $ERASE can copy or erase indirectly. When issuing one of these commands, you specify in parentheses the name of a matrix containing the names of objects to be processed.</td>
</tr>
<tr>
<td>$STACK</td>
<td>No replacement. Handled automatically.</td>
</tr>
<tr>
<td>$WSSIZE</td>
<td>No replacement.</td>
</tr>
<tr>
<td>WAS lib wsname, which is displayed with loading of the CONTINUE workspace under TSO</td>
<td>No replacement.</td>
</tr>
<tr>
<td>Horizontal Tabs $HT</td>
<td>No replacement. APL2 accepts the name $HT. However, referencing $HT yields 1 0 no matter how it is specified.</td>
</tr>
</tbody>
</table>

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](image)

<table>
<thead>
<tr>
<th>Dropped from VS APL</th>
<th>Replacement in APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A numeric ID could be specified as a first positional parameter when invoking VS APL under CMS.</td>
<td>The ID keyword provides equivalent function.</td>
</tr>
</tbody>
</table>
## 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**

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

<table>
<thead>
<tr>
<th>External Routine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Conversion</strong></td>
<td></td>
</tr>
<tr>
<td>ATR</td>
<td>Convert an APL array to a record with mixed data types</td>
</tr>
<tr>
<td>CTK</td>
<td>Convert extended character data to mixed DBCS data</td>
</tr>
<tr>
<td>CTN</td>
<td>Convert character data to numeric data</td>
</tr>
<tr>
<td>DFMT</td>
<td>Format an array of extended character data</td>
</tr>
<tr>
<td>KTC</td>
<td>Convert mixed DBCS data to extended character data</td>
</tr>
<tr>
<td>PFA</td>
<td>Generate a pattern for ATR or RTA</td>
</tr>
<tr>
<td>RTA</td>
<td>Convert a record to an APL array</td>
</tr>
<tr>
<td>CAN</td>
<td>Compress and Nest</td>
</tr>
<tr>
<td>DAN</td>
<td>Delete and Nest</td>
</tr>
<tr>
<td>SAN</td>
<td>Slice and Nest</td>
</tr>
</tbody>
</table>

**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
- **BUILDRDL**: 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**: A program interface to Editor 2
- **EDITORX**: A program interface to a named system editor
- **IN**: Program access to system command IN
- **OUT**: Program access to system command OUT
- **PIN**: Program access to system command PIN

**REXX Access (Processor 10)**

- **EXEC**: Execute a REXX program
- **F**: Obtain information about a CMS or MVS file
- **FM**: Read or write a file as a matrix
- **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
- **CSRSSCOT**: Save MVS/ESA virtual data object changes in a scroll area
- **CSRVIEW**: Define a view on an MVS/ESA virtual data object
- **DSQClA**: Interact with the database Query facility
### Figure 31 (Page 2 of 2). APL2/370 External Routines

<table>
<thead>
<tr>
<th>External Routine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Environment Control</strong></td>
<td></td>
</tr>
<tr>
<td>ATTN</td>
<td>Query or reset the attention flag</td>
</tr>
<tr>
<td>MSG</td>
<td>Use APL2 message facilities from an application</td>
</tr>
<tr>
<td>OPTION</td>
<td>Query or set APL2 invocation options</td>
</tr>
<tr>
<td>PBS</td>
<td>Query or set the PBS state</td>
</tr>
<tr>
<td>RAPL2²</td>
<td>Run the remote-session manager</td>
</tr>
<tr>
<td>SERVER</td>
<td>Start a TCP/IP port server</td>
</tr>
<tr>
<td>SVI</td>
<td>Determine shared variable processor numbers or user IDs.</td>
</tr>
</tbody>
</table>

| **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 |
| IDIOMS²           | 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 (I) 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 \textit{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 \textit{TRANSFER} workspace.

6. Use \texttt{)ERASE (GPTRANSFER)} to eliminate \textit{TRANSFER} workspace functions and variables.

\textbf{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. \texttt{)SAVE}.

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

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

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

\texttt{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. \texttt{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 quadFX:

\texttt{'| 'quadFX' FLAG ALL'}

If you specify no search argument, \texttt{FLAG} prompts for character strings, which you enter one at a time.

The \textit{TRANSFER} workspace includes the variable \texttt{FLAGMVSAPL}; this is a prepared list of character strings, which includes all known migration problem areas. \texttt{FLAGMVSAPL} is a convenient argument for \texttt{FLAG}. For example, the following expression searches for all known migration problem areas in all functions in the workspace:

\texttt{FLAGMVSAPL FLAG ALL}

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

The \textit{TRANSFER} workspace includes the variable \texttt{FIXMIUP}; this is a prepared list of old and new pairs.

\textbf{Warning:} \texttt{FIX} modifies the workspace. Be sure that you want to make the changes you have indicated in all examples in all functions listed. \texttt{FLAG} can be used to identify all examples of the old character strings for inspection before you use \texttt{FIX}. Also, although \texttt{FLAGMVSAPL}, \texttt{FIXMIUP}, and \texttt{ALL} are available arguments to \texttt{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 (\( \text{L} | \text{R} \)) can be affected by \( \text{CT} \) (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 \( \text{NC} \) (name class) to test for an invalid object name. Under VS APL, \( \text{NC} \) returns \( ^4 \) for an invalid object name. Under APL2, \( \text{NC} \) returns \( ^{-1} \). (\( ^4 \) indicates a defined operator name.)

Use FLAG to identify functions that use \( \text{NC} \) to test for invalid object names.

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

\[
\text{VZ} \cdot \text{FN} \text{ A} ; \text{A}
\]

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, \( \text{input prompts until valid data is received, but no longer rejects } ^{\text{AB}} \ 2 \).

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 \( \text{EA} \) (execute alternate). With \( \text{EA} \), 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:

\[ \begin{array}{c}
4 \\
5 \\
6[3] \\
\end{array} \]  

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.

\[ \begin{array}{c}
4 \\
5 \\
(6[3]) \\
\end{array} \]  

*RANK ERROR*  
\[ \begin{array}{c}
4 \\
5 \\
(6[3]) \\
\end{array} \]  

`^^`

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

\[ \begin{array}{c}
(4 \\
5 \\
6)[3] \\
\end{array} \]  

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 (\( \pi \)) functions
- Use of \( AV \)
- \( PR \) setting
- Numerical precision

Formatting: Reports or results that you produce with format (default) (\( \pi R \)) or format by specification (\( LR \)) 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 \( \pi \). VS APL functions that use character input/output \( \pi \) 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.

<table>
<thead>
<tr>
<th>AP Environment</th>
<th>VS APL Option</th>
<th>VS APL Conversion</th>
<th>APL2 Option and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 CMS</td>
<td>none</td>
<td>$\Box$ to a . . . $\wedge$ to &amp; . . .</td>
<td>370 (default) – equivalent EBCD – no conversion</td>
</tr>
<tr>
<td>100 TSO</td>
<td>none</td>
<td>Full 256 EBCDIC</td>
<td>EBCD (default) – equivalent 370 – like CMS 370</td>
</tr>
<tr>
<td>101 CMS</td>
<td>$APL$</td>
<td>$\Box$ to A(bsp) . . .</td>
<td>Not supported</td>
</tr>
<tr>
<td>101 TSO</td>
<td>none</td>
<td>Full 256 EBCDIC</td>
<td>192 (default) – equivalent</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>AP Environment</th>
<th>VS APL Option</th>
<th>VS APL Conversion</th>
<th>APL2 Option and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>110 CMS</td>
<td>APL</td>
<td>À to A(bsp) ...</td>
<td>APL – equivalent</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>Full 256 EBCDIC</td>
<td>EBCD (or 192) – equivalent</td>
</tr>
<tr>
<td></td>
<td>370</td>
<td>À to a ...^ to &amp; ...</td>
<td>370 (or BCD) – equivalent</td>
</tr>
<tr>
<td>BIT</td>
<td>11000001 to A ...</td>
<td>BIT – equivalent</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>Unchanged for copying or AV indexing</td>
<td>BYTE – no converted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unchanged for storing APL character data</td>
<td>COD1 – EBCDIC to VS APL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>With translate table</td>
<td>BYTE – equivalent with conversion table</td>
<td></td>
</tr>
<tr>
<td>111 ALL</td>
<td>APL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>210 TSO</td>
<td>192(or EBCD)</td>
<td>See 110 CMS</td>
<td>See 110 CMS</td>
</tr>
<tr>
<td></td>
<td>370(or BCD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BYTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111 TSO</td>
<td>TN</td>
<td>Superscripts, plotting characters</td>
<td>TN – equivalent</td>
</tr>
<tr>
<td>210 TSO</td>
<td>none</td>
<td>Unchanged for copying or AV indexing</td>
<td>T(default) – equivalent</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Unchanged for storing character data</td>
<td>T1 – equivalent or T2 – store in EBCDIC</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>With translate table</td>
<td>T – equivalent with converted table</td>
</tr>
<tr>
<td>123 ALL</td>
<td>4 1(default).</td>
<td>256 EBCDIC on data, tables/symbol sets unchanged.</td>
<td>4 0(default), if tables/symbol sets unconverted</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>Nothing changed</td>
<td>4 1 if tables/symbol sets converted</td>
</tr>
</tbody>
</table>

**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 \textit{BYTE} option. Data can be read or written compatibly, using the \textit{BYTE} option, if:

- The file \textbf{does not} contain character data. On either system, the \textit{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 \textit{AV}. (This is functionally equivalent to the \textit{BIT} option.)
- An application is using the \textit{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 \textit{MCOPY}. If not, use the \texttt{ZC} table in \texttt{1 UTILITY} and apply the following conversion:

\begin{verbatim}
NEWTAB←\texttt{AV[zc\textbackslash OLDTAB]}
\end{verbatim}

Functions using the \textit{BYTE} option, which do not meet any of the above criteria must be modified.

- As a short-term solution, the \textit{BYTE} option can be changed to \textit{COD1}.
- An efficient long-term solution is to convert the files and option to \textit{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 \textit{SWC} or \textit{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:
  
  \[
  GOODTAB \leftarrow zc[AV\backslash BADTAB]
  \]

- Change your application to use the AP 126 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:

<table>
<thead>
<tr>
<th>APL characters</th>
<th>Passed to CMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>^ &quot; ÷ ∆ ≠ ×</td>
<td>&amp; &quot; % # $ @</td>
</tr>
</tbody>
</table>

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 \texttt{COPY} command, when used to copy an entire workspace, has been changed to include the system variables \texttt{CT}, \texttt{FC}, \texttt{IO}, \texttt{LX}, \texttt{PP}, \texttt{PR}, and \texttt{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:

\begin{itemize}
\item \texttt{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, \texttt{NL}, \texttt{CR}, \texttt{FX}, \texttt{SVQ}, and \texttt{TF}), and system commands and messages that display names, convert lowercase letters to underbarred letters for their output.

\item \texttt{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.

\item \texttt{CASE(2)}: Underbarred characters are treated as invalid in names and are not accepted or produced in system functions, commands, or messages.
\end{itemize}

The invocation option \texttt{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 \texttt{CLEAR} commands. Note that \texttt{CLEAR} is the only way in APL to create a new workspace.

The CASE attribute assigned to a workspace during \texttt{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 \texttt{COPY}, \texttt{PCOPY}, or \texttt{IN} does not affect the CASE attribute of the workspace into which the objects are copied. A \texttt{CASE} function has been provided in the \texttt{UTILITY} workspace that returns the case of the active workspace.

When in a CASE(0) or CASE(1) workspace, \texttt{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 con-
nections 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 environ-
ments, 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 work-
space 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:

```apl
)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 `REBUILD211` 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, GC23-3058
- 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 the IBM PC: User’s Guide, SC33-0600
- 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:

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

Special Characters
)CONTINUE 18
)COPY 18, 26, 42
    extended for indirect copy 26
)CS 12, 33
    caution when using 33
    syntax 33
    use of 33
)ERASE 26, 39
)FNS 18
)HOST 12
)IN 12, 21
)LIB 18
)LOAD 19
)MCOPY 12
    AP 126 problems 38
    as migration tool 3
    choosing an alphabet 3
    comparison tolerance 7
    copying groups of objects 9
    copying individual objects 9
    during use of )MCOPY 9
    effects of 7
    effects on character data 4
    error messages 9
    for transferring workspaces 5
    handling of indexed vector constants 21
    index origin 7
    indirect copy 26
    installation options 7
    latent expression 7
    locked functions 7
    password 7
    random link 7
    specifying objects to be copied 9
    syntax 7
    system requirements for using 6
    system variables 7
)MORE 12
)NMS 12
)OFF 18
)OPS 12
)OUT 12
)PBS 12
)PCOPY 18, 26
)PIN 12
)RESET 18
)SIC 18
)SIS 18
)STACK 26
)STACK VS APL 26
)SYMBOLS 18
)VARS 18
)WSSIZE VS APL 26
)AF 12
)AT 12
)AV 34
)CR 23
)CT 7, 32
)EA 12
)EC 12, 21
)EM 12
)ES 12
)ET 12
)EX 23
)FC 12
)FX 18, 23, 32
)IO 7
)L 12
)LX 7, 19
)NA 12
)NC 23, 32
)NLT 41
    use by applications 41
)PP 7, 17
)PR 12
)PW 22
)R 12
)RL 7
)SVE 12
)SVO 24
)SVQ 26
)SVR 24
)SYS 12
)TF 12
)TZ 12
)WA 24

A
adjusting for )AV dependencies 4
ALL_, TRANSFER workspace function 31
alphabet
    underbarred letters 42
alternate input, checking 39
alternate input, possible problems with AP 101 37
alternative input, checking 22
AP 100 (CMS Command Processor)
    file translation 37
    translation options 35
AP 100 (TSO Command Processor)
    translation options 35
AP 101 (Alternate Input Processor)
possible translation problems 37, 39
translation options 24, 35
AP 102 (main storage access processor) 13
AP 110 (CMS File Processor)
BYTE option 37
possible translation problems 37
translation options 24, 36
AP 111 (QSAM Processor)
possible translation problems 37
translation options 24, 36
AP 120 (Session Manager Command Processor)
possible translation problems 37
AP 121 (APL File Processor)
file translation 37
AP 123 (VSAM Processor)
possible translation problems 37
translation options 36
AP 126 (GDDM Processor)
correcting problems 38
full-screen facilities 26
possible translation problems 38
replaces AP 124 26
translation options 36
AP 127
SQL access 13
AP 127 (SQL Processor) 13
AP 210 (BDAM Processor)
possible translation problems 38
translation options 24, 36
AP 211
transferring files 45
APL2
auxiliary processor translation options 24, 35
data types 32
transfer of workspaces from TryAPL2 45
transferring workspaces 3
APL2PI 28
APL2PIE 28
APLIBTAB file 7
APLYUOPT (TSO) module 8
arrays
as operands 17
characters and numbers mixed 15
display of nested data 16
nested 11
selective specification 17
assignment
under APL2 15
under VS APL 15
ASVPREP, return code from 24, 39
atomic vector (AV)
dependencies 4, 34
differences between VS APL and APL2 23
ATR 28
ATTN 28
auxiliary processor 1
AP 124 replaced by AP 126 26
APL2 compared to VS APL 20
distributed with APL2 13
solutions to file translation problems 35
translation options 24, 35
user-written
compatibility with APL2 39
facilities for 13
handling of new APL data types 25
B
backslash operator 12, 16
backup during migration 5
BUILDRD 28
BUILDRL 28
C
CAN 28
canonical representation 23
caret, to indicate error 18
CASE attribute 42
choosing an alphabet 3
setting 5
changes
summary of ix
changes in APL2 auxiliary processor translation options 35
character data
erroneous stacking of 39
input/output 23, 34
mixed with numeric data in arrays 15
translation from VS APL to APL2 4
character representation (CR) 23
checking
alternate input 39
alternative input 22
data files used by a function 35
user-written auxiliary processors 39
choosing an alphabet
MCOPY 3
CASE attribute 3
for workspace names 3
circle (L-R) function 22
CMS
LIBTAB APL2 file 3
workspace location for MCOPY 8
CMSIVP 28
compared to VS APL 11
comparison tolerance
copied by MCOPY 7
compatibility
VS APL with APL2 11
complex numbers 22
compression, primitive function 16
copy command
  VS APL compared to APL2 18
copying system variables 42
CSRIDAC 28
CSRREFR 28
CSRSAVE 28
CSRSCOT 28
CSRVIEW 28
CTK 28
CTN 28

D
DAN 28
data files used by a function, checking 35
data type
  cause of interrupted function processing 32
  for user-written auxiliary processors 39
  preventing accidental entry 32
  supported by auxiliary processors 20
data, translation of 20
DEBUG invocation option 19
debugging 18, 30
derived functions 12, 16
describe function 5
display 28
display of output 16
display format (default) function 22, 34
display by specification (L*R) function 22, 34
dropping backup workspaces 40
dyadic shared variable query 26
drop
  derived function 16
  expansion 16
  primitive function 16
expunge (\EX) 23
extended-compatible features
  language 15
  system 19
extended-incompatible features
  language 21
  system 24

F
facilities no longer supported under APL2 26
features
  extended-compatible 15, 21
  extended-incompatible 21
  new under APL2
    language 11
    system 13
  VS APL, not supported by APL2
    language 26
    system 26
FED 28
files 1
  checking translation of 35
  list of workspaces for access to 27
FIX, TRANSFER workspace function 31
FLAG, TRANSFER workspace function 31
format
  primitive function 16
  format (default) (+) function 22, 34
  format by specification (L*R) function 22, 34
formatting 34
full-screen editor 20
full-screen editor, editor 2 14
full-screen management 26
function interrupted during testing 32
function not interrupted during testing 34
functions
  defined
    as operands 17
    character representation 23
    editing the name of 20
    likely problems after transfer 30
    testing and debugging 30
  derived
    formed with backslash operator 12
    formed with slash operator 12, 16
locked 7
primitive 11
system 17, 23

G
Graphical Data Display Manager (GDDM)
  AP 126 problems 38
  required by full-screen editor 13
groups
    definition 7
    migration 8, 39
    named in stacked system commands 39
    not found by )MCOPY 9
    replacement under APL2 26, 39
    GSVP 44

H
HELP 28
highlighting of input, output 20
horizontal tabs (\HT) command 26
host system
    list of workspaces for access to 27
    requirements for migration 3

I
ID parameter 25
IDIOMS 28
index origin
    copied by )MCOPY 7
indexed numeric constants 33
indexed vector constants 21
Inner Product 22
inspecting, correcting, and testing functions 30
installation options
    )MCOPY 7
interrupted processing 32
invocation options
    compatible system features 19
    ID parameter 25
    new facilities under APL2 13

K
KTC 28

L
language features no longer supported by APL2 26
language names in \NLT 41
latent expression
    copied by )MCOPY 7
libraries
    creating directory for 3
    improperly referenced (error messages) 9
    specifying for )MCOPY 7
LIBTAB APL2 file 3
line editor 20
line editor, editor 1 14
local names 20, 32
local variables 32
locked functions
    copied by )MCOPY 7

lowercase 41
lowercase alphabet 42

M
messages
    lowercase 41
migrating
    under CMS
        APLIBTAB VS APL file 3
    migrating workspaces 43
migration
    backup 5
    definition 1
    from another host system 3
    objects from VS APL to APL2 8
    overview 1
    planning for 1
    preparing for 2
    tasks 1
    time stamp 7
    under CMS 8
        APLIBTAB APL2 file 3
        workspace location for )MCOPY 3
        workspace location for )MCOPY 8
    workspace location for )MCOPY 8
    mixed case 41
    mixed data 11
    MSG 28

N
name class function (\NC) 23, 32
national language
    names in \NLT 41
    three-character codes 41
national language support 12
national language translation 12
nested arrays 11, 16
new features in APL
    language 11
    new features in APL2
        system 13
    numerical precision 34

O
object names 16
objects
    not copied by )MCOPY 9
    not found by )MCOPY 9
    specifying with )MCOPY 8
operators
    defined
        character representation 23
        editing the name of 20
operators (continued)
    defined (continued)
        new language features under APL2 12
        primitive 11
OPTION 28
output, display of 16

P
PACKAGE 28
packaged workspaces 28
parentheses
    high precedence during evaluation 21
    use in APL2 expressions 16
password
    missing or incorrect (error message) 9
    used with )MCOPY 7
PBS 28
PFA 28
power (L*R) function 22
preparation for transferring workspaces 3
primitive functions
    changes incompatible with VS APL 22
    compatible language features 16
    new language features under APL2 11
primitive operators 11
printing precision (PP)
    system functions and variables 18
prompt replacement (PR) 23, 34

Q
QNS 28
quote-quad 23

R
random link
    copied by )MCOPY 7
RAPL2 28
replacement for VS APL features in APL2 26
replication, derived function
reports, alignment problems with 34
RESIDUE function 32
right arrow 18
RTA 28
run-time instructions 40

S
SAN 28
session manager
    FIND command 13
    shared variable offer 24
    shared variable processor (SVP) 39, 44
    considerations 44
shared variable query (SVQ), dyadic 26
shared variable retraction (SVR) 24
shared variables 12
space, insufficient for copying objects 9
SQL, access through AP 127 13, 27
stacked data
    possible problems with AP 101 39
    system commands with group names 39
state indicator, (*) 18
stop control (SI) 18
summary of changes ix
SVI 28
symbol set, problems with )MCOPY 38
syntax
    )CS 33
    )MCOPY 6
system commands
    )ERASE 39
    indirect )COPY 26
    indirect )ERASE 26
    indirect )MCOPY 26
    indirect )PCOPY 26
    national language support for 12
    new in APL2 12
    used in debugging 18
system editors 14, 20
system features
    compatible-extended 19
    incompatible—extended 23
    new for APL2 13
    no longer supported 26
system functions
    changes compatible with VS APL 19
    changes incompatible with VS APL 23
system limits 9
system messages
    AXIS ERROR 13
    SYSTEM ERROR 13
    VALENCE ERROR 13
system messages, new 12
system requirements
    )MCOPY 6
system variables 42
    changes incompatible to VS APL 23
    copied by )MCOPY 7
    new under APL2 12

T
testing and debugging 30
testing functions under APL2 32
    function interrupted during testing 32
    function not interrupted during test 34
testing the application as a whole 40
testing transferred functions 32
things to look for when debugging 32

TIME 28
time stamp
during migration 7
transfer file 12
transfer files
moving between systems 44
transfer procedure 5
TRANSFER workspace
functions and variables
ALL_ 31
FIX_ 31
FLAG_ 31
procedure for using 30
use of DESCRIBE function 5
transferring files 45
transferring selected objects
groups of objects 8
individual objects 8
transferring workspaces 2, 44
from VS APL to APL2 3
preparing for 3
translate tables, problems with )MCOPY 38
translation option
for auxiliary processors 35
inappropriate use 39
translation problems 37
TryAPL2
transfer of workspaces to workstations 45
TSO
APLYUOPT module 8
workspace location for )MCOPY 8
TSOIVP 28

U
underbarred alphabet 42
user-written auxiliary processor
APL2 facilities for 25
new return code for 39
used by function 39
user-written auxiliary processors, checking 39

V
vector
constants 7
VS APL
continuing use of 1
discontinuing use of 2
features compatible with APL2 15
features incompatible with APL2 21
features not supported by APL2 26
running concurrently with APL2 1
service continued 1
service discontinued 2

VS APL (continued)
transferring workspaces 3
workspace 4
VS APL compared to APL2
auxiliary processor options (incompatible) 24
auxiliary processors (compatible) 20
data representation (compatible) 15
display of output 16
editors (compatible) 20
evaluation of expressions (compatible) 15
evaluation of expressions (incompatible) 21
invocation options (compatible) 19
object names (compatible) 16
primitive functions (compatible) 16
primitive functions (incompatible) 22
selective specification (compatible) 17
system commands (compatible) 18
system functions and variables (incompatible) 23

W
workspace
backup during migration 5
backup for migration 2
location for )MCOPY 8
locked (error message) 9
not found (error message) 9
size error 9
use of )MCOPY command for copying 3
VS APL 4
workspace case 42
workspace migration 43
workspaces 1
migration from TryAPL2 45
transferring between APL2 systems 44
workspaces distributed with VS APL and APL2 27
workspaces, packaged 28

U
underbarred alphabet 42
user-written auxiliary processor
APL2 facilities for 25
new return code for 39
used by function 39
user-written auxiliary processors, checking 39

V
vector
constants 7
VS APL
continuing use of 1
discontinuing use of 2
features compatible with APL2 15
features incompatible with APL2 21
features not supported by APL2 26
running concurrently with APL2 1
service continued 1
service discontinued 2
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?

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technically accurate</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Complete</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to find</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Well organized</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Grammatically correct and consistent</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Graphically well designed</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Overall satisfaction</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

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.
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
GC26-3359 APL2 for Sun Solaris Licensed Program Specifications
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