Burroughs

Progression Guide

B 1000 Series to A Series

(Relative to the Mark 12.0 Small Systems Software Release, the 7.0 Small Systems GEMCOS Software Release, and the Mark 3.6 A Series Systems Software Release)

Distribution Code SE

Priced Item
Printed in U.S.A.
February 1987

1180593

Burroughs

Progression Guide

B 1000 Series to A Series

(Relative to the Mark 12.0 Small Systems Software Release, the 7.0 Small Systems GEMCOS Software Release, and the Mark 3.6 A Series Systems Software Release)

Distribution Code SE

Priced Item
Printed in U.S.A.
February 1987

1180593

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the Class specified as "2" (Systems Software), the Type specified as "1" (F.T.R.), and the Product specified as the seven-digit form number of the manual (for example, "1180593"). The FCF should be sent to the following address:

Unisys Corporation
Product Assurance and Support
19 Morgan Avenue
Irvine, CA 92718

CONTENTS

1 INTRODUCTION

Progression is the process of upgrading a data processing facility from a B 1000 Series to the A Series or B 5000/B 6000/B 7000 Series of Systems. This manual provides instruction and tips for this process.

This manual is written to take you through the progression process. It discusses the differences encountered when progressing from the Small Systems to the A Series and B 5000/B 6000/B 7000 Series, describes common progression problems and provides solutions to these problems, and gives general information for handling the overall progression process.

Throughout this manual, the B 1000 Series is referred to as "Small Systems." "A Series" applies to both the A Series and the B 5000/B 6000/B 7000 Series.

ORGANIZATION OF THE MANUAL

The manual is divided into the following sections:

Section 1	Introduction
Section 2	The Progression Process This section discusses the keys for successful progression, progression tasks, and progression aids.
Section 3	File Handling This section discusses the differences between Small Systems and A Series file naming conventions, file naming attributes, file security, usercodes, file assignments, and serial numbers.
Section 4	Work Flow (Jobs) This section discusses how to progress to A Series WFL from Small Systems WFL, job control attributes, and job spawning.

Section 5	File And Program Attributes This section lists the Small Systems file and program attributes and their A Series equivalents.
Section 6	Transferring Files This section discusses the various file transfer methods and their use.
Section 7	COBOL(68) This section discusses the reasons why you should convert to COBOL74 as part of the progression to the A Series.
Section 8	COBOL74 This section discusses the differences between Small Systems COBOL74 and A Series COBOL74 and gives instruction about progressing to A Series COBOL74.
Section 9	RPG This section discusses the differences between Small Systems RPG and A Series RPG and how to progress to A Series RPG.
Section 10	DMSII Operations This section discusses the operational differences between DMSII on Small Systems and DMSII on the A Series.
Section 11	DMSII DASDL This section discusses the differences between Small Systems DASDL and A Series DASDL and how to progress to A Series DASDL. It also discusses differences which affect application programs.
Section 12	SMCS to COMS This section describes the process of progressing from the Small Systems SMCS to the A Series COMS.

Section 13	GEMCOS/COMS	This section gives a general overview of the differences between Small Systems GEMCOS and A Series COMS. It lists the GEMCOS statements and their A Series direct or functional equivalents and shows how to progress from GEMCOS to COMS.
Section 14	SORT	This section discusses the differences between Small Systems and A Series SORT and how to progress to A Series SORT.
Section 15	REPORTER III	This section lists the differences between Small Systems and A Series REPORTER III.
Section 16	ODESY	This section discusses the progression from Small Systems On-Line Data Entry System (ODESY) to the A Series ODESY.
Section 17	ISAM Files	This section discusses progressing from the three types of Small Systems ISAM files to A Series ISAM files.
Section 18	QUEUE/PORT Files	This section discusses the differences between Small Systems Queue files and A Series Port files and how to progress to A Series Port files.

PROGRESSION ASSISTANCE

If you have a progression problem or solution that is not covered in this document, send a detailed description to:

BURROUGHS CORPORATION
Progression Aids Support
19 Morgan Avenue
Irvine, CA 92718

Of course, if you have a problem that requires immediate attention, contact your local Burroughs Technical Representative.

2 THE PROGRESSION PROCESS

Upgrading from Small Systems to the A Series requires changes to the software. Burroughs supports progression by providing progression aids, this documentation, and tips to aid you during the progression process.

KEYS FOR SUCCESSFUL PROGRESSION

The following are the three steps to a successful progression.

1. Plan. Planning should start when you sign the order for the A Series. You should have a plan for the entire progression process before you begin the progression. To help you plan your progression, a list of the progression tasks and types of information you need for the progression is given later in this section.

2. Train. Take advantage of Burroughs Customer Education classes before you start the progression. A thorough understanding of the A Series will help the progression process considerably. Burroughs Customer Education classes include a Progression Series oriented specifically for the Small Systems user moving to the A Series. Contact your local Burroughs Technical Representative for more information about the customer education classes.

3. Do a straight progression. Do not attempt to enhance or fix the software during the progression. Make only those changes necessary to get the software running on the A Series. After the progression is complete and you feel comfortable with the A Series, take advantage of the features on the A Series and enhance the software.

PROGRESSION TASKS

The following is a list of suggested progression tasks. You may need to modify the activities on this list to meet your needs.

1. Freeze the program library. Compile the source code and make a copy of the source and object files. To the maximum extent possible, do not make changes to the source or copy libraries after this point.

2. Prepare the progression package. To convert the code, you need to gather the complete source code, copy libraries, translation aids, and conversion instructions (new naming conventions, data base implementation information, data communication information, etc.).

3. Deliver the progression package. Give the progression package to the individual or team who is responsible for the progression.

4. Translate the source code to a clean compile. Generally this is done using one of the Burroughs-supplied translators, but it may be done manually if no tool exists. Even with the use of a translator, it may be necessary to make some manual changes to achieve a clean compile.

5. Transfer the test data files. Retain a copy of the Small Systems test data on the Small Systems for testing purposes until you have completed the progression. The test data files should be varied enough to ensure a thorough test of the entire program and small enough to keep the run times short.

6. Perform the unit tests and make the necessary corrections. Run unit tests to verify individual programs after they have been converted. To complete a unit test successfully and efficiently, obtain access to the source code, documentation, all input and output files, and all work files. The work files are an important part of the debugging process because they enable you to check your intermediate results.

   When the unit test reveals that the results on the A Series are identical to the results on the Small Systems, you have successfully converted the program.

7. Perform a system test. Do a systems test after completing all the unit tests of the programs which make up the systems. A systems test requires documentation, all input files, and all files associated with the systems.

   The systems test is successful when the results produced by the A Series are identical to the results obtained from the Small Systems.

8. Update the documentation. Update all documentation to reflect the changes made to the applications software.

9. Implement changes to non-frozen programs. Once you are comfortable with the progression process, make the necessary changes to any non-frozen programs.

   Unit and systems test these programs just as you did the frozen programs.

10. Deliver the translated program to operations.

11. Run parallel tests. Run identical processing on both the Small Systems and A Series, then compare the results.

12. Begin live operation. Continue processing on the A Series. The progression is now complete.

PROGRESSION AIDS

Burroughs provides a series of aids to assist in the progression from the Small Systems to the A Series. These progression aids include source code translators, data base conversion aids, file transfer utilities, and sample programs. Available with the A Series 3.6 release is the BTA360 Migration Aids tape. This tape contains the following modules. Several of the modules have supporting programs and printer backup files.

Online Controller to A Series (OCA)	A menu driven program for running BRT, B7T, and CTA. The primary program is OBJECT/BTA/CONTROLLER.
COBOL(68) to COBOL74 on A Series (CTA)	Translates Small Systems COBOL(68) to A Series COBOL74. The primary program is CTACOB.
Burroughs RPG Translator (BRT)	Translates Small Systems RPG to A Series RPG. The primary program is BRTRPG.
Burroughs COBOL74 Translator (B7T)	Translates Small Systems COBOL74 to A Series COBOL74. The primary program is B7TCOB.
Data Base Data Translator (DBT)	A Small Systems DMS to A Series DMS conversion tool. The primary program is SYSTEM/DBTGEN.
B1000COPY	Reads B1000 System/Copy tapes on the A Series.
Miscellaneous Sample Programs	

The BTA350 Small Systems Conversion Tape, available with the previous progression aids release, also contains some useful progression aids. These include:

B6000COPY	Reads A Series Library Maintenance tapes on the Small System.
Miscellaneous Sample Programs	

3 FILE HANDLING

There are important differences between Small Systems and A Series file handling. The differences are in the file naming conventions, the file naming attributes, file security, usercodes, and serial numbers.

FILE NAMING CONVENTIONS

On the A Series, file names are composed of multiple components. The first component is the usercode followed by the file-directory identifiers (ID) under which the file is stored. The next component is the file ID that identifies the file. Finally, the family name specifies the physical device family upon which the file is stored. The syntax to access a file is:

```
(<usercode>) <file-directory ID> <file ID> ON <family name>
```

Each of these components is built out of identifiers. Each identifier consists of from 1 to 17 alphanumeric characters, hyphens, and underscores, or it consists of a quoted string up to 17 characters. Although special characters are allowed in file names, they require special handling by the system, so we recommend that you avoid using them.

The usercode is optional. If a usercode is specified, it must have parentheses around it. An asterisk (*) may be used instead of a usercode to indicate that the file is located among the general system files. If a usercode or asterisk is specified, a slash (/) cannot be used to separate the usercode or asterisk from the rest of the file name.

The file-directory ID is also optional. This identifier is the same as the multi-file ID on Small Systems. There may be from 0-11 file-directory IDs. This structure reflects the hierarchical nature of the file system. Each file-directory ID is followed by a slash (/).

The file ID is the only file identifier that is required.

On the A Series, files are stored in "families." A family is one or more disk packs which are treated by the system as if they were a single unit. The user controls how many and which disk packs constitute any family. Each family is assigned a family name for identification. The family name is similar to the pack ID on Small Systems. The family name is optional. However, when the family name is used, it must be separated from the rest of the file name by the keyword ON.

On the A Series, the first identifier is never used to specify the family name (pack ID on the Small Systems).

The following are examples of valid A Series file titles. In all of the examples:

F	represents the file ID
Z and Y	represent the family names
U	represents the usercode
all other letters	represent file-directory IDs
*	denotes that the file does not have a usercode and that the file is located among the general system files.

```
A/B/F
D/E/F ON Z
E/F ON Y
A/F ON DISK
*A/B/C/D/E/F
*C/F
*A/F ON Z
(U)B/F ON Z
(U)C/D/F
(U)C/F ON Z
(U)C/F ON DISK
(U)A/B/C/D/E/G/H/I/J/K/L/F ON Y
```

FILE NAMING ATTRIBUTES

Listed below are the Small Systems file naming attributes and their A Series equivalents. For more information about the A Series attributes, see the "A Series I/O Subsystem Reference Manual."

MULTI-PACK = TRUE	On the A Series, the default allows the areas of a disk file to be distributed over the entire family, which may consist of multiple packs. The A Series SINGLEUNIT attribute indicates whether areas for the disk file are to be allocated from a single family member (pack). The default value for SINGLEUNIT is false.
NAME	The A Series equivalent is TITLE with an optional FAMILYNAME. The value of the

	TITLE attribute on the A Series specifies the external file name. The default TITLE for the file is the value of the INTNAME attribute (INTNAME is the internal file name). Setting the TITLE attribute sets the FILENAME attribute. If "ON <family name>" is included in the TITLE, the FAMILYNAME attribute is set to <family name> and the KIND attribute is set to DISK.
PACK-ID	The A Series equivalent is FAMILYNAME. The FAMILYNAME indicates the name of the family (one or more packs) on which the physical file is located. FAMILYNAME must be a simple identifier of up to 17 characters. If a FAMILYNAME is not specified for a disk file, areas for the file are allocated from a family with a label of DISK. Because the A Series allocates space for files without a FAMILYNAME from DISK, we recommend labeling at least one pack as DISK on the A Series.
TITLE	The A Series equivalent is TITLE. The A Series TITLE attribute specifies the external file name in the form "<file identifier> ON <family name>." The "ON <family name>" is optional.
SECURITYUSE	The A Series equivalent is SECURITYUSE. See SECURITYUSE in the "File And Program Attributes" section.
SECURITYTYPE	The A Series equivalent is SECURITYTYPE. See SECURITYTYPE in the "File And Program Attributes" section.

FILE SECURITY

Access to a file may be restricted by:

1. Associating a usercode with the file.

2. Setting the file SECURITYTYPE to PRIVATE.

3. Setting the file SECURITYUSE to other than IO.

4. Any combination of the above.

Once a file is secured, it can be accessed, removed, or changed only by an authorized user.

Usercodes on Small Systems are created through the SYSTEM/MAKEUSER program. This program creates a file on the system disk labeled (SYSTEM)/USERCODE. This file contains all valid usercode/password pairs along with information such as default pack ID, maximum priority, charge code, and other security information associated with each usercode/password pair.

Usercodes on the A Series are also created through a program called SYSTEM/MAKEUSER. The file created to store the usercodes, passwords, and related information resides on the pack specified for USERDATA through the ODT Disk Location (DL) command and is labeled SYSTEM/USERDATAFILE. For more information about SYSTEM/MAKEUSER on the A Series, see the A Series System Software Site Management Reference Manual.

If a SYSTEM/USERDATAFILE does not exist, running SYSTEM/MAKEUSER from the ODT with no input will create one. The next step is to create the first usercode via the MAKE USER (MU) ODT command. The first usercode should be privileged in order to allow the running of SYSTEM/MAKEUSER from a terminal. The appropriate usercodes can then be entered and maintained. The MU ODT command can be inhibited through SYSTEM/MAKEUSER once the SYSTEM/USERDATAFILE has been established. We suggest using SYSTEM/MAKEUSER to establish and maintain the usercodes, not the MU ODT command.

On the A Series, a default FAMILYNAME of DISK is assigned to any disk file name for which no FAMILYNAME is specified. This default FAMILYNAME assignment is done prior to any FAMILYNAME substitution through the FAMILY specification.

The A Series FAMILY specification indicates a FAMILYNAME substitution which is used in assigning a FAMILYNAME to files referenced by a task. The default FAMILY specification for a task is obtained from the SYSTEM/USERDATAFILE entry for the usercode associated with that task. This default FAMILY specification can be overwritten through the FAMILY statement in WFL or CANDE. The A Series also allows specification of an alternate FAMILYNAME substitution through the use of the OTHERWISE clause of the FAMILY statement. This alternate FAMILYNAME, if specified, is used only when referencing existing files; it is ignored when a new file is being created.

Example 1

In this example, a job is run under a usercode with a FAMILY entry in the SYSTEM/USERDATAFILE of "FAMILY DISK = USERA OTHERWISE DISK".

If this specification is not overridden, all references within the job to the FAMILYNAME "DISK" will reference FAMILYNAME "USERA." All new files not specifically assigned to another FAMILYNAME will be created on USERA. USERA will be searched first when an existing file with the FAMILYNAME "DISK" is referenced. If the desired file cannot be found on USERA, then the alternate FAMILYNAME (labeled "DISK") will be searched.

Example 2

In this example, the above job is modified to the following WFL FAMILY statement:

```
FAMILY DISK = USERB ONLY.
```

If this statement is used, all references within the job to the FAMILYNAME "DISK" will reference FAMILYNAME "USERB."

Example 3

In this example, the FAMILY statement is changed to:

```
FAMILY USERA = USERB OTHERWISE USERC.
```

If this statement is used, all references within the job to the FAMILYNAME "USERA" will reference FAMILYNAME "USERB." All new files assigned to USERA will be created on USERB. USERB will be searched first when an existing file with the FAMILYNAME "USERA" is referenced. If the desired file cannot be found on USERB, then the alternate FAMILYNAME (labeled "USERC") will be searched.

USERCODES

On Small Systems, the usercode is not appended if the external file name has both a multi-file ID and a file ID specified. On the A Series, a usercode is automatically appended to the file name being created as long as the task is running under a usercode and an asterisk does not precede the file name. When the usercode is appended, it becomes the first identifier. For example:

```
     (X)A ON USERA    or   (X)A/B/C/D ON USERA.
```

On the A Series, multiple users may log on to CANDE with the same usercode/password. This is not permitted on Small Systems.

On both the Small Systems and the A Series, there are two types of usercodes: privileged and non-privileged.

Privileged Userdodes

On the A Series, when running under a privileged usercode, any file can be accessed or created. A privileged usercode is one defined as such in the SYSTEM/USERDATAFILE. On the A Series, the privileged status may be given or withdrawn through the SYSTEM/MAKEUSER program or through the MU ODT command. Some restrictions exist with the MAKE USER (MU) command.

The following tables show how Small Systems and A Series file names are handled while running under a privileged usercode. For these tables:

X	represents a privileged usercode with a FAMILY specification of "FAMILY DISK = USERX OTHERWISE DISK" (default pack of USERX on Small Systems)
Y	represents a non-privileged usercode with a FAMILY specification of "FAMILY DISK = USERY OTHERWISE DISK" (default pack of USERY on Small Systems)

File Handling

Creating a new file

If a program attempts to create a new file while running under usercode (X), the system modifies the file names as follows:

Declared File Name		Actual File Name	
Small Systems	A Series	Small Systems	A Series
A	A	(X)/A ON USERX	(X)A ON USERX
*A	*A	A	*A ON USERX
A/B	A/B	A/B ON USERX	(X)A/B ON USERX
*A/B	*A/B	A/B	*A/B ON USERX
(Y)/A	(Y)A	(Y)/A ON USERY	(Y)A ON USERX
*(Y)/A	*(Y)A	(Y)/A	Illegal
USERY/(Y)/A	(Y)A ON USERY	(Y)/A ON USERY	(Y)A ON USERY

Accessing an existing file

If a program attempts to access an existing file while running under usercode (X), the system modifies the file name as shown in the following example.

Declared File Name		Actual File Name	
Small Systems	A Series	Small Systems	A Series
A	A	(X)/A ON USERX then A	(X)A ON USERX then *A ON USERX then (X)A ON DISK then *A ON DISK
A/B	A/B	A/B ON USERX then A/B	(X)A/B ON USERX then *A/B ON USERX then (X)A/B ON DISK then *A/B ON DISK
(Y)/A	(Y)A	(Y)/A ON USERY then (Y)/A	(Y)A ON USERX then (Y)A ON DISK
USERX/(Y)/A	(Y)A ON USERX	(Y)/A ON USERX	(Y)A ON USERX
USERY/(Y)/A	(Y)A ON USERY	(Y)/A ON USERY	(Y)A ON USERY

In some cases, the A Series performs a more extensive search then Small Systems when trying to locate a file. The preceding example displays the order in which the system looks for a file, however, an open is attempted only on the first file found.

<u>Non-privileged Usercodes</u>

On the A Series, a program running under a non-privileged usercode can create a file on any FAMILYNAME that is specified by the program. An attempt to create a new file whose file name begins with either an asterisk (*) or with a usercode other than the one under which the program is running results in a security violation error. An attempt to access an existing file whose file name begins with either an asterisk (*) or with another usercode will be successful only if the file is PUBLIC and the SECURITYUSE of the file matches the open type specified by the program.

File Handling

For the A Series, as on Small Systems, while running under a non-privileged usercode, there are restrictions on the file names that can be accessed. For the following tables:

X	represents a privileged usercode with a FAMILY specification of "FAMILY DISK = USERX OTHERWISE DISK" (default pack of USERX on Small Systems)
Y	represents a non-priveleged usercode with a FAMILY specification of "FAMILY DISK = USERY OTHERWISE DISK" (default pack of USERY on Small Systems)

Creating a new file

If a program attempts to create a new file while running under the non-privileged usercode (Y), the system modifies the file name as follows.

Declared File Name		Actual File Name	
Small Systems	A Series	Small Systems	A Series
A	A	(Y)/A ON USERY	(Y)A ON USERY
USERZ/B/	B ON USERZ	(Y)/B ON USERZ	(Y)B ON USERZ
(Y)/A	(Y)A	(Y)/A ON USERY	(Y)A ON USERY
*(Y)/A	*(Y)A	(Y)/A	Illegal
USERZ/(Y)/A	(Y)A ON USERZ	(Y)/A ON USERZ	(Y)A ON USERZ

Accessing an existing file

If a program attempts to access an existing file while running under the non-privileged usercode (Y), the system modifies the file name as follows.

B 1000 SERIES TO A SERIES PROGRESSION GUIDE

Declared File Name		Actual File Name	
Small Systems	A Series	Small Systems	A Series
A	A	(Y)/A ON USERY then (Y)/A	(Y)A ON USERY then *A ON USERY then (Y)A ON DISK then *A ON DISK
*A	*A	A	*A ON USERY then *A ON DISK
A/B	A/B	A/B ON USERY then A/B	(Y) A/B ON USERY then *A/B ON USERY then (Y)A/B ON DISK then *A/B ON DISK
(X)/A	(X)A	(X)/A ON USERX	(X)A ON USERY then (X)A ON USERX
USERX(X)/A	(X)A ON USERX	(X)/A ON USERX	(X)A ON USERX
USERZ/A/B	A/B ON USERZ	A/B ON USERZ	(Y)A/B ON USERZ then *A/B ON USERZ
USERX/A/B	A/B ON USERX	A/B ON USERX	(Y)A/B ON USERX then *A/B ON USERX

The A Series, in some cases, performs a more extensive search when trying to locate a file than Small Systems. The preceding examples display the order in which the system looks for a file. However, an open is attempted only on the first file found. Once found, access is granted only if the SECURITYUSE of the file matches the open type. In addition, the file must either be PUBLIC or must be usercoded with the same usercode under which the accessing program is running (in this case (Y)) for access to be granted.

FILE ASSIGNMENT

Small Systems and the A Series take different approaches in their search for a file. Consider the case of a Small Systems program running under a usercode (X) with a default pack ID of USERX, versus an A Series program running under a usercode (X) with a FAMILY specification of "FAMILY DISK = USERX OTHERWISE DISK". Suppose each of these programs attempts to access a file with the declared filename A.

On Small Systems, the system will:

1. If the first character of the multifile ID is an asterisk (*), discard the asterisk and go to step 5.

2. If either the multifile ID or the file ID are not specified, append the usercode under which the program is running to the declared file name as the multifile ID.

3. If the pack ID is not specified, append the default pack ID to the filename. The pack ID that is appended is the default pack ID for the usercode that appears as the multifile ID of the file. If the multifile ID of the file is not a usercode, the default pack ID of the usercode under which the accessing program is running is used.

4. Search for the file.

5. If the file is not found, search for the file using the file name as originally specified.

Based on the previous examples, the Small Systems would search for the file USERX/(X)/A. If USERX/(X)/A is not found, the system would look for file A on the system disk.

On the A Series, the system will:

1. If the file name declared in the accessing program does not begin with a usercode or an asterisk, append the usercode under which the program is running to the file name.

2. Append the default FAMILYNAME of "DISK" if the file name is not specified with the "ON <family name>" clause.

3. Apply the FAMILYNAME substitution specified by the FAMILY specification in effect, if applicable.

4. Search for the file.

5. If the file is not found, substitute an asterisk for the usercode appended in step 1, and search for the file (if no usercode was appended in step 1, skip this step and step 7).

6. If the file is not found, go back to the file name as it existed just prior to step 3, apply the alternate FAMILYNAME substitution specified by the FAMILY statement in effect, and search for the file. If no FAMILYNAME substitution was done in step 3 or if no alternate FAMILYNAME substitution is specified in the FAMILY statement (that is, no OTHERWISE clause), then skip this step and step 7.

7. If the file is not found (and neither step 5 nor 6 were skipped), substitute an asterisk for the usercode appended in step 1 and search for the file.

Based on these rules, the A Series would search for:

```
(X)A ON USERX,
  then   *A ON USERX,
  then   (X)A ON DISK,
  then   *A ON DISK.
```

As on Small Systems, an open is attempted only on the first file found.

As shown in the examples in this section, the A Series potentially does a more extensive search, depending on the file name and the FAMILY specification in effect.

Accessing A File Under Another Usercode

Differences in Small Systems and A Series file assignments are evident when a program attempts to access a file under a different usercode than that under which the program is running.

Suppose a program running under usercode (X) attempts to access a file under usercode (B). (X) has a default pack ID of USERX on Small Systems or a FAMILY specification of "FAMILY DISK = USERX OTHERWISE DISK" on the A Series. (B) has a default pack ID of USERB on Small Systems or a FAMILY specification of "FAMILY DISK = USERB OTHERWISE DISK" on the A Series.

On Small Systems, if the program tries to open the file (B)/C, the system searches for file USERB/(B)/C. If the file is not found, the system searches for file (B)/C on the system disk.

On the A Series, if the program tries to open the file (B)C, the system searches for file (B)C ON USERX. If not found, it then searches for file (B)C ON DISK.

Thus, when determining the FAMILYNAME or pack ID for a file where none was originally specified, Small Systems use the pack ID associated with the usercode in the file name, while the A Series assigns a default FAMILYNAME of "DISK" and then applies any FAMILYNAME substitution called for by the effective FAMILY specification. This effective FAMILY specification is not the one associated with the usercode contained in the file name. Rather, the effective FAMILY specification defaults to the FAMILY specification associated with the usercode under which the program is running and may be explicitly changed within the job.

For a more detailed discussion of the A Series file naming conventions, refer to "File-Naming Conventions" in the "A Series I/O Subsystem Reference Manual."

SERIAL NUMBERS

The SERIALNO attribute exists on both the Small Systems and the A Series. This subsection describes the SERIALNO attribute and the importance of serial numbers to the A Series.

On the A Series, when the SERIALNO attribute is assigned a value, the SERIALNO is used in file assignment. When searching for a permanent disk file or selecting a family on which to create a disk file, the first element in a SERIALNO list, if assigned a value, is used to identify the base unit of the family.

Creating A New Non-Disk File

When creating a new non-disk file, the peripheral is assigned on the basis of the availability of a scratch or unassigned peripheral. When creating a new tape file and the SERIALNO attribute is set, a tape with a matching serial number that has a write ring and is one of the following:

not locked
not saved
uptape (not at the beginning of the tape)
not ready
not in use

is selected. The tape does not have to be scratched. If the tape has not been scratched, it will be rewound so that the new labels can be written at the beginning of the peripheral (i.e., purging the tape). Setting SERIALNO to null characters (all bits zero) indicates that the attribute is not to be considered during file assignment.

Finding A Permanent File

When a logical file is assigned to a permanent file, a number of attributes (KIND, TITLE, FILESECTION, SERIALNO, CYCLE, VERSION) are used to uniquely describe the physical file.

The KIND attribute narrows the search to certain peripherals. The TITLE attribute gives the external file name of the permanent file, and where appropriate, the FAMILYNAME (which corresponds to the pack ID on Small Systems).

Once the permanent file with the proper TITLE and correct KIND is found, a more detailed selection process follows.

For a tape file, the FILESECTION attribute must agree with the file section number of the permanent file.

For a disk or a tape file when genealogy checking is requested, the CYCLE and VERSION attributes are matched with those in a permanent file. If genealogy checking is not requested, the file with the best genealogy (the highest CYCLE value and the highest VERSION of that CYCLE) is selected. If the SERIALNO attribute is set for a tape file then the serial number of the physical tape must match the value of the SERIALNO attribute.

If a permanent tape file is found that meets all the requirements for assignment except for the serial number, an UNMATCHED SERIALNO notification is given to the operator. The operator can then respond by making the file available, or by entering one of the following system input messages:

1. IL (Ignore Label)

2. OF (Optional File)

3. FA (File Attribute)

4. DS (DiScontinue)

For a further discussion about the A Series SERIALNO, refer to the "A Series I/O Subsystem Reference Manual."

FILE REMOVAL

Removal of in-use files is different on the A Series than on Small Systems. On Small Systems, any attempt to remove an in-use file is rejected and a message is displayed stating that the file is in use. On the A Series, any attempt to remove an in-use file makes the file unavailable to users who did not have the file open when the remove was issued. The file is no longer visible through the file directory (i.e., it is not visible to a CANDE FILES command) and a new file with the same name can be created. However, the file remains available to those users who had the file open when the remove was issued. The file is removed when the last user closes the file.

4 WORK FLOW (JOBS)

Small Systems job flow can be controlled by Small Systems Work Flow Language (WFL), by job spawning, or by the AFTER, AFTER.NUMBER, THEN, CONDITIONAL, and UNCONDITIONAL attributes. The A Series job flow is controlled by WFL.

SMALL SYSTEMS WFL

Small Systems WFL is basically a subset of the A Series WFL with some exceptions.

Small Systems WFL can process only one program at a time. A Series WFL can process several programs simultaneously. 13.Φ

Small Systems WFL does not allow the passing of parameters to tasks. A Series WFL supports the passing of parameters to tasks. This feature can be used in many instances to replace ACCEPT functions and switch (SW) task attributes used during program initialization. 13.Φ

Converting to A SERIES WFL

Since Small Systems WFL is a subset of the A Series WFL, most of the constructs are the same. However, those Small System WFL constructs that are not allowed on the A Series are clearly flagged when the Small Systems WFL is compiled on the Small Systems. The Small Systems WFL features that are not available on the A Series are also flagged by the A Series compiler.

The following constructs are implemented on the Small Systems but not on the A Series:

1. <START statement> with a <task equation list>.
2. <COPY statement> using the following constructs:
    a. <Copy options list>.
    b. <Input volume spec> or <output volume spec> without an explicit KIND attribute.
    c. <Creation file attribute list>.

d. <Output volume attribute list> with SAVEFACTOR or DENSITY.

e. Multiple copy.

3. <MODIFY statement>.

4. <SECURITY statement> with a <directory name>.

JOB SPAWNING

The Small Systems job spawning capability is unique to Small Systems and is not available on the A Series. To progress to the A Series, the COBOL job spawning logic must be rewritten so that the jobs previously controlled by the job spawning program are handled by A Series WFL.

SMALL SYSTEMS CONTROL CARDS

The Small Systems Control Cards control the job flow by using the AFTER, AFTER.NUMBER, THEN, CONDITIONAL, and UNCONDITIONAL attributes. They default to processing all tasks asynchronously, unless one of the preceding attributes is specified.

The Small Systems Control Cards will have to be rewritten in A Series WFL.

The normal processing in A Series WFL is synchronous with each successive task being executed unconditionally. To run a task asynchronously, a WFL PROCESS statement must be placed immediately before each RUN statement. The A Series WFL uses task identifiers and task states to control the conditional execution of tasks.

The attributes used for controlling the job flow are described in the following pages.

AFTER

The AFTER attribute is used to conditionally schedule a program to run after the termination of another program (identified by program name).

Work Flow (Jobs)

On the A Series, as shown in the following example, the RUN PROG1 statement finishes execution of PROG1 before executing the next statement. The state of PROG1 is then checked before proceeding. PROCESS RUN PROG4 is an asynchronous task. The state of PROG2 is checked before running PROG3.

Example

```
     Small Systems                              A Series
     -------------                              --------

     EX PROG1;                            RUN PROG1 [TASK1];
     EX PROG2 AFTER PROG1;                IF TASK1 IS NOT COMPLETEDOK
                                             THEN ABORT;
                                          PROCESS RUN PROG4;
     EX PROG3 AFTER PROG2;                RUN PROG2 [TASK2];
     EX PROG4 AFTER PROG1;                IF TASK2 IS COMPLETEDOK
                                             THEN RUN PROG3;
```

<u>AFTER.NUMBER</u>

The AFTER.NUMBER attribute is used to conditionally schedule a program to run after the termination of another program (identified by job number) that is already in the mix or scheduled for execution. The A Series WFL controls the flow with the RUN and PROCESS statements. The WFL should be set up before the first task is entered into the mix or scheduled. The following example then becomes identical to the previous example.

Example

```
     Small Systems                              A Series
     -------------                              --------

EX PROG1;                                  RUN PROG1 [TASK1];
EX PROG2 AFTER.NUMBER <prog1-mix-number>;  IF TASK1 IS NOT COMPLETEDOK
                                              THEN ABORT;
                                           PROCESS RUN PROG4;
EX PROG3 AFTER.NUMBER <prog2-mix-number>;  RUN PROG2 [TASK2];
EX PROG4 AFTER.NUMBER <prog1-mix-number>;  IF TASK2 IS COMPLETEDOK
                                              THEN RUN PROG3;
```

THEN

On Small Systems, the THEN attribute is used to conditionally schedule a program to run after the termination of another program. The THEN attribute works like the AFTER attribute where the program name is from the previous program.

Example

```
Small Systems                         A Series
-------------                         --------

EX PROG1;                          RUN PROG1 [TASK1];
THEN EX PROG2;                     IF TASK1 IS NOT COMPLETEDOK
                                      THEN ABORT;
                                   RUN PROG2 [TASK2];
THEN EX PROG3;                     IF TASK2 IS COMPLETEDOK
                                     THEN RUN PROG3;
```

The AFTER, AFTER.NUMBER, and THEN attributes can be modified by using the CONDITIONAL and UNCONDITIONAL attributes.

CONDITIONAL

The CONDITIONAL attribute is used inhibit a program from being executed unless its predecessor successfully reaches normal EOJ. Programs terminated with either the ODT DS (Discontinue) command or a program fault, or compiles in which syntax errors are detected are considered to have reached abnormal termination. The CONDITIONAL attribute is set by default on Small Systems.

The A Series WFL uses the COMPILEDOK and COMPLETEDOK task states to conditionally execute tasks.

Example

```
Small Systems                         A Series
-------------                         --------

COMPILE X WITH COBOL74 LIBRARY;       COMPILE X WITH COBOL74[T]
FILE CARD NAME XSOURCE DISK DEF;         LIBRARY;
EX PROG1                              COMPILER FILE CARD
```

```
        AFTER COBOL74 CONDITIONAL;              (TITLE=XSOURCE,
                                                   DISK, DEPENDENTSPECS);
                                             IF T IS COMPILEDOK THEN
                                                RUN PROG1;
```

UNCONDITIONAL

The UNCONDITIONAL attribute is used to force the execution of a program regardless of its predecessor's outcome. This is the default on the A Series.

Example

```
     Small Systems                                 A Series
     -------------                                 --------

     COMPILE X WITH COBOL74 LIBRARY;         COMPILE X WITH COBOL74
     FILE CARD NAME XSOURCE DISK DEF;           LIBRARY;
     EX PROG1                                COMPILER FILE CARD
         AFTER COBOL74 UNCONDITIONAL;           (TITLE=XSOURCE,
                                                   DISK, DEPENDENTSPECS);
                                             RUN PROG1;
```

5 FILE AND PROGRAM ATTRIBUTES

To progress from Small Systems to the A Series, all references to Small Systems file and program attributes must be replaced by their A Series file and task attribute equivalents. Small Systems Control Card statements should be incorporated into WFL jobs on the A Series. Small Systems Control Card file attributes are accessed through the WFL FILE statement.

FILE ATTRIBUTES

The A Series has more file attributes than mentioned in this section. However, they are not necessary for the progression. After the migration is complete, familiarize yourself with these additional attributes then incorporate them into the A Series programs.

Small Systems	A Series
ALLOCATE.AT.OPEN	There is no A Series equivalent.
AREAS	The A Series equivalent is also AREAS. On the A Series, the default value for the maximum number of areas that can be allocated for a disk file is 20; the legal values are 1 to 1000.
ASCII	The A Series equivalent is EXTMODE/INTMODE. EXTMODE is the character recording mode of the physical file as stored on disk. INTMODE is the character recording mode of the logical file as used by the program. The default on the A Series is EBCDIC. To set the recording mode on the A Series to ASCII, set EXTMODE and INTMODE to ASCII in the program.
AUTOPRINT	The A Series equivalent is PRINTDISPOSITION. The default value for PRINTDISPOSITION is EOJ, which queues files for printing at end-of-job. Setting PRINTDISPOSITION to CLOSE is equivalent to AUTOPRINT; the files are queued for when the file is closed. Setting PRINTDISPOSITION to DONTPRINT is equivalent to NO AUTOPRINT on Small Systems.

BACKUP	The A Series equivalent is BACKUPKIND. When used with the KIND attribute, it allows the specification of printer and backup devices.
BACKUP.DISK	The A Series equivalent is BACKUPKIND = DISK or BACKUPKIND = DISKPACK. If BACKUPKIND is specified with the option DONTCARE, the A Series default values are used. It is not necessary to set the BACKUPKIND attribute if the MCP option LPBDONLY is set using the A Series ODT command OPTIONS (OP+LPBDONLY). When LPBDONLY is set, the printer output files are assigned to the printer backup disk. These files can then be printed using the Print System (PrintS).
BACKUP.TAPE	The A Series equivalent is BACKUPKIND. To direct a backup file to tape on the A Series, BACKUPKIND must be set to either TAPE, TAPE7, TAPE9, or TAPEPE.
BCL	The A Series equivalent is EXTMODE/INTMODE. This option is only permitted on the B 6800/B 7700/B 7800 systems. Refer to ASCII for a more complete explanation of EXTMODE/INTMODE.
BINARY	The A Series equivalent is EXTMODE and pertains only to card files. Refer to ASCII for additional information about EXTMODE.
BLOCKS.AREAS	The A Series equivalent is AREASIZE. AREASIZE is the number of logical records in an area of a disk file. To determine the value of AREASIZE, multiply records-per-block by blocks-per-area. If AREASIZE is unspecified or equal to zero and AREALENGTH is unspecified, a value equal to or close to 1000 is used.
BUFFERS	The A Series equivalent is BUFFERS. On Small Systems, this specifies the number of buffers assigned to a file. On the A Series, the default value is two buffers and the maximum is 63.

File and Program Attributes

CARD.PUNCH	The A Series equivalent is KIND = PUNCH.
CARD.READER	The A Series equivalent is KIND = READER.
CASSETTE	The A Series does not support this attribute.
COPY	The A Series does not support this attribute.
DATA.RECORDER.80	This attribute is not supported on the A Series.
DEFAULT	The A Series equivalent is DEPENDENTSPECS = TRUE. This attribute overrides the declared block and record sizes and uses the block and record sizes specified in the disk file header or tape label.
DELAYED.RANDOM	This attribute is not applicable on the A Series.
DISK	The A Series equivalent is KIND = DISK.
DISK.CARTRIDGE	This attribute is not supported on the A Series.
DISK.FILE	The A Series equivalent is KIND = DISK.
DISK.PACK	The A Series equivalent is KIND = DISK.
DRIVE	The A Series attribute FAMILYINDEX is similar to this Small Systems function but is not directly equivalent. FAMILYINDEX does not specify the drive, it specifies a particular member of a disk family that is to be used for the disk file. The default value 0 specifies that the areas of the disk file are to be allocated in the system's normal rotational order.
DUMMY.FILE	This attribute is not available on the A Series.
EBCDIC	The A Series sets recording mode using the EXTMODE and INTMODE attributes. EBCDIC is the default recording mode for the A Series. For more information about EXTMODE and EBCDIC, see ASCII.

EMULATOR.TAPE	This attribute is not available on the A Series.
END.OF.PAGE	The A Series equivalent is PAGESIZE. To request end-of-page reporting on the A Series, set PAGESIZE to the number of lines between the channel 1 and channel 12 punch on the printer.
EU	The A Series equivalent is DRIVE. See DRIVE.
EVEN	The A Series equivalent is PARITY = NONSTANDARD or PARITY = EVEN.
EXTEND	This attribute is not implemented on the A Series. However, the function can be imitated by by using the EXTEND option of the OPEN statement of COBOL74.
FILE.TYPE	The A Series equivalent is FILEKIND. The A Series has approximately 200 different file types. The list of FILEKINDs is in the "A Series I/O Subsystem Reference Manual."
FOOTING	This attribute is not supported on the A Series.
FORMS	The A Series equivalent is FORMID. FORMID can be set to a string containing up to 100 characters. It is reset by specifying a NULL string. If FORMID is set, the message assigned to the attribute is displayed on the operator's console at file open time (on-line printing) or at print time (backup files). If there is no printer with the requested FORMID, the program will be suspended until the operator responds with a Form Message (FM) or an Output Unit (OU) system input message. For more information about FM and OU system input messages, refer to the "A Series ODT Reference Manual."
HARDWARE	This function can be controlled by MCP options. On the A Series, the preferred method of setting the hardware device is to reset the CPBDONLY (for card punches) or the LPBDONLY (for line printers) option

	using the A Series ODT OPTION (OP-CPBDONLY or OP-LPBDONLY) command. At the same time, the BACKUPKIND attribute must be equal to DONTCARE, which uses the system default. For more information about the OPTION command, refer to the "ODT Messages" section in the "A Series Operator Display Terminal (ODT) Reference Manual."
HEADER	There is no A Series equivalent.
IMPLIED.OPEN	There is no A Series equivalent.
INCREMENT.EU	The A Series SINGLEUNIT attribute is similar. SINGLEUNIT does not specify to which disk drive the areas should be written, instead it indicates whether areas for the disk file are to be allocated from a single family member (pack). The default, FALSE, distributes areas over the entire family (multiple packs). If SINGLEUNIT = TRUE, all areas for the file are allocated on a single diskpack.
INPUT	The A Series equivalent is MYUSE = IN. MYUSE = IN specifies that the file is opened INPUT.
INPUT.SELECTIVITY	This attribute is not implemented on the A Series.
INTERPRETER	This attribute is not applicable on the A Series.
INVALID.CHARACTER	This attribute has no A Series equivalent.
LABEL.TYPE	The A Series equivalent is LABEL. The Small Systems label type ANSI must be changed to STANDARD for the A Series, and UNLABELLED changed to OMITTED or OMITTEDEOF.
LINE FORMAT	This attribute is not implemented on the A Series.
LOCK	The A Series PROTECTION attribute is similar. With PROTECTION set to PROTECTED, an entry is immediately made in the disk directory when the file is opened. As the disk areas are allocated, they are encoded with a pattern which makes it possible to

	discover the last valid block written on that area in the event of a Halt/Load.
LOWER.MARGIN	This attribute is not implemented on the A Series.
MAXIMUM.BLOCK.SIZE	The A Series equivalent is BLOCKSIZE. If the UNITS attribute equals CHARACTERS, BLOCKSIZE is given in INTMODE units, otherwise, it is specified in words.
MAXRECSIZE	The A Series equivalent is MAXRECSIZE. MAXRECSIZE may be assigned a value from 0 to 65,535, inclusive.
MAXSUBFILES	The A Series equivalent is MAXSUBFILES.
MINRECSIZE	The A Series equivalent is MINRECSIZE. MINRECSIZE may be assigned a value from 0 to 65,535, inclusive.
MULTI.PACK	The A Series equivalent is SINGLEUNIT. See the "File Naming Conventions" section, earlier in this manual.
MYNAME/MY.NAME	The A Series equivalent is MYNAME.
NAME	The A Series equivalent is TITLE. See the "File Handling" section, earlier in this manual.
NEW	The A Series equivalent is NEWFILE. If NEWFILE is TRUE, a new file is created. If NEWFILE is FALSE, an existing file is sought. If NEWFILE is not specified, the MYUSE attribute is used to determine if a new file is created or if an existing file is sought. For a detailed explanation, refer to the "New File vs. Permanent File" discussion and the NEWFILE and MYUSE attributes in the "A Series I/O Subsystem Reference Manual."
NO, NOT	On Small Systems, NO/NOT negates the file attribute following it. For example, NO BACKUP. On the A Series, to set, reset, reverse, or change the status of a file attribute, place the appropriate value with the corresponding attribute. On the

	A Series, these values may be Boolean, numeric, etc.
	Examples are PARITY = NONSTANDARD, KIND = DISK, NEWFILE = FALSE, and FILEUSE = IN.
NUMBER.STATIONS	There is no A Series equivalent for this attribute since the A Series has no limit on the number of stations that can be attached to a remote file.
ODD	The A Series equivalent is PARITY = STANDARD or PARITY = ODD.
OPEN.LOCK	This attribute is not supported on the A Series.
OPEN.LOCKOUT	The A Series equivalent is EXCLUSIVE = TRUE. EXCLUSIVE allows a program to open a permanent disk file and lock out all other programs and unopened files while the permanent file is open.
OPTIONAL	The A Series equivalent is OPTIONAL.
OUTPUT	The A Series equivalent is MYUSE=OUT. MYUSE = OUT specifies that the file is opened OUTPUT.
PACK.ID	The A Series equivalent is FAMILYNAME. See the "File Handling" section, earlier in this manual.
PAGE.SIZE	The A Series equivalent is PAGESIZE. It indicates the number of lines on a logical page.
PAPER.TAPE.PUNCH	The A Series equivalent is KIND = PAPERPUNCH.
PAPER.TAPE.READER	The A Series equivalent is KIND = PAPERREADER.
PORT.FILE	The A Series equivalent is KIND=PORT.
PORT.KEY (BNA)	There is no A Series equivalent. However, when the A Series attribute MAXSUBFILES is greater than one, this implies the same thing as the PORT.KEY on Small Systems.

PRINTER	The A Series equivalent is KIND = PRINTER.
PRINTER.5	This attribute is not supported on the A Series.
PROTECTION	The A Series equivalent is PROTECTION. The Small Systems ABNORMALSAVE option must be changed to PROTECTED on the A Series.
PROTOCOL	There is no A Series equivalent.
PSEUDO	There is no A Series equivalent.
QUEUE	Queue files are not implemented on the A Series. See the "Queue/Port Files" section, later in this manual.
Q.FAMILY.SIZE	There is no A Series equivalent for this attribute.
Q.MAX.MESSAGES	There is no A Series equivalent for this attribute.
RANDOM	This attribute is not applicable on the A Series. However, the A Series does allow SEQUENTIAL and RANDOM access depending on the access method used in the program.
READER.PUNCH.PRINTER	This attribute is not supported on the A Series.
READER.SORTER	This attribute is not supported on the A Series.
READER.SORTER.STATIONS	This attribute is not supported on the A Series.
READER.SORTER.2	This attribute is not supported on the A Series.
READER.96	This attribute is not supported on the A Series.
RECORDS.BLOCK	The A Series equivalent is BLOCKSIZE. BLOCKSIZE is the value of RECORD.SIZE * RECORDS.BLOCK. If BLOCKSIZE is less than MAXRECSIZE, BLOCKSIZE is set to MAXRECSIZE when the file is opened.
RECORD.SIZE	The A Series equivalent is MAXRECSIZE. MAXRECSIZE specifies maximum size of the

	record. If the UNITS attribute equals CHARACTERS, MAXRECSIZE is expressed in INTMODE units, otherwise, it is specified in words.
REEL	There is no A Series equivalent for this attribute.
REMOTE	The A Series equivalent is KIND = REMOTE.
REPETITIONS	The A Series equivalent is PRINTCOPIES.
REVERSE	The A Series equivalent is DIRECTION = REVERSE. The attribute DIRECTION on A Series indicates the direction in which records are accessed from a tape or paper tape file.
REWIND	This attribute is not implemented on the A Series. However, some languages, such as COBOL74, provide the functional equivalent of this attribute in the OPEN statement.
SAVE	The A Series equivalent is SAVEFACTOR. On the A Series, the SAVEFACTOR is the expiration date of the file denoted by the number of days past creation date. This attribute is useful mainly for tape files. SAVEFACTOR must be set for tape files otherwise they will be purged. For more information about A Series SAVEFACTOR, refer to the "A Series I/O Subsystem Reference Manual."
SECURITYTYPE	The A Series equivalent is SECURITYTYPE. SECURITYTYPE specifies which users, apart from the owner (creator) of a permanent disk file (as identified by the usercode), may access the disk or pack file. The A Series SECURITYTYPE has the additional mnemonic GUARDED.
SECURITYUSE	The A Series equivalent is SECURITYUSE. SECURITYUSE is the I/O access rights permitted for a physical disk or pack file. SECURITYUSE on the A Series has the additional mnemonic SECURED.
SENDALL	The A Series has no equivalent for this attribute.

SEQUENTIAL	The attribute is not applicable on the A Series. However, the A Series does allow SEQUENTIAL and RANDOM access, depending on the access method used in the program.
SERIAL	This attribute is not applicable on the A Series. On the A Series, if the access method of a file in the program is declared as SEQUENTIAL, the file will be processed sequentially.
SERIAL.NUMBER	Refer to the "File Handling" section, earlier in this manual.
SIMPLE.HEADERS	There is no A Series equivalent for this attribute.

STATION/STATIONS	The A Series STATIONLIST attribute provides a similar function. The STATIONLIST attribute allows you to dynamically alter the stations associated with an open data comm file. On the A Series, stations may only be added programmatically.
	
	Though STATION/STATIONS and STATIONLIST are not directly equivalent, they can serve the same function. For example, on the Small Systems you would enter:
	
	`EX X; STA = A1, B1, C1`
	
	at the ODT. On the A Series, you would include:
	
	`CHANGE ATTRIBUTE STATIONLIST OF <filename> TO <station-name>.`
	
	in your COBOL74 program.

TAPE	The A Series equivalent is KIND = TAPE.
TAPE.NRZ	The A Series equivalent is KIND=TAPE with DENSITY=BPI800. You must set both the KIND and the DENSITY attribute to achieve the desired result.
TAPE.PE	The A Series equivalent is KIND = TAPEPE.
TAPE.7	The A Series equivalent is KIND = TAPE7.
TAPE.9	The A Series equivalent is KIND = TAPE9.

File and Program Attributes

TRANSLATE	The A Series equivalent is TRANSLATE. The values, mnemonics, and meanings of the attributes differ from the Small Systems. For a detailed explanation of the A Series TRANSLATE attribute and a description of A Series software translation, refer to the "A Series I/O Subsystem Reference Manual."
TRANSLATE.NAME	This attribute is not available on the A Series.
UNIT.NAME	The A Series equivalent is UNITNO. Peripheral units are specified by number on the A Series. For example, to send a file to tape drive MT14, UNITNO must be set to 14.
UNLABELLED	The A Series equivalent is LABEL = OMITTED. To indicate the file does not have a label record, set LABEL equal to OMITTED or OMITTEDEOF.
UPPER.MARGIN	This attribute is not implemented on the A Series.
USER.BACKUP.NAME	The A Series equivalent is USERBACKUPNAME. The default value of USERBACKUPNAME is FALSE, in which case the output printer backup file name will be BD/<job number>/<task number>/000<internal file name>. A backup file will default to the backup pack specified through the ODT commands Disk Location (DL) and Substitute Backup (SB).
VARIABLE	The A Series equivalent is BLOCKSTRUCTURE = VARIABLE. By specifying BLOCKSTRUCTURE equal to VARIABLE, the file is processed using variable length records.
WITH.INTERPRET	This attribute is not available on the A Series.
WITH.PRINT	This attribute is not available on the A Series.
WITH.PUNCH	This attribute is not available on the A Series.

WITH.STACKERS	This attribute is not available on the A Series.
WORK.FILE	This attribute is not available on the A Series.

STANDARD FILE ATTRIBUTES

The following is a list of Small Systems standard file attributes (implemented in COBOL74, WFL, and SDL2) and their A Series equivalents. These attributes can define, monitor, or change the properties or attributes of a file. For more information about these A Series attributes, refer to the "A Series I/O Subsystem Reference Manual."

Small Systems	A Series
AREALENGTH	The A Series equivalent is AREALENGTH.
AREAS	The A Series equivalent is AREAS.
ATTERR	The A Series equivalent is ATTERR.
AUDITED	This attribute is not available on the A Series.
BACKUPKIND	The A Series equivalent is BACKUPKIND with the additional mnemonic values TAPE7, TAPE9, and TAPEPE.
BACKUPPERMITTED	The A Series equivalent is PRINTDISPOSITION. Setting PRINTDISPOSITION to DIRECT is equivalent to NO BACKUPPERMITTED. Any other value is equivalent to BACKUPPERMITTED.
BLOCK	The A Series equivalent is BLOCK.
BLOCKSIZE	The A Series equivalent is BLOCKSIZE.
BLOCKSTRUCTURE	The A Series equivalent is BLOCKSTRUCTURE with the additional mnemonic values of EXTERNAL and LINKED.
BUFFERS	The A Series equivalent is BUFFERS. The default setting on the A Series is 2.
CENSUS	See the "Queue/Port Files" section.

File and Program Attributes

CHANGEDSUBFILE	See the "Queue/Port Files" section.
CREATIONDATE	The A Series equivalent is CREATIONDATE. The CREATIONDATE is specified with a 5-digit integer in the Julian format YYDDD.
CURRENTBLOCK	The A Series equivalent is CURRENTBLOCK.
COMPRESSION	The A Series equivalent is COMPRESSION. For information, refer to the "Queue/Port Files" section.
DENSITY	The A Series equivalent is DENSITY.
DEPENDENTSPECS	The A Series equivalent is DEPENDENTSPECS. By setting DEPENDENTSPECS to TRUE, the structure of the physical file is assumed by the logical file.
DIRECTION	The A Series equivalent is DIRECTION.
EXTMODE	The A Series equivalent is EXTMODE with the additional values SINGLE, HEX, and BCL.
FILEKIND	The A Series equivalent is FILEKIND. There are over 200 possible values for FILEKIND on the A Series. For information about FILEKIND, see the "A Series I/O Subsystem Reference Manual."
FILESECTION	The A Series equivalent is FILESECTION.
FILESTATE	See the "Queue/Port Files" section.
FRAMESIZE	The A Series equivalent is FRAMESIZE. On the A Series, FRAMESIZE can be 4 hexadecimal characters, 8 EBCDIC or ASCII characters, or one 48-bit word. A word is the equivalent of six bytes and is the main memory unit on the A Series.
HOSTNAME	The A Series equivalent is HOSTNAME.
INTNAME	The A Series equivalent is INTNAME.
KIND	The A Series equivalent is KIND with the additional values DONTCARE, REMOTE, and DC.
LABEL	The A Series equivalent is LABEL with the additional mnemonic values STANDARD and OMITTEDEOF.

LASTRECORD	The A Series equivalent is LASTRECORD.
MAXCENSUS	The A Series equivalent is MAXCENSUS. For information about MAXCENSUS, see the "Queue/Port Files" section.
MAXRECSIZE	The A Series equivalent is MAXRECSIZE.
MAXSUBFILES	The A Series equivalent is MAXSUBFILES. For information about MAXSUBFILES, see the "Queue/Port Files" section.
MINRECSIZE	The A Series equivalent is MINRECSIZE.
MYHOSTNAME	See the "Queue/Port Files" section.
MYNAME	See the "Queue/Port Files" section.
MYUSE	The A Series equivalent is MYUSE with the additional value CLOSED.
NEWFILE	The A Series equivalent is NEWFILE.
NEXTRECORD	This attribute is not implemented on the A Series.
OPEN	The A Series equivalent is OPEN.
OPTIONAL	The A Series equivalent is OPTIONAL.
OTHERUSE	The A Series equivalent is EXCLUSIVE. The A Series attribute produces results similar to the Small Systems attribute OTHERUSE = SECURED by setting EXCLUSIVE equal to TRUE.
PARITY	The A Series equivalent is PARITY.
RECORD	The A Series equivalent is RECORD. The A Series RECORD attribute is zero-relative, rather than one-relative.
SAVEFACTOR	The A Series equivalent is SAVEFACTOR. SAVEFACTOR must be set for tape files otherwise they will be purged. For more information about the A Series SAVEFACTOR, refer to the A Series I/O Subsystem Reference Manual.
SECURITYTYPE	The A Series equivalent is SECURITYTYPE. SECURITYTYPE is the same on the A Series as

	on Small Systems with the additional mnemonic values GUARDED and CONTROLLED.
SERIALNO	The A Series equivalent in SERIALNO.
TITLE	See the "File Handling" section, earlier in this manual.
TRANSLATE	The A Series equivalent is TRANSLATE, with the additional mnemonics DEFAULTTRANS, FULLTRANS, SOFTONLY, FORCESOFT, NOSOFT, and NOTRANS.
TRANSLATING	The A Series equivalent is TRANSLATING. The A Series TRANSLATING attribute is read-only.
UPDATEFILE	The A Series equivalent is UPDATEFILE.
USEDATE	The A Series equivalent is USEDATE.
VOLUMEINDEX	This attribute is not implemented on the A Series.
YOURNAME	See the "Queue/Port Files" section.
YOURUSERCODE	See the "Queue/Port Files" section.

PROGRAM ATTRIBUTES

Program attributes are system control parameters used by the MCP. They are used to control the behavior and environment of a task or job before execution, during execution, and after execution. The following is a list of the Small Systems Program attributes and their equivalent A Series task attributes.

Small Systems	A Series
AFTER	This attribute is replaced by the appropriate WFL statements on the A Series. See the "Work Flow (Jobs)" section, earlier in this manual.
AFTER.NUMBER	This attribute is replaced by the appropriate WFL statements on the A Series. See the "Work Flow (Jobs)" section, earlier in this manual.

CHARGE	The A Series equivalent is CHARGECODE. The A Series CHARGECODE value must be alphanumeric.
CONDITIONAL	This attribute is replaced by the appropriate WFL statements on the A Series. See the "Work Flow (Jobs)" section, earlier in this manual.
DYNAMIC.SPACES	There is no A Series equivalent of this attribute since the A Series automatically assigns additional dynamic memory.
FREEZE	There is no A Series equivalent for this attribute.
HOLD	There is no A Series equivalent for this attribute. This function is done on the A Series through the Work Flow Language. A QUEUE statement may be placed in the WFL job deck where the QUEUE has a MIXLIMIT of 0. This would cause the program to wait in the queue until it is forced out using the ODT Force Schedule (FS) command.
INTERPRETER	This attribute is not applicable on the A Series.
INTRINSIC.DIRECTORY	This attribute is not applicable on the A Series.
INTRINSIC.NAME	This attribute is not applicable on the A Series.
LEVEL	There is no A Series equivalent for this attribute.
MAXWAIT	The A Series equivalent is MAXWAIT. MAXWAIT cannot be abbreviated on the A Series.
MEMORY	There is no A Series equivalent for this attribute. The A Series automatically allocates memory to the program as required.
MEMORY.PRIORITY	The A Series equivalent is PRIORITY. A Series has only one type of PRIORITY and it applies to both the memory and the processor.

File and Program Attributes

NODIF	There is no A Series equivalent for this attribute.
OBJ	Any file or task attribute assignment specified on an A Series compile is applied to the object code of the compile. By preceding the file or task attribute assignment with the word COMPILER, the file or task attribute is applied to the compiler itself. For example,

Small Systems (Control Card with OBJ)

```
COMPILE OBJPROG COBOL74 LIBRARY;
FILE CARD A/B DISK DEF;
OBJ FILE SPECFILE BUFFERS = 2 DISK
```

A Series example (WFL)

```
COMPILE OBJPROG COBOL74 LIBRARY;
COMPILER FILE CARD (KIND = DISK,
  TITLE = A/B,
  DEPENDENTSPECS = TRUE);
FILE SPECFILE (BUFFERS =2, KIND = DISK);
```

OVERRIDE	This attribute is not applicable on the A Series.
PRIORITY	The A Series equivalent is PRIORITY. The values for the A Series PRIORITY range from 0 to 99, with a default of 50.
PROCESSOR.PRIORITY	The A Series equivalent is PRIORITY. There is only one type of PRIORITY on the A Series and it applies to both processor and memory.
PROTECTED	The A Series equivalent is the Lock Program (LP) ODT command. The LP command prevents the use of the DISCONTINUE (DS) and QUIT (QT) command from interfering with the program execution.
RR	There is no A Series equivalent for this attribute.

SCHEDULE.PRIORITY	The A Series equivalent is PRIORITY. There is only one type of priority on the A Series and it applies to both processing and memory.
SECONDS.BEFORE.DECAY	The A Series provides similar functions through the SET FACTOR (SF) ODT command. See the "A Series ODT Reference Manual" for information about the SF command.
SLAVE.OK	A similar A Series attribute is SUBSYSTEM. SUBSYSTEM specifies the subsystem on which the task is to run or is running. This attribute may be set only when the task is inactive and is not automatically inherited by descendent tasks. SUBSYSTEM is applicable only on tightly-coupled and ASN systems (generally this is true of systems with more than 6M bytes of memory.) For more information about the A Series attribute SUBSYSTEM, refer to Appendix A of the "A Series WFL Reference Manual."
SWITCH	The A Series SWITCH attribute is similar. On the A Series, the switches are numbered from one to eight, and can be set to a Boolean value of TRUE or FALSE.
SYMBOLIC.QUEUE.NAME	The A Series equivalent is STATIONLIST.
THEN	This attribute is replaced by the appropriate WFL statement on the A Series. See the "Work Flow (Jobs)" section, earlier in this manual.
TIME	The A Series equivalent for this attribute is MAXPROCTIME. MAXPROCTIME is specified in seconds, not minutes.
UNCONDITIONAL	This attribute is replaced by the appropriate WFL statements on the A Series. See the "Work Flow (Jobs)" section, earlier in this manual.
UNFREEZE	This attribute is not applicable on the A Series.
UNOVERRIDE	This attribute is not applicable on the A Series.

VIRTUAL.DISK	This attribute is not applicable on the A Series. The size of the overlay area is controlled by either the default value of the Overlay Row size or by the value specified for the Overlay Row Size at coldstart time. The Overlay Row Size can not be set for each program.

6 TRANSFERRING FILES

Because of the difference between Small Systems and the A Series, it is generally not possible to simply dump files from the Small Systems and then load them on the A Series. This section describes the four methods of transferring files.

SOME NOTES BEFORE YOU START

You do not need to change the data format between the two systems. Any data and sign format that is accepted by Small Systems is also accepted by the A Series.

File formats for some of the files are different between the two systems and may have to be changed.

RPG ADDROUT files cannot be transferred; they must be recreated by the A Series SORT program.

Previous to the 11.0 release on Small Systems, all program sources were FILEKIND=DATA. On Small Systems starting with the 11.0 release, program source FILEKINDs have been implemented. The implemented FILEKINDs are:

COBOL(68) SYMBOL	RPG SYMBOL
NDL SYMBOL	FORTRAN SYMBOL
BASIC SYMBOL	COBOL74 SYMBOL
FORTRAN77 SYMBOL	IBASIC SYMBOL
DASDL SYMBOL	PASCAL SYMBOL
SORT SYMBOL	SEQ DATA
JOB SYMBOL	

Each program source file that has one of the above FILEKINDs will be loaded with the appropriate A Series CANDE-compatible FILEKIND and BLOCKING factor.

CHOOSING THE FILE TRANSFER METHOD

There are four ways of transferring files between Small Systems and the A Series:

1. B1000COPY

2. B6000COPY

3. Data Base Data Transfer Utility

4. Progammatically

The list below matches the recommended method of transferring files with the file types.

File Types	Method of File Transfer
Sequential files	B1000COPY or B6000COPY
Source files	B1000COPY or B6000COPY
ISAM files	Programmatically
DMSII files	Data Base Data Transfer Utility
Relative files	Programmatically
Files with missing areas	Programmatically

INTERCHANGE disk packs is not a recommended method for transferring files.

THE B1000COPY METHOD

B1000COPY is an A Series utility program that reads Small Systems SYSTEM/COPY tapes. It is the recommended method of transferring sequential data files and program sources.

Because of their format differences, B1000COPY cannot be used to transfer DMSII files, COBOL74 ISAM files, or relative files. To transfer these types of files, refer to the descriptions for transferring files using "DMSII File Transfer" and "Programmatic Transfer Methods" found later in this section.

B1000COPY can be found on the A Series 3.6 release BTA360 Migration Aids tape.

Transferring Files

<u>Using B1000COPY</u>

B1000COPY requires a string parameter at execution. This parameter is used to identify which of three functions B1000COPY is to perform.

1. A parameter of TEACH or HELP displays instructions about the use of B1000COPY. Use this for additional information about B1000COPY.

2. A parameter of DIR <tapename>, TPDIR <tapename>, or TD <tapename> prints a directory of the Small Systems SYSTEM/COPY tape.

3. A parameter of a COPY statement or an ADD statement prints a directory of the Small Systems SYSTEM/COPY tape and will COPY or ADD the appropriate files from the tape to disk. The COPY and ADD statements are a simple form of the standard COPY and ADD statements.

No special characters are allowed in a name. B1000COPY automatically removes all special characters from the file name on the tape.

To transfer sequential data files, follow these steps:

1. Use the Small Systems SYSTEM/COPY utility to load files from disk to tape.

   The syntax for this step is:

   ```
   COPY DATA/= FROM MYPACK (KIND=DISK) TO MYTAPE(KIND=TAPE);
   ```

2. Mount the tape on the A Series and use B1000COPY to load those files from the tape.

   The syntax for this step is:

   ```
   RUN B1000COPY ("COPY DATA/= FROM MYTAPE TO NEWDISK");
   ```

B1000COPY loads all file types. Files with a recognized FILEKIND (see the above list) are loaded as their A Series equivalents. Files with other FILEKINDs are loaded as FILEKIND=DATA.

To transfer program sources, follow the steps below:

1. Use the Small Systems SYSTEM/COPY utility to copy the source from disk to tape.

   The syntax for this step is:

   ```
   COPY COBOLSRC/= FROM MYPACK (KIND=DISK) TO MYTAPE
       (KIND=TAPE);
   ```

2. Then load the tape on the A Series and enter:

   ```
   RUN B1000COPY ("COPY COBOLSRC/= FROM MYTAPE TO
       SOURCEPACK");
   ```

Missing Areas

Files with missing areas do not transfer correctly using B1000COPY. For example, if a file has 1000 records per area and a program writes record numbers 2000, 4000, and 6000, the file only has areas 2, 4, and 6 defined. The end-of-file pointer is 6000 even though only three records were written. The Small Systems COPY tape has areas 2, 4, and 6 on it. B1000COPY creates a file with 1000 records per area, but it loads those areas into 1, 2, and 3 of the A Series file and marks the end-of-file pointer as 3000. This situation results in a warning message. The message includes the file name and both the original and the new end-of-file pointers. Because of these problems, we recommend transferring files with missing areas using the Programmatic Method described later in this section.

THE B6000COPY METHOD

During the progression process, you may want to transfer files from the A Series to the Small Systems to compare results from parallel runs or to handle situations where only part of the progression is complete. B6000COPY is a Small Systems utility program that reads A Series library maintenance tapes. We recommend you use this method for sequential data files and program sources.

B6000COPY is found on the BTA350 Small Systems tape.

Because of format differences, B6000COPY cannot be used to transfer DMSII, ISAM, or relative files. Refer to "DMSII File Transfer" and "Programmatic Transfer Methods" described later in this section for information about these types of file transfers.

Transferring Files

Using B6000COPY

B6000COPY requires a single ACCEPT message at execution. This message is used to identify which of three functions B6000COPY is to perform:

1. A message of TEACH or HELP displays instructions about the use of B6000COPY. Use this to obtain more information about B6000COPY.

2. A message of DIR <tapename>, TPDIR, <tapename>, or TD <tapename> prints the tape directory.

3. A message of a COPY statement copies the appropriate files from the A Series tape to disk. The COPY statement is a simple form of the standard COPY statement.

The B6000COPY will only copy files that start on the first reel.

To transfer sequential data files and program sources, follow the steps below:

1. Use the Small Systems SYSTEM/COPY utility to load files from disk to tape.

   The syntax for this step is:

   ```
   COPY MYFILES/SOURCE/= AS MYFILES/= FROM MYPACK(KIND=PACK)
        TO MYTAPE (KIND = TAPE);
   ```

2. Then mount the tape on the Small Systems and enter:

   ```
   EX B6000COPY;  AX<mix number>COPY MYFILES/=
        FROM MYTAPE TO MYPACK
   ```

B6000COPY loads all file types. Files with recognized FILEKINDs (see the list in the "Choosing the File Transfer Method" subsection) are loaded as their A Series equivalents. Files with other FILEKINDs are loaded as FILEKIND=DATA.

Missing Areas

Files with missing areas do not transfer correctly using B6000COPY. Information about transferring files with missing areas is given in the "B1000COPY" section and in the "Programmatic Transfer Methods" section.

DMSII FILE TRANSFER

We recommend you use the Data Base Data Transfer Utility (DBT) to help with the Data Management Systems II (DMSII) data transfer. This utility uses the Data and Structure Definition Language (DASDL) and the description file to generate COBOL74 programs that unload the data from the Small Systems DMSII data base and then load the data onto the A Series DMSII data base. All Small Systems data set types (including variable formats and embedded data sets), item types, and embedded manual subsets are transferred.

The Data Base Data Transfer Utility (SYSTEM/DBTGEN) is available on the A Series 3.6 Migration Aids tape. Refer to the Data Base Data Transfer Utility User's Guide for more information.

PROGRAMMATIC TRANSFER METHOD

For those files that cannot be transferred using one of the previously described utilities, write a program on the Small Systems that reads one or more disk files and writes them to tape. Then write a similiar A Series program that reads the tape and creates one or more disk files on the A Series. We recommend using this method for B and Tag style ISAM files, COBOL74 ISAM files, relative files, and files with missing areas.

B And Tag Style ISAM Files

Use SYSTEM/COPY and B1000COPY to transfer the B file or the data portion of the TAG file to the A Series. Next, write an A Series program to read the data file and create an ISAM file. The A Series 3.6 BTA360 Migration Aids tape contains a COBOL74 sample program called LOADISAMS that reads a data file and creates an ISAM file. Refer to the "ISAM Files" section, later in this manual, for more information about A Series ISAM files.

Transferring Files

COBOL74 ISAM Files

To transfer COBOL74 ISAM files, write a Small Systems program that reads the ISAM file and creates a sequential data file. The BTA350 Small Systems tape has a COBOL74 sample program, DUMPISAMS, that reads an ISAM file and creates a sequential file. You can create the sequential file on tape or disk. If you create the file on disk, use SYSTEM/COPY and B1000COPY to transfer the file to the A Series. Then write an A Series program to read the tape or disk sequential file and create an ISAM file. The LOADISAMS sample program on the A Series 3.6 Release BTA360 Migration Aids tape shows how to do this.

Relative Files

To transfer relative files, write a Small Systems program that reads the relative file and creates an intermediate sequential file on tape or disk. You can use the sample COBOL74 program, DUMPRELS, on the BTA350 Small Systems tape as an example.

If you create a disk file, transfer it to the A Series with SYSTEM/COPY and B1000COPY. Then write an A Series program to read the tape or disk intermediate sequential data file and create a relative file. The A Series 3.6 Release BTA360 Migration Aids tape contains LOADRELS, a sample COBOL74 program, which you can use as an example.

Missing Area Files

To transfer files with missing areas, write a Small Systems program that reads the file and writes an intermediate sequential file to disk or tape. Then write an A Series program to read the intermediate sequential file and recreate the file with missing areas. SYSTEM/COPY and B1000COPY can be used to transfer the intermediate disk file from the Small Systems to the A Series. The BTA350 Small Systems tape has a sample COBOL74 program, DUMPRANS, which you can use as an example. This sample program creates the intermediate file. The A Series 3.6 BTA360 Migration Aids tape contains a sample load program called LOADRANS.

7 COBOL(68)

For the following reasons we recommend converting all of the Small Systems COBOL(68) programs to COBOL74 as part of the progression process:

1. COBOL74 is continually being enhanced. No new features are being added to COBOL(68).

2. The COBOL74 ISAM is fully compatible with RPG. The COBOL(68) ISAM cannot be accessed by RPG programs.

3. The COBOL74 ISAM allows file sharing just like the Small Systems. The A Series COBOL(68) ISAM does not allow files to be shared.

4. There is no filter program and no documentation available to assist the progression of Small Systems COBOL(68) to A Series COBOL(68).

5. Any feature available with Small Systems COBOL(68) is either a feature of A Series COBOL74 or is missing from both COBOL(68) and COBOL74 on the A Series.

6. Most of the desirable Burroughs extensions in COBOL(68) have been added to COBOL74.

We also recommend that you use the Burroughs to Burroughs Translator (CTA) filter to assist with the COBOL(68) to COBOL74 conversion. The filter accepts Small Systems COBOL(68) source code and produces A Series COBOL74 source code. It translates most constructs and clearly flags those it doesn't. The CTA filter is located on the A Series 3.6 Release BTA360 Migration Aids tape.

8 COBOL74

The Small Systems COBOL74 is basically a subset of the A Series COBOL74. Therefore, most of the COBOL74 programs should easily translate to the A Series. To assist with the conversion of the COBOL74 programs, the Burroughs B 1000 Series COBOL74 to Burroughs A Series and B 5000/B 6000/B 7000 Series translator (B7T) is located on the A Series 3.6 Release BTA360 Migration Aids tape.

This section contains information about the general COBOL74 conversion, using the translator, and how to manually change those Small Systems constructs that cannot be changed by the translator.

GENERAL COBOL74 CONVERSION INFORMATION

The COBOL syntax for file attributes is the same on the two systems, but the file attributes are often different. Some of these differences are discussed here; for more information, see the "File and Program Attributes" section.

Hexadecimal Literals

There are no differences in the handling of hexadecimal literals for A3s, A9s, and A10s running LEVEL1 code (the default for programs compiled on these machines). If you have an A3, an A9, or an A10 and are running LEVEL1 code, skip to the next topic in this section.

For B5000, B6000 and B7000 machines and any A3, A9, or A10 running LEVEL0 code, there are significant differences between the handling of hexadecimal literals on the A Series and the Small Systems. To help convert these hexadecimal constructs, the following information explains how the A Series treats certain data types and why some code produces incorrect results.

To describe a computational (4-bit) number, the code produced by the compiler does not include any indication of the presence or absence of a sign. The code (or descriptor) for PIC 9999 COMP is:

```
    4-bit, starting at location L1, 4 long
```

The descriptor for PIC S999 COMP is:

4-bit, starting at location L2, 3 long

If you assume that the first item has a value of 1234 and the second has a value of -123, everything works correctly. When the system evaluates the first descriptor, it looks at the first four bits. Since the number there is between 0 and 9, the hardware assumes that it is unsigned and picks up the next four digits, creating the number 1234. For the second descriptor, the hardware looks at the first four bits starting at L2 and notices that the digit there (the minus sign is D) has a value between A and F. It therefore assumes that the number is signed and picks up the next three digits, creating the number -123.

Problems occur when bad data is present in these two memory locations. For an example, assume the data in the first number is -123 and the second has the value 1234. When the hardware looks at the first descriptor, it sees a value of D (the minus sign) in the first digit and assumes that the number is signed and 4 digits long (excluding the sign). The hardware takes the numbers 123 and whatever digit follows the 3 in memory. If the 3 is at the "end" of memory for this structure, the program is DSed with a SEG ARRAY ERROR (or similar message). If there happens to be data after the 3, this data is picked up and used.

In the preceding example, if the two numbers were described one after the other within an 01 level, the first number would end up with a value of -1231, with the second 1 coming from the first digit of the second number.

For the second descriptor, the hardware looks at the first digit and decides that the number is unsigned and creates a number with a value of 123.

Remember, this problem only occurs with COMPUTATIONAL numbers. The examples below show the most common uses of hexadecimal literals. Included with the examples are methods for converting the code.

Example 1

```
01  SCREEN-FORMAT.
  02    FILLER          PIC 99     COMP  VALUE FF.
  02    FILLER          PIC 9999   COMP  VALUE OCOC.
```

COBOL74

This code does not have to be changed because these items are never referenced. However, we recommend changing the picture clauses to PIC X and PIC XX, respectively.

Example 2

```
01  SOME-DATACOMM-CONSTANTS.
   02     HOME-CLEAR        PIC 99      COMP  VALUE OC.
   02     FORMS-ON          PIC 9999    COMP  VALUE 27E6.
01  SCREEN.
   02     ONE-CHARACTER     PIC 99      COMP  OCCURS 1920 TIMES.

          MOVE HOME-CLEAR TO ONE-CHARACTER (1).
```

Because these data items are referenced, they must be changed. The fields used with them must also be changed. The changed code is shown below:

```
01  SOME-DATACOMM-CONSTANTS.
   02     HOME-CLEAR        PIC X             VALUE OC.
   02     FORMS-ON          PIC XX            VALUE 27E6.
01  SCREEN.
   02     ONE-CHARACTER     PIC X             OCCURS 1920 TIMES.

          MOVE HOME-CLEAR TO ONE-CHARACTER (1).
```

Example 3

```
01  SCREEN-INPUT.
   02     FIRST-TWO         PIC 9999    COMP.

         IF FIRST-TWO IS EQUAL TO 277F
```

This code is changed to:

```
01  SCREEN-INPUT.
   02     FIRST-TWO         PIC XX.

         IF FIRST-TWO IS EQUAL TO 277F
```

Example 4

```
     01  X.
        02     A                PIC 9      COMP.
        02     FILLER           PIC 9(9)   COMP.
     77  V                      PIC 9      COMP  VALUE E.

              IF A IS EQUAL TO V OR
                 A IS EQUAL TO B
```

The code is changed to:

```
     01  X.
        02     X1.
           03     A               PIC 9     COMP.
           03     FILLER          PIC 9(9)  COMP.
        02     X2  REDEFINES X1.
           03     A-H             PIC X.
           03     FILLER          PIC 9(8)  COMP.
     01  SOME-HEX-VALUES.
        02     HEX-AF             PIC X           VALUE AF.
        02     HEX-CO             PIC X           VALUE CO.
        02     HEX-DF             PIC X           VALUE DF.
        02     HEX-FO             PIC X           VALUE FO.

            IF (A-H GREATER THAN HEX-DF AND A-H LESS
                 THAN HEX-FO)

            OR

            (A-H GREATER THAN HEX-AF AND A-H LESS
                 THAN HEX-CO)
```

Example 5

```
     01  TR-CODE                  PIC 99    COMP.
        88     A                                  VALUE 7B.
        88     B                                  VALUE A5.
           IF A OR B
```

This code is changed to:

```
     01  TR-CODE                  PIC X.
        88     A                                  VALUE 7B.
        88     B                                  VALUE A5.
            IF A OR B
```

Example 6

```
01  DATA-BUFFER.
   02    ONE-DIGIT          PIC 9     COMP  OCCURS 200 TIMES.
77  HEX-F                   PIC 9     COMP  VALUE F.

       IF ONE-DIGIT (I) EQUALS HEX-F OR
          ONE-DIGIT (I) EQUALS A
```

If every digit within DATA-BUFFER is in the range 0 through 9 and all comparisons are made to digits in the range 0 through 9, then the code works correctly. If any of the digits in the buffer is in the range A through F, then the generated code does not produce the desired results. Because the A Series COBOL74 compiler does not support the Small Systems extension to ANSI COBOL74, you cannot convert this program using the A Series COBOL74. Instead, convert this function on the A Series using ALGOL.

Non-numeric Arithmetic

There is also a difference in the treatment of arithmetic dealing with non-numeric digits within numbers. Note that both Small Systems and the A Series documentation state that the results of such arithmetic are undefined. However, the A Series results are different from the Small Systems results. We suggest that you avoid using non-numeric arithmetic constructs because the results are unpredictable.

Review the IS NUMERIC clause in the A Series COBOL ANSI-74 Reference Manual for help in dealing with non-numeric digits. If you have any questions about your data, this clause allows you to check the data accurately before using it. The same rules apply to both the Small Systems and the A Series.

Example

```
77  A           PIC S99    COMP.
77  B           PIC 99     COMP.
77  C           PIC S99.
77  D           PIC 99.

   IF A IS NUMERIC         True only if the sign digit is
                           a C, D, or F and all other digits
                           are 0-9.
```

IF B IS NUMERIC	True only if all the digits are 0-9.
IF C IS NUMERIC	True only if the sign zone is a C, D, F, all other zones are F, and all digits are 0-9.
IF D IS NUMERIC	True only if all zones are F and all digits are 0-9.

HIGH-VALUE, LOW-VALUE

Small Systems always treat HIGH-VALUE as hexadecimal Fs and LOW-VALUE as hexadecimal 0s (zeroes). The A Series treats the values differently depending upon the type of the associated data item. Display numeric data items have high and low values of F9 and F0, respectively. All other data types on the A Series have values of hexidecimal FF and 00.

Default Sign Position

Small Systems have implemented left-signed as the default for both display and computational data. The A Series has implemented a default sign position of left-signed for computation data and right-signed for display data.

There are two ways to handle this difference: change the data or change the program. Changing the location of the sign usually causes a number of problems. Therefore, we recommend that the data be left as it is and the programs be changed. After the progression is complete and all the programs are working, consider changing some of the data. However, make sure the performance improvement is going to be worth the effort; unless the program does a great deal of arithmetic, it probably is not worth the effort. If the program does a lot of arithmetic, we recommend you use BINARY data.

To change the programs, add the following code:

```
SPECIAL NAMES.
  DEFAULT DISPLAY SIGN IS LEADING.
```

This change is automatically done by B7T if the clause is not already present in the code.

TASKVALUE

The A Series offers a task attribute called TASKVALUE that can be set by either the program, the operator, or WFL. It can be read by WFL or the program.

To access TASKVALUE in a COBOL74 program, use the following code:

```
MOVE ATTRIBUTE TASKVALUE OF MYSELF TO
  <numeric data name>

or

CHANGE ATTRIBUTE TASKVALUE OF MYSELF TO
  <numeric data name>
```

The largest possible value for TASKVALUE is 549,755,813,887. The normal COBOL rules of truncation apply when TASKVALUE is larger than the receiving data name. For example, moving a TASKVALUE of 1234 to a PIC 99 item results in a value of 34.

TASKVALUE can be set with a task equate clause in the RUN statement. For example, to set the TASKVALUE to 1234, enter:

```
RUN <program name>; TASKVALUE = 1234
```

To change the TASKVALUE of a running program, enter:

```
<mix number> HI <number>
```

For example, to set the TASKVALUE to 5678 for mix number 1234, enter:

```
1234 HI 5678
```

The HI causes the task attribute EXCEPTIONEVENT (see "Switches" later in this section).

A WFL program can query the TASKVALUE of a program. This allows the queried program to communicate something back to the WFL program.

Example

```
IF T(TASKVALUE) = 1 THEN
ABORT "The edit is out of balance";
```

COBOL DIVISIONS

Some of the COBOL divisions require changes when you progress from Small Systems to the A Series. The following sections discuss each COBOL division and the required changes, if any, for that division.

Identification Division

No changes are required.

Environment Division

Changes are required in the OBJECT-COMPUTER and FILE-CONTROL paragraphs.

OBJECT-COMPUTER

The record size, block size, area length, and areas clauses of an SD are ignored by the A Series. Therefore, to specify the size of a SORT workfile, it must be entered with a DISK SIZE clause in the OBJECT-COMPUTER paragraph. The syntax for this is:

```
DISK SIZE IS <integer> WORDS
```

To calculate the integer needed for this clause:

1. Take the sort record size, in bytes, divide it by 6 (6 bytes per word) and round it up to the nearest word.

2. Multiply that number by the number of records in the file.

3. Multiply the result of step 2 by 2.25.

For example, assume you have a file of 50,000 93-byte records. The record size in words is 93/6, which equals 15.5. The 15.5 is rounded up to 16, then multiplied by 50,000 (the number of records in the file) for a total of 800,000. 800,000 is then multiplied by 2.25 for the result of 1,800,000.

The default disk size is 900,000 words. This is sufficient to sort approximately 13,000 180-byte records.

For more information about the amount of disk work space needed, refer to the subsection titled "Disk Size" in the SORT section of the "A Series Systems Software Utilities Reference Manual."

FILE-CONTROL

The A Series uses port files in place of the Small Systems queue files. Refer to the "Queue/Port Files" section later in this manual for details about this change.

Data Division

The FD and SD paragraphs require changes when you progress from Small Systems to the A Series.

FD

For a file opened INPUT or I/O on Small Systems, if there is no BLOCK CONTAINS clause, the DEFAULT bit is set in the FPB. This sets the block size and record size according to the physical file characteristics when the file is opened. If there is a BLOCK CONTAINS clause, this bit is not set on the Small Systems.

On the A Series, this bit is never automatically set by the compiler. To set it, the following must be added to your code:

```
VALUE OF DEPENDENTSPECS IS TRUE
```

This clause is automatically added by B7T350 when the BLOCK CONTAINS clause is missing, or you can add this clause manually.

SD

The BLOCK CONTAINS and VALUE OF clauses are ignored. See "OBJECT-COMPUTER" earlier in this section.

Procedure Division

Changes will probably be required to some portion of your Procedure Division. General changes include switches and segmentation. Changes in verbs include ACCEPT, CALL, CLOSE, DISPLAY, IF, INDEX, MOVE, OPEN, PERFORM, SORT, and WAIT.

Switches

The A Series has implemented switches as task attributes. This has one advantage and one disadvantage.

Because switches are task attributes and because COBOL74 on the A Series can set task attributes, an A Series COBOL74 program can change the value of a switch from within the program. (This capability existed on Small Systems COBOL(68), but not on Small Systems COBOL74.) The syntax to change a switch is:

```
     CHANGE ATTRIBUTE SWn OF MYSELF TO TRUE (or FALSE).
```

The disadvantage of having switches as task attributes is performance. Accessing a switch on the A Series COBOL74 is 10 times slower than on Small Systems. Usually the slower processing time is not noticeable. However, if the program has a loop which tests one or more switches during every pass of the loop, the reduction in performance can be significant.

One possible solution is to declare pseudo switches in working storage and set them at the beginning of the program. Then change all references from real switches to pseudo switches. However, this does not allow the operator to change the value of the switches once the program is running.

Another solution is to change the program to use an interrupt procedure to perform the action triggered by a switch. For example, one of the most common switches found in a program loop is a status inquiry switch. The program checks the switch on every pass through the loop. If the switch is equal to 1, the program displays its current status and resets the switch to 0. The following code accomplishes this on the A Series:

```
DECLARATIVES.
STATUS-INTERRUPT SECTION.
    USE AS INTERRUPT PROCEDURE.
STATUS-INTERRUPT-PARA.
    DISPLAY "HELLO.  MY STATUS IS ...........".
END DECLARATIVES.
```

Then place the following code at the beginning of the Procedure Division:

```
ATTACH STATUS-INTERRUPT TO
    ATTRIBUTE EXCEPTIONEVENT OF MYSELF.
ALLOW STATUS-INTERRUPT.
```

When the operator enters: <mix number> HI.

The program responds with: HELLO. MY STATUS IS ....

Additional information about this use of switches is available in the ODT-INPUT-OUTPUT discussion of the WAIT statement found later in this section and in the TASKVALUE discussion earlier in this section.

Segmentation

Physical segmentation is automatically provided by the A Series compiler. Sections can be split into multiple segments. Non-contiguous segments are not gathered into the same segment.

All ALTER code associated with segmentation works the same on both Small Systems and the A Series. Unless you are using ALTER, we recommend that you eventually remove all section numbers.

Physical segmentation is documented in the "Segmentation" section of the "COBOL74 Reference Manual."

ACCEPT <DATA-NAME> FROM TIMER

The TIMER clause on Small Systems returns a 10-digit number representing the number of tenths of a second since midnight. On the A Series, TIMER returns an 11-digit number representing the number of 2.4 micro-seconds since midnight. B7T generates source code to convert the time from the A Series format to perform like the Small Systems format.

ACCEPT

The A Series accepts only one ACCEPT message. If several ACCEPT messages are entered without the program doing an ACCEPT, all but the last one entered are discarded.

CALL SYSTEM ZIPSB

The Small Systems control syntax that is zipped is different from the A Series statements that are passed to WFL. The Small Systems use the syntax CALL SYSTEM ZIPSB, while the A Series uses the syntax CALL SYSTEM WFL. When progressing to the A Series, you must either convert all the CALL SYSTEM ZIPSB statements to CALL SYSTEM WFL or remove those statements. You can change the syntax manually or the translator will automatically change the Small Systems CALL SYSTEM ZIPSB to CALL SYSTEM WFL. However, the translator will not change any Small Systems control statements that are zipped via the ZIPSB statement.

IPC CALL

The A Series handles IPC CALL statements in a slightly different manner than Small Systems. The following subsections discuss the differences of Redefines, Type Checking, and Length Checking.

Redefines

The A Series does not allow an 01 level item which redefines another 01 level item to be passed as a parameter in an IPC CALL statement. For example, the following Small Systems code produces a syntax error on the A Series:

```
01  A               PIC 99.
01  B REDEFINES A   PIC XX.

CALL "PROG-X" USING B.
```

The syntax can be corrected by passing "A" or by reversing the order of the declaration and having "A" redefine "B".

Type Checking

Small Systems does not check the item type of each parameter. The A Series requires that the item type must match. Just a reminder, group items, PIC X, and PIC 9 items are always display and therefore match each other. The A Series Type check occurs at the time of the IPC CALL.

The following examples illustrate how the A Series handles type checking.

Example 1

```
        CALLER                                CALLED
        ------                                ------

   01  A      COMP.                      01  B        PIC 9999    COMP.
      02  A1     PIC 99.
      02  A2     PIC 99.
```

This program runs on Small Systems but produces a run-time error on the A Series. "A", even though declared as COMP for its subordinates, is a display item and does not match "B".

Example 2

```
        CALLER                                CALLED
        ------                                ------

   77  A         PIC S99.                01  B        PIC XX.
```

Since both items types are display, this is not an error.

Length Checking

Length checking works differently on the A Series than on Small Systems. However, this difference only affects programs being written for the A Series. Any program that runs on the Small Systems will run on the A Series.

The Small Systems checks the length at the time of the IPC CALL. The A Series checks the length at the time the parameter is used in the called program. On the A Series, the length of the item is always the length described in the Linkage Section of the called programs. The end of the data is always on a word (six bytes) boundary. If a parameter is passed that is longer than the receiving area, the characters that exceed the length of the receiving area are truncated and cannot be referenced. If a parameter is passed that is shorter than the receiving area, the unused area cannot be referenced. Any reference to this unused area causes the program to be DSed.

The following examples show the actions taken by the A Series under various conditions.

Example 1

```
        CALLER                            CALLED
        ------                            ------

    01  A.                            01  B.
        02  A1   PIC X.                   02  B1   PIC X.
        02  A2   PIC X(6).                02  B2   PIC X(6).
        02  A3   PIC X.                   02  B3   PIC X.
                                          02  B4   PIC X(4).
                                          02  B5   PIC X.
```

In this example, assume "A" is passed to "B". A reference to "B1", "B2", and "B3" refers to "A1", "A2", and "A3", respectively. A reference to "B4" refers to the four invisible bytes that make A a full word (integral number of six bytes). These invisible bytes are always initialized to hex 00. A reference to "B5" causes the program to be DSed with a SEG ARRAY ERROR or an INVALID INDEX. Even though "A" takes up 12 bytes (two words) its length is only eight bytes. "MOVE "123456789012" TO A" produces a truncation warning and only the leftmost eight bytes are moved.

Example 2

```
        CALLER                          CALLED
        ------                          ------

    01  A   PIC X(10).            01  B   PIC X(5).
```

This does not cause an error. The length of "B" is 5, therefore, it is not possible for "B" to reference the last five bytes of "A".

Example 3

```
        CALLER                            CALLED
        ------                            ------

    01  A   PIC 9  COMP VALUE 1       01  B   PIC 9999 COMP.
```

The value of "B" is 1000. Remember, "A" is followed by 11 invisible digits that are initialized to hex 0.

Example 4

```
        CALLER                              CALLED
        ------                              ------

    01  A   PIC 9   COMP   VALUE 1.    01  B   PIC 9(12)   COMP.
    01  C   PIC 9   COMP   VALUE 1.    01  D   PIC 9(13)   COMP.
```

"B" has a value of 100,000,000,000. Any reference to "D" causes a SEG ARRAY ERROR or an INVALID INDEX. Remember that you can fit 12 digits into one word. "B" exactly fits into the word occupied by "A". "D" takes two words and therefore any reference to it goes past the end of "C" and an error occurs.

DISPLAY

On the A Series, the DISPLAY statement stops displaying a string as soon as it encounters a hex 00 (NULL). On the Small Systems, the NULL is not displayed, but it does not stop the display. The following example shows one occurrence of this and a possible solution.

```
01  A.
  02  B       PIC X(5)    VALUE "12345".
  02  FILLER  PIC X.
  02  C       PIC X(5)    VALUE "ABCDE".

DISPLAY A.
```

On Small Systems, the DISPLAY command would show "12345ABCDE". On the A Series, the DISPLAY command would show "12345". To solve this problem, add VALUE SPACES to the FILLER item. Then the A Series will display "123456ABCDE".

IF STATEMENT

On Small Systems, the following IF statement would take the true branch if the data base item had never been initialized:

```
IF DATA-BASE-ITEM = ALL F
```

To ensure that the true path is taken on the A Series, change the code to:

```
IF DATA-BASE-ITEM IS NULL
```

This syntax is valid only with data base items.

INDEXING

The A Series does not do complete bounds checking on indexing. If an index is out of range, the program is DSed only if the result of the index is completely outside of the 01 level structure.

For example:

```
01   EXAMPLE-TABLE.
   02   A     PIC X.
   02   B     PIC X    OCCURS 5   INDEXED BY BX.
   02   C     PIC X.

SET BX TO 0.
```

DISPLAY B(BX).	Displays A
SET BX TO 6. DISPLAY B(BX).	Displays C.
SET BX TO 7. DISPLAY B(BX).	Since every 01 is an integral integral number of words (six bytes to a word), this this displays "invisible" data between C and the end of the 01. This data is initialized to hex 00.
SET BX TO 12.	DSed because of invalid index.

MOVE

A move in which the areas defined by the sending and receiving fields overlap is different on both systems. Overlapping moves are not recommended on either system.

A move in which either the sending or receiving field is computational and the other field is a group is different on both systems. If the receiving field is computational, the Small Systems removes zone bits. If the sending field is computational, the Small Systems adds zone bits. In both situations, the A Series moves the contents unchanged and is treated as a group move.

OPEN

The LOCK ACCESS option of the OPEN statement is not implemented on A Series.

PERFORM

There is a minor difference in the action taken by a PERFORM statement with multiple varying clauses (format 4 PERFORM). For information, refer to the "Procedure Division" section and the "PERFORM" subsection of the "A Series COBOL74 Reference Manual". The flowchart in Figure 8-2 of the reference manual provides the best explanation of how the format 4 PERFORM statement works on the A Series.

PERFORM OR SEARCH EARLY EXIT CODE

The A Series does not generate early exit code for SEARCH statements. The following code, while valid in Small Systems, causes the program to be DSed with an invalid subscript if it is used on the A Series:

```
01  EBCDIC-STUFF.
  02  S-TABLE    PIC XX   OCCURS 12 TIMES
                          INDEXED BY I-1.
SET I-1 TO 1.
SEARCH S-TABLE VARYING I-1
     WHEN I-1 = 13 OR S-TABLE(I-1) = SPACES
          NEXT SENTENCE.
```

When "I-1" is 13, the Small Systems would stop the evaluation at the "I-1=13". The A Series would continue and try to evaluate "S-TABLE(13)", which would cause the program to be DSed. The same situation occurs with only those PERFORM statements in programs where $OPTIMIZE is set.

Changing the code to achieve early exit from the PERFORM and SEARCH statements is program dependent.

SORT

On the A Series, you may put a disk size clause in the SORT statement instead of putting it in the object-computer paragraph; the syntax and semantics are the same. This clause immediately follows the memory size clause. If the disk size is stated in the object-computer paragraph and in the SORT statement, the disk size clause in the SORT statement overrides the disk size clause in the object-computer paragraph.

WAIT

The following is a list of the differences between Small Systems and the A Series WAIT:

1. The A Series does not accept the USING clause.

2. The A Series has implemented READ-OK, but it is valid only for REMOTE files. The A Series has not implemented WRITE-OK.

3. The A Series has implemented INPUTEVENT, OUTPUTEVENT, and CHANGEEVENT. INPUTEVENT is valid for REMOTE files and PORT files. OUTPUTEVENT and CHANGEEVENT are valid only for PORT files. (The compiler accepts the syntax for other kinds of files, but the events are never set to TRUE.) Refer to the "Queue/Port Files" section later in this manual for details about INPUTEVENT, OUTPUTEVENT, and CHANGEEVENT.

4. READ-OK ON <queue-file> is similar to ATTRIBUTE INPUTEVENT OF <port-file>. READ-OK <remote-file> is valid on the A Series.

5. WRITE-OK ON <queue-file> is similar to ATTRIBUTE OUTPUTEVENT OF <port-file>.

6. The primary use of the CHANGEEVENT attribute is to notify one program that another program has closed a port file. See the "Queue/Port Files" section later in this manual, and the "A Series I/O Subsystem Reference Manual" for more information regarding this attribute.

7. The A Series has implemented ODT-INPUT-PRESENT. The following example illustrates the use of ODT-INPUT-PRESENT.

```
       WAIT UNTIL
                 600,
                 ODT-INPUT-PRESENT,
                 READ-OK ON QUEUE-FILE
                 GIVING G.

       WAIT AND RESET
                 600,
                 ODT-INPUT-PRESENT,
                 ATTRIBUTE INPUTEVENT OF PORT-FILE,
                 ATTRIBUTE CHANGEEVENT OF PORT-FILE,
                 GIVING G.
```

   If AND RESET is not present, the ODT-INPUT-PRESENT attribute stays on until an ACCEPT statement is executed and all subsequent WAIT statements that wait on ODT-INPUT-PRESENT immediately become true.

COMPILER-DIRECTING DOLLAR SPECIFICATIONS

On Large Systems, dollar specifications are call Compiler Control Images (CCI). There are two types of CCIs: temporary and permanent. A temporary CCI has only one $ in column seven and can be made permanent by adding another $ in column eight. Dollar specs are listed by default on the Small Systems. Permanent CCIs are listed by default on Large

Systems. The temporary CCIs on Large Systems will not appear on the listing unless the default is overridden by a $ SET LISTDOLLAR.

On the Small Systems, lines of code from a copy library are listed by default. On the Large Systems, lines of code from a copy library are not listed by default. This may be overridden with $ SET LISTINCL.

On the Small Systems, the compiler attempts to continue no matter how many errors it encounters. The Large Systems has a Compiler Control Image called ERRORLIMIT that tells the compiler to quit after encountering a specified number of errors. If the compile is done through WFL, the default for ERRORLIMIT is 10. If the compile is done through WFL, the default for ERRORLIMIT is 150.

The only compiler-directing dollar specification which requires a manual change is the following:

Small Systems	A Series
XSEQ	SREF

ADDITIONAL FEATURES

The following is a list of significant features available on the A Series that are not available on Small Systems.

1. The A Series ISAM (KEYEDIO) provides all the features offered in the Small Systems TAG style and COBOL74-style ISAM files, plus these additional features:

    a. Full recovery up to the last record written.

    b. Serial, sequential, random, and keyed access with update capabilities.

    c. Declaration in each program of only those keys used by that program.

2. The A Series COBOL74 compiler compiles programs much faster than the Small Systems compiler.

3. The A Series offers a separate compilation (SEPCOMP) ability that allows only the parts of a program that have been changed to be compiled.

COBOL74

4. The A Series COBOL74 compiler includes the ANSI REPORT WRITER feature.

5. The A Series allows CALLED IPC programs to be shared.

6. The A Series allows program modules to be bound to other programs, including programs written in different languages.

9 RPG

Most of the Small Systems RPG constructs can be translated to the A Series using the Burroughs RPG Translator. Those constructs that can not be handled by the translator are discussed in this section along with methods of converting them. To help you better understand A Series RPG, some of the changes made by the translator are also discussed here.

THE RPG TRANSLATOR

To help transfer Small Systems RPG to A Series RPG we recommend you use the Burroughs B 1000 Series RPG to Burroughs A Series RPG Translator (BRT). The translator accepts Small Systems RPG source code and produces A Series RPG source code. The translator is available in two versions: one that runs on Small Systems and one that runs on the A Series. The translator translates most constructs; those constructs that cannot be converted by the translator are clearly flagged for manual change.

The Burroughs RPG Translator is located on the A Series 3.6 Release BTA360 Migration Aids tape.

DOLLAR SPECIFICATIONS

There are two types of dollar specifications: compiler-directing and file attribute. On the A Series, compiler-directing dollar specifications are called Compiler Control Images (CCI).

Compiler Control Images (CCI)

There are two types of CCIs, temporary and permanent. A temporary CCI is specified with a dollar sign ($) in column six. A permanent CCI has a dollar sign in columns six and seven. Permanent CCIs are listed along with the program. Temporary CCIs are not listed unless $ SET LISTDOLLAR is specified.

LISTINCL

On Small Systems, lines of code from a LIBR file are listed by default. On the A Series, lines of code from an INCLUDE file are not listed by default. To list the included file, specify $ SET LISTINCL.

ERRORLIMIT

On Small Systems, the compiler attempts to continue processing regardless of the number of errors it has encountered. The A Series has a CCI called ERRORLIMIT that can be set to tell the compiler to terminate processing after the error limit has been reached. If the compile is done through CANDE, the default ERRORLIMIT is 5. If the compile is done through WFL, the default error limit is 150.

Small Systems VOID

The only compiler-directing dollar specification that requires manual change is the Small Systems VOID. The VOID statement must be changed to the A Series DELETE statement. For example,

Small Systems	A Series
00100$ VOID 500	00100$ SET DELETE
	00500$ POP DELETE

File Attribute Dollar Specification

Most of the Small Systems file attribute dollar specifications are handled by the RPG Translator. Those not handled by the translator are listed below along with information about how to change them. Additional information is available in the RPG Syntax Guide.

Small Systems	A Series
TAG	This specification is not applicable on the A Series. All A Series ISAM files are similar to Small Systems COBOL74 ISAM files; refer to the "ISAM Files" section of this manual for additional information.
DNAME	The A Series equivalent is the UNITNO attribute. The value field of $DNAME contains an alpha-mnemonic for the physical hardware device to which it refers. DNAME must be changed to UNITNO and the

	alpha-mnemonic must be changed to a numeric value.
DRIVE	No direct equivalent exists on the A Series. However, the A Series FAMILYINDEX file attribute is similar to the Small Systems DRIVE and can be used to replace it.

FILE NAMING

There is no direct equivalent to the Small Systems FAMILY attribute. The A Series FAMILYNAME file attribute is the same as the Small Systems PACKID file attribute; it is not related to the Small Systems FAMILY.

The A Series title elements may contain up to 17 characters per element. There may be a usercode, up to 12 levels of file name, and a family name.

If there is a usercode in the Small Systems TITLE, use the A Series syntax (USERCODE)A/B ON C. There is no slash after the USERCODE.

We recommend that you limit all file names to uppercase letters and numbers and that you start each part of the title with a letter. Many parts of the A Series will not support special characters. If you must use special characters, the hyphen (-) and the underscore (_) cause the fewest problems. The period (.) is prohibited as a special character.

ISAM

All B-Indexed files, Tag files, and ISAM files should be progressed to the A Series KeyedIO files. Refer to the "ISAM Files" section of this manual for additional information. The Small Systems B-Indexed and Tag files ignore the sign on numeric keys (the absolute value is compared). This is not true for A Series KeyedIO.

GENERAL LANGUAGE ELEMENTS

The Small Systems support up to 31-digit numbers. The A Series supports up to 23-digit numbers. Check your Small Systems RPG programs for numbers exceeding 23 digits. Any attempt to move data from an alphanumeric field larger than 23 characters to a numeric field is not allowed.

H-Specification

The only differences between the Small Systems and A Series H SPEC are in column 16 and column 53.

Column 16

The A Series does not support B-Indexed files.

Column 53

On Small Systems, the default for serial input files is to explicitly close the file upon encountering the end-of-file (EOF). On the A Series, normally, all files are closed at end-of-job (EOJ) as specified in the RPG program. To close serial input files on the A Series at the end-of-file (EOF), you must put an S in column 53. For transferring your Small Systems RPG programs to the A Series, the RPG Translator automatically makes this change for you.

X Edit Code

On the Small Systems, when an unpacked, unedited, positive number is processed, the sign is not stripped unless there is an X edit code. For instance, a positive 1234 would print as A234 unless there was an X edit code. On the A Series, the default is to strip the sign. If you want the same result on the A Series that you had on your Small Systems, you must put the letter O in column 40.

Regardless of the value in column 40, the X edit code still works the same way on both systems. Use caution with X edit codes and indexed files; if your programs expect the sign in the key field for positive values of unpacked numeric keys, make sure the programs that add or update the record in the indexed files have an O in column 40. If the sign was stripped (the O was not specified), programs expecting a value such as A234 might receive a NOT FOUND error when accessing the file because the actual value is 1234.

D-Specification

Small Systems allow a file attribute dollar specification (only $PACKID or $DISKID) before the D-specification when library files reside on a disk other than the systems disk. The A Series does not allow any file attribute dollar specifications before the D-specification. The RPG translator deletes the file attribute dollar specification when it appears before a D-specification.

A-Specification After D-Specification

On Small Systems, an A-specification may not follow a D-specification.

On the A Series, an A-specification is allowed but it must immediately follow the D-specification.

The format of data base attribute specification lines is the same as File Attribute Images described in Section 6 of the "A Series RPG Reference Manual." Use of the data base attributes is subject to the same rules as use of file attributes.

Currently TITLE is the only attribute which is supported. TITLE may not be changed when the data base is open.

F-Specification

The differences between Small Systems F specs and the A Series F specs are column 40-46 and column 53.

Column 40-46

Small Systems accepts many device names that the A Series does not. The devices that are accepted on the A Series are listed in Section 5 of the "A Series RPG Reference Manual."

Column 53

By default, Small Systems allow all files to be shared. On the A Series, by default, all output, add, and update files are opened with "lock." This means that no other program can open these files, not even for input. In order for the A Series to allow shared files, you must put a U in column 53.

When transferring your Small Systems RPG programs to the A Series, the RPG Translator automatically places a U in column 53.

E-Specification

The E spec for A Series differs from Small Systems in Tables, columns 9-10, and compile-time vectors.

Tables

The A Series has several restrictions for table handling. They are:

1. You cannot display (DSPLY) an unsubscripted table.
2. You cannot use an unsubscripted table in either the FACTOR 2 or RESULT field of a MOVEA operation.
3. You cannot use an unsubscripted table on input specifications.

Columns 9-10

Automatic chaining is not supported on the A Series.

Compile-Time Vectors

On Small Systems, the CARDS and SOURCE files accept RPG source statements, and the TABCRD file accepts compile-time vectors. On the A Series, the compile-time vector data is included at the end of the source statements. The Small Systems allow compile-time vectors to be up to 96 characters long. The A Series limits compile-time vectors to 80 characters.

RPG

Because neither CANDE nor the EDITOR allows editing of mixed source and data files, editing RPG source files with vector data at the end can be cumbersome. We recommend that you keep two separate files: one for source and one for the vectors (type DATA). Then make your last card in the source file $ INCLUDE "<vector file title>". (The quotes are required and the vector title's format is (USERCODE)A/B ON C).

The data for all the vectors should be included in the same file, but each vector must start with a VVECTOR card. Because each vector starts with a VVECTOR card, it is possible to have other than the last vector as a short vector. For example,

```
00100VVECTOR
DATA
   <data>
     VVECTOR             Sequence numbers on VVECTOR cards
                              are optional.
DATA
   <data>
     VVECTOR
```

For more information, see the "Extension Specification and Vectors" section in the "A Series RPG Reference Manual."

When transferring Small Systems RPG programs to the A Series, the RPG Translator automatically converts your compile-time vectors.

T-Specification

The translator makes all corrections to the T spec.

C-Specification

The Small Systems and A Series C specs differ in the areas of MOVEA, DSPLY, SEND, SETLL, RECV, ZIP, OPEN, and CLOS.

MOVEA

The A Series does not allow an unsubscripted table entry.

DSPLY

The A Series does not allow an unsubscripted table entry.

SEND

The BRT translator converts SEND to EXCEPT. FACTOR 2 on the A Series can be a file name for an EXCPT.

SETLL

On Small Systems, a SETLL causes the next record to have a key greater than the key specified on the SETLL statement. On the A Series, the next record will have a key greater than or equal to the specified key.

If you have been subtracting one (1) from the key before the SETLL, you will need to remove the subtraction in the A Series program. If you want the next greater key, add one (1) before the SETLL.

RECV

The translator converts RECV to READ.

ZIP

Zipped Small Systems text differs from zipped A Series text. On the A Series, you may only ZIP a valid WFL job or command. For example, you may ZIP "START DAILY/INVENTORY/JOB" or "RUN PROGRAM/A; FILE FILEIN(TITLE = A/B ON C);".

The Small Systems allow FACTOR 2 to contain an unsubscripted array. It then zips each element of the array as a separate statement. The A Series does not support unsubscripted array syntax.

OPEN

OPEN is not supported on the A Series.

CLOS

CLOS is not supported on the A Series.

O-Specification

The differences between the Small Systems and A Series O specs are in column 15, columns 23-31, column 38 (edit code), and column 39.

Column 15

On the Small Systems, this field specification causes the record to be read and the file to be updated before the data is made available to the program via input specs. The A Series does not allow total time output for primary and secondary update files.

Columns 23-31

On Small Systems, this field specification causes the first record to be updated before the data is made available to the program via input specs. The A Series does not allow 1P output for update and combined files.

Column 38 (Edit Code)

See H-Specification, earlier in this section.

Column 39

We recommend that you do not use this field specification. However, the A Series will accept this field specification but will issue the warning "Blank after not allowed with constants (COLUMN 39)."

RUN-TIME ERRORS

There are differences between Small Systems and A Series handling of run-time errors. The following is a list of the A Series run-time errors. For more information, see Appendix C of the A Series RPG Reference Manual.

```
DCERR <further error information>
DELERR
DIVZ <location of zero division>
DOPKY <file identifier>:<record contents>
IDENT <file identifier>:<record contents>
IOERR <file identifier>:
KEYMOD <current key value>:<update key value>
KEYNF <file identifier>:<key value>
KEYSEQ <file identifier>:<key value>
MSEQ <file identifier>:<match field values>
RECKY <file identifier>:<search key>:<record key>
REOF <file identifer>
SEQ <file identifier>:<record contents>
SQRT
VLOAD = <vector file identifier>
VSEQ <vector identifier>:<record contents>
VSUBS <location of invalid subscript>
```

ADDITIONAL FEATURES

The following is a list of the significant features available on the A Series RPG that are not available on the Small Systems RPG.

1. The A Series ISAM (KEYEDIO) offers all the features offered in the Small Systems B style, TAG style, and COBOL74 style ISAM files, plus these additional features:

    a. Full recovery up to the last record written.

    b. Serial, sequential, random, and keyed access with update capabilities.

    c. Full support of all data types as keys.

    d. Alternate keys.

   e. CHAIN, followed by READ, allows random positioning followed by sequential-by-key accessing.

   f. The DELETE operation.

   g. The ability to change to an alternate key.

2. The ability to use the ASCII collating sequence for comparison operations.

3. The ability to access and create variable length records.

4. Additional data types of unsigned packed decimal (useful for accessing data created by COBOL programs), and separate leading and trailing signed display data (-123 and 123-).

5. The addition of an optional file name in FACTOR 2 of the EXCPT operation code to allow additional control over output operations.

ADDITIONAL A SERIES DOLLAR SPECIFICATIONS

The following is a list of the A Series dollar specifications that are not available on Small Systems.

Dollar Specification	Purpose
LINEINFO	$ SET LINEINFO causes the compiler to compile code into the program so that if it is DSed, the MCP prints the sequence number of the line of source code that was being executed at the time of the DS. It also prints any sequence numbers on the stack from subroutine calls; it will tell you what line of code was being executed and, if that line was in a subroutine, what line of code called that subroutine. LINEINFO is set by default for programs compiled by CANDE and reset by default for programs compiled by WFL.
PAGESIZE	$ PAGESIZE=<number> causes the compiler to skip to the top of the page after printing <number> of lines. The default number of lines is 58.

TITLE	$ TITLE=<"string"> causes the compiler to print <string> as the heading at the top of each page of the compiled listing.

10 DMSII OPERATIONS

This section describes the operational differences between the Small Systems DMSII and the A Series DMSII. It is assumed that the A Series user is running CANDE and any syntax examples are geared for CANDE. For example, the use of the Dollar Card ($) in the "COMPILE AS $MYDB" statement tells CANDE to not put "OBJECT/" in front of "MYDB". It is also assumed that the user is going to create a data base named MYDB and that there is an existing DASDL input file named MYDB/DASDL/INPUT.

COMPILING A DATA BASE

Similar to Small Systems, DASDL input can be created using CANDE or ADDS on the A Series. To compile the data base on the A Series, the following syntax can be used:

```
GET MYDB/DASDL/INPUT
COMPILE AS $MYDB
```

During the compile, a description file is created. This file contains information similar to the dictionary file created on the Small Systems. If the DASDL input does not explicitly reset DMCONTROL, the DASDL compiler will automatically run SYSTEM/DMCONTROL to create the control file that is needed at run time on the A Series.

INITIALIZE

On the A Series, if the DASDL input specified "INITIALIZE;", DASDL will automatically run SYSTEM/DMUTILITY so that your DMSII data base files will be initialized. If the DASDL input did not contain the INITIALIZE clause, the following syntax can be used (in CANDE):

```
RUN $SYSTEM/DMUTILITY("DB = MYDB INITIALIZE =")
```

Reset ZIP

On the A Series, if the DASDL input did not explicitly reset ZIP, the DASDL compiler will initiate the compilation of any required software. This will always include a DMSUPPORT library. This code file contains the tailored code that is analogous to the code that the Small Systems DASDL put into the dictionary file. This code efficiently handles data base specific code such as VERIFY, SELECT, and WHERE clauses, and REMAPS as specified in the DASDL input. If MYDB is an audited data base, some tailored programs for handling the various types of recovery are automatically compiled.

If the ZIP option was reset, this compilation will have to be started manually. The following syntax may be used for an unaudited data base:

```
     START DATABASE/WFL/COMPILEACR("DB = MYDB COMPILE = DMSUPPORT");
```

If your data base is audited use the following syntax:

```
     START DATABASE/WFL/COMPILEACR("DB = MYDB AUDIT = SET");
```

This causes the DMSUPPORT library and all necessary recovery programs to be compiled. For more information refer to the documentation in the DATABASE/WFL/COMPILEACR file.

Once the DMSUPPORT library (and any necessary recovery programs) is compiled, the data base is ready for use. Actually, an application program could be compiled as soon as the description file was created by the DASDL compiler.

COMPILING A DMSII APPLICATION PROGRAM

When compiling a DMSII-capable program on the A Series, the appropriate compiler must be chosen. A Series is a little different here. For example, the Small Systems COBOL compiler is capable of compiling DMSII syntax. The A Series COBOL compiler is not. Instead, the BDMSCOBOL compiler must be used. The following table shows the correct compiler to use:

DMSII Operations

Application Code	Compiler
COBOL (68)	BDMSCOBOL
COBOL (74)	BDMSCOBOL74
RPG	BDMSRPG
PL/I	BDMS/PL/I
ALGOL	BDMSALGOL

One way to compile a DMSII-capable COBOL74 application program on A Series would be to get the file in CANDE and initiate a compile (no file equation is necessary in CANDE) as follows:

```
GET MY/APPLICATION
COMPILE WITH BDMSCOBOL74
```

However, for the compilation to be successful, the description file must be available. The compiler will use a co-routine named DATABASE/INTERFACE. This routine reads the description file and tells the compiler about the record layouts of any data sets that are invoked. Unlike the Small Systems, there is no option to create "copy library" record layout files that can be modified (e.g., MYDB/#DSA).

OPENING DATA BASE STACKS

When a program opens a data base on A Series systems, two stacks are brought into the mix. One is the Data Base Stack (or DBS). The accessroutines control this stack, but the name of the stack will be the name of the data base (e.g., MYDB). The other stack will be the DMSUPPORT library (generally named DMSUPPORT/MYDB for a data base named MYDB). This library contains all of the tailored code for a particular data base. Note that all databases can share the same SYSTEM/ACCESSROUTINES code file, but each data base will have its own data base stack.

Once a data base is up and running, there are several global data base parameters that can be queried and dynamically changed through a feature called Visible DBS Commands. Refer to the "A Series Data Management System II (DMSII) Utilities and Operations Guide" for more information. There is no equivalent to the Small Systems DB command which allows a user to get some current run-time statistics about an application.

Use the DBS command (at the ODT) to get the mix number of the data base stack. Then, enter commands such as the following:

<mix number> SM STATUS	This command causes the data base stack to display information about the amount of memory being used, the number of users of the data base, and the state of the audit files.
<mix number> SM ALLOWEDCORE equals 30000	This change lets you dynamically change the parameters, such as allowed core, and the frequency of syncpoints.
<mix number> SM STATISTICS equals ON	This command enables the collection of data base statistics. Note that these statistics are not gathered "by program"; they are gathered by data base. The statistics include buffer usage, counts of physical reads and writes, wait times for physical reads and writes, and counts of logical FINDs, STOREs, and DELETEs by structure.

There are other Visible DBS Commands. Refer to the "A Series DMSII Utilities And Operations Guide" for further details.

MEMORY

The A Series DMSII treats memory differently than the Small Systems DMSII. Small Systems will potentially fill all of the available memory on a system with data buffers. A Series limits a data base to a particular amount of memory. The ALLOWEDCORE parameter can be set in DASDL, and dynamically changed.

ASN Memory

If you have a machine with ASN memory (e.g., B7900, A3, A9, A10, A15), you should consider running most of your data bases in a local subsystem. There are tradeoffs here that need to be examined on an individual basis. Shared memory is a tight resource on ASN memory systems. If you bring up five data bases into shared memory, the system will most likely run poorly. If you put the five data bases into the local subsystems, the system would run more efficiently. A subsystem specification in the DASDL input is used to direct a data base to a local subsystem.

Shared/GLOBAL Memory

If you decide to put a data base in shared (or GLOBAL) memory, investigate the use of the LOCALBUFFERING DASDL option. This option allows the data base buffers to be allocated in the local subsystem instead of shared memory.

UPDATE and REORG

The way that a data base update or reorganization is done on A Series is similar to that on the Small Systems. On the Small Systems, an update has an immediate effect. DASDL updates the dictionary, and the update is done. On the A Series, DASDL updates the description file. The update does not take effect until DMCONTROL has been run to move the description information into the control file where the run-time data exists.

On the Small Systems, if a reorganization is done, the DASDL compiler creates a control file that tells the reorganization program what changes to make to the data base. On the A Series, after you do the DASDL run to specify the reorganization, you must run the BUILDREORG program. For most cases, the only input you need to supply the BUILDREORG program is "UPDATE;". This causes the BUILDREORG program to look at the description file and generate a reorganizaion program to make the required changes to the data base.

If you want to keep your current version of the data base up and running while you get ready for a reorganization, reset ZIP in the DASDL input. This will suppress the generation of a new DMSUPPORT library (and recovery programs). However, you can run BUILDREORG and generate the reorganization program. Because you also now have the new description file, you can begin the recompilation of any programs that need recompilation. Existing programs can still run because the control file has not been updated, and the existing DMSUPPORT library is still present. When you are ready to do the reorganization, bring down the data base, run the reorganization program, and compile the new DMSUPPORT library and recovery programs. The data base is ready to run (with the recompiled applications).

The Small Systems DASDL compiler accepts simple reorganization statements such as GENERATE for garbage collection. On A Series machines, that input should be directed to the BUILDREORG program (you do not have to run DASDL). For more information on BUILDREORG refer to the "A Series DMSII Utilities and Operations Guide."

BACKING UP A DMSII DATA BASE

Backing up a DMSII data base is different on the A Series. There is a general utility (SYSTEM/DMUTILITY) that is used to make backup tapes of a data base and to restore them as necessary. You should not use a library maintenance COPY command (equivalent to Small Systems SYSTEM/COPY) for data base backup. The SYSTEM/DMUTILITY program computes additional checksums to guarantee that the data is correctly written and read from the tape. Also, the online dump feature and rebuilding parts of a data set require SYSTEM/DMUTILITY dumps of the data base.

RECOVERY

Data base recovery on the A-Series is very similar to recovery on Small Systems. The following paragraphs describe recovery on the the A-series.

Abort Recovery

Abort Recovery occurs when a program closes the data base while in transaction state. When this occurs, all users of the data base are stopped while the abort recovery backs out changes to a sync point. The only significant difference on the A Series is that all users of the data base (including inquiry users) have their current record pointers "fixed up."

CLEAR/START Recovery

Both systems have CLEAR/START (HALT/LOAD on A Series) recovery. Prior to Mark 11.0, you had to key in RC <data base name> to cause this type of recovery to occur on Small Systems. Jobs hung "waiting for recovery" until the RC was explicitly done. On Mark 11.0, this type of recovery automatically occurs on the first OPEN of the data base after the interruption.

To manually do a HALT/LOAD recovery on the A Series, before the first open, the A Series recovery program can be initiated by a RUN <data base name>/RECOVERY statement. For details, see the "A Series DMSII Utilities and Operations Guide."

DMSII Operations

Full Data Base Recovery

A full data base recovery from a previous dump of the data base is functionally the same as on Small Systems. However, on the A Series, the backup dump of the data base is created by a SYSTEM/DMUTILITY DUMP command (not an "ODT" COPY command). The recovery is initiated by running SYSTEM/DMUTILITY with input such as DB = MYDB ON DBPACK RECOVER (REBUILD THRU AUDIT <number>) FROM <tape name>. This causes the previously created dump of the data base (on <tape name>) to be loaded out to disk. Then, the recovery program will read the necessary audit files to roll the data base forward through time. The stopping point can be a date and time or the BOJ/EOJ of a program as well as an audit file number. Note that the RECOVERY request to DMUTILITY takes care of the whole task including loading out the backed up copies of the data base files.

Single Structure Recovery

In order to do recovery of a single structure on Small Systems, you had to load the backup copy of that structure out to disk, put the old dictionary file that goes with the old structure out on disk, and then do an RC MYDB ON DBPACK DBPACK/MYDB/MYDATASET. On the A Series, you run SYSTEM/DMUTILITY with the following input: RECOVER (ROWS USING BACKUP) <file name> FROM <tape>. This causes DMUTILITY to load the old version of the structure out to disk. Then, the recovery program will read the necessary audit files to roll that structure forward through time. This will stop when the structure is up to date.

Row Recovery

If one row (or area) of a DMSII structure has been corrupted, it can be rebuilt from a previous copy of the structure and the audit files. This can occur while the data base is up and running.

Rollback Recovery

The recovery programs can roll a data base forward or backward in time. The stopping point can be BOJ or EOJ of a job, a date and time, or an audit file number.

Online Dump Recovery

A data base may be safely dumped while updates are going on. This is called an "online dump". The recovery programs can then take this dump and the audit files created during the dump and produce a clean, ready to use data base.

For further information concerning DMSII Recovery, refer to the DMUTILITY section of the DMSII Utilities and Operations Guide.

DBCERTIFICATION

On the Small Systems, you run DBMAP to verify that a data base is not corrupted and to help you determine if a garbage collection is necessary. With A Series DMSII, you run DBCERTIFICATION to do the actual verification that the data base is not corrupted. This program checks, for example, that all records in a data set are contained in all spanning sets, and that each entry in a set points to the proper data set record. For more information on DBCERTIFICATION refer to the "A Series DMSII Data Base Certification Software Operation Guide."

DBANALYZER

DBANALYZER will examine each structure and provide information that can help you tune the data base or help you determine if a garbage collection is necessary. This information includes, for example, an analysis of the available space in a standard data set, and the number of levels of tables in an index sequential set (and how full the tables are). Refer to the A Series DMSII Utilities and Operations Guide for details on DBANALYZER.

11 DMSII DASDL

Since Small Systems DASDL is basically a subset of the A Series, most of the capabilities are the same. However, there are certain areas of the A Series DASDL that are implemented differently and they may affect your application programs. This section discusses the changes you need to make to Small Systems DASDL source deck to make it acceptable to the A Series DASDL. There are a few differences which do affect the application programs. These differences are also discussed in this section.

GENERAL INFORMATION

The changes described in this subsection are global and will have to be made wherever the constructs occur.

TITLE Statements

Because you cannot control the data base file names in the A Series, the A Series DASDL does not have the TITLE attribute. You must remove all TITLE statements from Small Systems source file before it will be accepted by the A Series.

INITIALVALUE

On the A Series, the INITIALVALUE cannot exceed the size of the item. For numeric items this applies to both the integer and fractional part of the item. For example,

```
     ITEM-1   NUMBER (3,2)     INITIALVALUE=0.100;
```

causes an error because the fractional part of the INITIALVALUE exceeds the size of the number. However, the following example is acceptable.

```
     ITEM-2   NUMBER (4,3)     INITIALVALUE=0.100;
```

For alphanumeric items, when the size of the INITIALVALUE is less than the size of the item, you must check the A Series DASDL rules to be sure the intended value is being used. We recommend that the INITIAVALUE match the size of the item.

Hexadecimal Literals

The A Series does not allow hexadecimal literals. Use the A Series LOW-VALUE and HIGH-VALUE to replace the Small Systems hexadecimal zeroes and Fs, respectively.

CONTROL File

The Small Systems have a data base DICTIONARY file to contain the description of the data base. It also contains information that is updated each time the data base is used. The A Series divides this information into two files, the DESCRIPTION file (static information) and the CONTROL file (run-time control information).

The Control File defaults to the systems disk unless a pack-id is set with the PACK option in the Control File declaration. For example,

```
CONTROL FILE
(
 PACK = USERPACK
);
```

For more information about the Control File, refer to Section 4 of the "A Series DMSII Data and Structure Definition Language (DASDL) Reference Manual."

AREASIZE

All occurrences of AREASIZE = <integer> in your Small Systems source file must be changed to the A Series syntax AREASIZE = <integer> BLOCKS.

DOLLAR CARDS

Dollar cards contain information to instruct the compiler to perform actions.

NO <dollar card option>

The Small Systems syntax of NO <dollar card option> must be replaced with the A Series syntax of RESET <dollar card option>.

<Option Name> And SET

In Small Systems, <option name> and SET <option name> are synonymous. In the A Series, specifying only <option name> resets all the options then sets the named option. This includes resetting options that are set by default. For example, $ MERGE resets the LIST option (which is set by default) and sets MERGE.

SET <option name> is the same on both Small Systems and the A Series.

Permanent Option Indicator

In Small Systems, permanent options are indicated with a dollar sign ($) in columns one and two. In the A Series, permanent options are indicated with a blank in column one and a dollar sign ($) in column two.

COMPILER OPTIONS

Listed below are Small Systems compiler options compared to the A Series compiler options. If a Small Systems compiler option does not appear on this list, it is the same on the A Series as it is on Small Systems.

NOTE

The A Series compilers directly read the description (dictionary) file. This eliminates the need for library files; therefore, none of the Small Systems library options can be used on the A Series.

B 1000 SERIES TO A SERIES PROGRESSION GUIDE

Small Systems	A Series
COBOL	WARNING. This is only a warning on the A Series, not a syntax error.
COBOLLIB	Not applicable, see NOTE.
CONVERT	Not applicable, see UPDATE.
DELETE	VOIDT.
DOUBLE	RESET SINGLE.
INCLUDE P/A/B	INCLUDE "A/B ON P" (requires quotes on the A/B ON P format).
INITIALIZE	This is not a dollar option on the A Series, it is a statement. You can initialize files if you include an INITIALIZE statement in your DASDL input. If initialized this way, the DASDL compiler runs SYSTEM/DMUTILITY to initialize your files.
LIST$	$. For example, replace the Small Systems "$LIST$" card with the A Series "SET" card.
LISTINCL	This is the default value for an A Series listing.
REORGANIZE	Reorganization is handled differently for the A Series than for Small Systems. Refer to the REORGANIZATION sections of the "A Series DMSII Utilities and Operations Guide" and the "A Series DMSII DASDL Reference Manual."
RPG, RPGII	RPG. There are different restrictions on DASDL items for the A Series than for Small Systems. Refer to Appendix A of the "A Series DASDL Reference Manual." This is only a warning on the A Series, not a syntax error.
RPGLIB	Not applicable, see NOTE.
SEQUENCE	SEQ. (Only the abbreviation SEQ is accepted on the A Series.)

	DMSII DASDL
SOURCE	Not applicable, see NOTE.
SOURCEONLY	Not applicable, see NOTE.
STANDARD	Remove the Small Systems $STANDARD card.
STRUCTURE	FILE.
SUPPRESS	There is no A Series equivalent. You cannot suppress warnings when using the A Series DASDL compiler.
TABLESIZE	There is no A Series equivalent. You cannot set a default maximum tablesize on the A Series.
TAPE	Because of the way in which REORGANIZATION is handled on the A Series, this is not accepted. See REORGANIZE.
UPDATE	This is not a dollar option on the A Series, it is a statement. The A Series UPDATE statement is described in Section 4 of the "A Series DMSII DASDL Reference Manual."
VERSIONCHECK	This Small Systems option is automatically done on the A Series DMSII software and cannot be overridden with a dollar option.
VOID	Use VOIDT, which is similar to the Small Systems DELETE option. The VOIDT option can accomplish the function of the Small Systems VOID but the syntax is different. Refer to Appendix A of the "A Series DMSII DASDL Reference Manual" for information. See also the OMIT dollar card.
WARNSUPR	This is the same as SUPPRESS.

OPTIONS

The A Series DASDL handles some physical options differently than the Small Systems DASDL. This subsection describes those options.

AUDIT SET and AUDIT RESET

The Small Systems AUDIT SET must be changed to the A Series syntax AUDIT.

If the Small Systems DASDL source contains AUDIT RESET, you must remove the RESTART DATA SET. The A Series MCP does not support the SM AUDIT SET/RESET command.

The A Series DASDL option RDSSTORE causes the A Series to treat the restart record exactly as it was treated on the Small Systems. To implement this A Series DASDL feature, add OPTIONS (RDSSTORE) to your Small Systems DASDL source and you will not have to change the recovery code in your programs. Refer to the "A Series DMSII DASDL Reference Manual" for more information about this option.

END-TRANSACTION

On Small Systems, application programs can get an ABORT exception at END-TRANSACTION only if SYNC is specified. On the A Series, the SYNC need not be specified to get an ABORT exception at END-TRANSACTION. However, the application program logic may require a change to handle this situation.

Physical Options

The A Series DASDL compiler handles some physical options differently than the Small Systems DASDL compiler. If you have specified AREASIZE, BLOCKSIZE, or TABLESIZE in the Small Systems DASDL source file, check what effect these options will have on the storage requirements for the data base. To check the storage requirements, put a $ SET FILE card in the converted A Series DASDL source, then compile it. The compile listing will show the storage requirements that DASDL has computed for each structure.

When you do not declare any physical options for a data set or set, the A Series DASDL compiler assigns values for these options based on the population and structure type of the data base. In most cases, the assigned values make efficient use of disk space, memory, and I/O.

If, after examining the listing produced by the FILE dollar options, the disk space needed for a structure seems large, try commenting-out the AREASIZE option. This causes the compiler to compute default values for the physical options (these values are shown in the resulting listing).

For more information regarding efficient values for physical options, see Appendix C of the "A Series DMSII DASDL Reference Manual."

Additional Options

These additional DASDL options, available only on the A Series, provide integrity checks for your data base. We recommend that you add these features to the A Series DASDL source after you have completed your progression.

KEYCOMPARE

KEYCOMPARE is an option that causes the accessroutines to verify that the value of the key item in the data set is equal to the value of the key item in the set or the automatic subset through which the data set is accessed. If an error is detected, the data base find or lock operation returns an exception.

ADDRESSCHECK

ADDRESSCHECK specifies that an addresscheck word be appended to all data blocks of every data base file. When a block is written, the segment address of the first segment of the block written is stored in the addresscheck word. When the block is read, the addresscheck value is verified. If an error is detected, the user program receives an exception.

When this option is specified, it applies to all data base files.

The ADDRESSCHECK option causes an extra word to be added to each block in every structure. You should check the effect the addition of this word will have on any physical structure options you have specified in the DASDL code.

The ADDRESSCHECK option is useful in detecting I/O errors, such as shifted errors.

CHECKSUM

A CHECKSUM is a value computed for each block by applying an equivalence operator to each word in the block. When the block is written, a CHECKSUM is calculated and stored in a CHECKSUM word appended to the end of each block. When the block is read, the CHECKSUM is recalculated and the result is compared to the stored value. If the CHECKSUM values are not equal, the user program receives an exception.

The CHECKSUM option causes an extra word to be added to each block in every structure. You should check the effect the addition of this word will have on any physical structure options you have specified in the DASDL code.

PARAMETERS

A change is required in the Small Systems MAXWAIT parameter when progressing from Small Systems to the A Series.

MAXWAIT is not a data base attribute on the A Series, it is a task attribute. Therefore, the Small Systems MAXWAIT specification is not allowed in the A Series DASDL. To use this option, you must set MAXWAIT in each of your programs. If you do not set the MAXWAIT attribute in the A Series program, the default limit is 0 (zero). This means that the program will wait forever. Refer to Appendix A in the "A Series WFL Reference Manual" for additional information regarding the MAXWAIT specification.

RESTART DATA SET

The A Series does not allow the restart data set to be remapped or to have variable format records.

The A Series RESTART data set always has the first seven bytes reserved for system use. If you have any programs that perform group moves, or the equivalent of a group move, to a RESTART data set name where the structure name is treated as the 01 destination, you will have to change those programs. When changing the programs, do not assume that the layout of the RESTART data set is exactly the layout specified in your DASDL source.

If there is a group move into the RESTART data set, we recommend that the destination of the move also be a group. For example, if you move a group from working storage into the RESTART data set, move it to a group in the RESTART data set, not to the RESTART data set name.

AUDIT TRAIL

The Small Systems BLOCKSIZE specification may need to be respecified for use in the A Series. If the Small Systems BLOCKSIZE specification reads BLOCKSIZE <integer>, it must be replaced with the A Series syntax BLOCKSIZE <integer> BYTES.

VARIABLE FORMAT RECORDS

Progressing from Small Systems to the A Series requires changes in the declarations for variable format records.

Small Systems allow the RECORD TYPE items in variable format records to be of type ALPHA or NUMBER, and the values of the RECORD TYPE may be anything that can be stored in the field described. Small Systems also allow the designation of FIXEDFORMATVALUE in DASDL.

The A Series does not allow a data type for the RECORD TYPE; it requires that the RECORD TYPE be a number from 1 to 254. This number represents the number of different variable formats. ALPHA record types are not allowed on the A Series, they must be converted to numbers. Zero is always the fixed format value. It is most efficient not to skip numbers. The syntax for variable format record specification is <data-name> RECORD TYPE (n), where n is the largest number allowed, not the size of the number field.

Small Systems allow 255 formats, the A Series allows only 254.

In Small Systems application programs, the record-type-item can be treated as any other data item in the DASDL. For example:

```
DASDL                               COBOL
-----                               -----

ADS DATA SET                        IF  RT = 2
   (                                    - STATEMENTS.
     A NUMBER (8);
     RT RECORD TYPE NUMBER (2);
     .
     .
     .
```

The preceding syntax is not valid on the A Series, instead, the code should be as follows:

```
DASDL                               COBOL
-----                               -----

ADS DATA SET                        IF ADS (RT) = 2
   (                                   - STATEMENTS.
    A NUMBER (8);
    RT RECORD TYPE;
    .
    .
    .
```

Small Systems implements variable format records as fixed length records with all records taking the size of the largest record. The A Series implements them as variable length records. This can decrease disk usage if many short records are used or increase disk usage if deleted records cannot be reused.

Small Systems allow the RECORD TYPE to be the only item in the fixed format part of the data set.

The A Series requires that at least one item in the fixed format portion of the data set must be a REQUIRED item. Any item that is a key of a set is automatically a REQUIRED item.

The A Series does not allow variable format ordered embedded data sets. The Small Systems DASDL must be changed to a standard embedded data set with a set. If a standard data set is embedded, it must have at least one set. This set may be index sequential or it may be an ordered list (ordered list is the default). This change means that what was one structure and file on the Small Systems becomes two structures and two files on the A Series, the data set and the data set's set.

ORDERED EMBEDDED DATA SETS

The differences between the way Small Systems and the way the A Series implements ordered data sets affect several of the physical attribute specifications. For further information, see the "A Series DMSII DASDL Reference Manual" about BLOCKSIZE and SUBBLOCKSIZE.

Small Systems BLOCKSIZE

To make the Small Systems DASDL acceptable to the A Series, the BLOCKSIZE integer must be replaced with a value which is the result of multiplying the value specified in the Small Systems DASDL for block size by the value specified in the Small Systems DASDL for table size. BLOCKSIZE is specified in RECORDS. For example,

```
    Small Systems:       DSA STANDARD DATA SET
                          (A1 ALPHA(10);
                           DSB ORDERED DATA SET
                                (B1 ALPHA(10)),BLOCKSIZE=5 TABLES,
                                             TABLESIZE=10 ENTRIES;
                         );

    A Series:            DSA STANDARD DATA SET
                              (A1 ALPHA(10);
                               DSB ORDERED DATA SET
                                   (B1 ALPHA(10)),BLOCKSIZE=50 RECORDS;
                              );
```

The change of BLOCKSIZE converts the DASDL while keeping the same physical attributes as the Small Systems DASDL. However, because of the differences in how the Small Systems and the A Series are implemented, the result of the change may be a less efficient use of disk space and I/O time.

FIND Statement

The difference between the Small Systems and the A Series FIND statements, though it does not require changes to your DASDL, is discussed here since it has to do with ordered data sets.

On Small Systems, access to the ordered data sets is through the FIND NEXT and FIND PRIOR statement. On the A Series, the syntax to access ordered data sets is either:

1. FIND <embedded data set> VIA NEXT <access name>

   or

2. FIND <embedded data set> VIA PRIOR <access name>.

The access to the ordered data set is through the access name. Change the A Series host language programs that access the ordered data sets.

LONG LISTS

If an embedded data set on the A Series has a large number of records per owner, changing to a embedded standard data set with an embedded set could improve performance.

UNORDERED EMBEDDED DATA SETS

The A Series does not support two levels of blocking. The following changes should be made to the Small Systems DASDL BLOCKSIZE and LONG LISTS.

BLOCKSIZE

Remove the Small Systems BLOCKSIZE specification and use the value of the Small Systems TABLESIZE as the value for the A Series BLOCKSIZE. See the other discussion about BLOCKSIZE earlier in this section.

MANUAL SUBSETS

The A Series does not support two levels of blocking or fast subsets. The following subsections describe how to change the Small Systems DASDL.

Blocking Levels

Since the A Series does not support two levels of blocking, entries are grouped into tables but tables are not grouped into blocks. This difference in handling can mean a substantial increase in the disk usage.

FAST Subsets

The A Series does not support FAST subsets. This means that each master record that has one or more subset entries uses at least one disk segment in the subset file. This, too, can mean a substantial increase in the disk usage. However, if this feature is needed, investigate the various types of LINKS the A Series supports.

SETS AND AUTOMATIC SUBSETS

The following subsections describe the changes necessary to convert sets and automatic subsets.

Group Keys

The Small Systems allow group items as keys but always require the RPG or COBOL program to specify each of the elementary items in all selection expressions.

The A Series requires that the RPG or COBOL language specify the group item in the selection expression. There are two ways to change Small Systems DASDL to A Series DASDL:

1. Change the DASDL to specify all the elementary items as keys.

2. Change the COBOL or RPG programs that use the group key.

Leaving the key as a group item reduces the usefulness of the general selection expression. Further, leaving the key as a group item and changing the programs may introduce semantic differences in group and elementary compares. For these reasons, we recommend that you change the DASDL, not the programs.

Indexed Random

We recommend that the MODULUS for index random structures be an odd number.

REMAPs

The A Series REMAP READONLY and OBJECT DATASET are different than on Small Systems.

READONLY and READONLY ALL

On Small Systems, READONLY ALL is treated as though READONLY had been specified for each data item in the data set. On the A Series, while READONLY is treated just as it is on Small Systems, READONLY ALL is treated quite differently. If a data set is marked READONLY ALL, you cannot do a DELETE, REMOVE, INSERT, or STORE. Attempting any of these actions causes a DATAERROR exception. In addition, if you have READONLY ALL on the data set, you can not have READONLY specified for single items within that data set.

Because of these differences, if you have a program that does a STORE, DELETE, REMOVE, or INSERT on a data set, you must remove the READONLY ALL and put READONLY on those items to be READONLY. If you are not doing a STORE, DELETE, REMOVE, or INSERT on the data set, you may leave the READONLY ALL intact when you progress to the A Series. If you have READONLY and READONLY ALL in the same data set, you must remove one of them.

The A Series does not allow READONLY on occurring items.

OBJECT DATASET in <REMAP-SUBSET-NAME>

The Small Systems syntax for this is <remap-subset-name> = <subset name> OF <object dataset> OR <remap name>.

The A Series does not allow the specification of the OBJECT DATASET or REMAP-NAME. Therefore, the Small Systems syntax must be changed to the A Series syntax of <remap-subset-name> = <subset name>.

LOGICAL DATA BASES

As described below, the security feature of the A Series is different than that of Small Systems.

Security

On Small Systems, security is specified with the SECURITYGUARD clause and file names are in the A/B/C format. The SECURITYGUARD file is accessed at DASDL compile time, thus requiring a recompilation of DASDL to change security.

On the A Series, security is specified with the GUARDFILE clause and files are names in the B/C ON A format. The guardfile is accessed by the accessroutines when the data base is opened by a program.

ADDITIONAL FEATURES

The following is a list of features available on the A Series that are not available on Small Systems:

1. The data base can be dumped while it is being updated.

2. Audit files can be duplicated.

3. Part of a file can be recovered while the data base is on-line.

4. Structures can be spread across multiple packs (a family).

Information about these features is available in the "A Series DMSII DASDL Reference Manual."

12 SMCS to COMS

This section describes the process of progressing from the Small Systems Supervisory Message Control System (SMCS) to the A Series Communication Management System (COMS).

COMS

COMS is a message control system (MCS) that controls on-line environments. COMS provides an integrated MCS that supports the needs of single and multi-station remote files.

COMS provides a window feature that allows you to operate a number of program environments independently and simultaneously. Each window has a name that can be specified in commands to perform various functions involving the program environment in that window. For information about COMS windows, refer to the A Series COMS Operator's Guide.

The COMS Utility program is the counterpart to the SMCS Jobs file. The Utility program is used to define and change specifications in the COMS configuration on-line without bringing down the MCS. COMS also provides a Utility Window where the Utility program runs. The Jobs file specifications must be manually converted to a format acceptable to the Utility program.

All Small Systems source programs will need to be recompiled on an A Series system after making any necessary changes required by the A Series compiler.

COMS provides access to CANDE on the A Series through the CANDE window. The COMS ON and PASS commands work like their SMCS counterparts and provide access to the CANDE window. Additionally, COMS provides a multiple session capability (called dialogs) that allows up to eight simultaneous CANDE sessions. While only one session can be current, all sessions may be passed to. Outputs for non-current dialogs are automatically suspended until that dialog becomes the current dialog through an ON command or the dialog is manually resumed.

COMS also provides access to a window called Menu-Assisted Resource Control (MARC). This window provides a menu-driven interface to A Series systems, while also allowing direct entry of commands. The menu-driven facility allows access to the system without having any knowledge of A Series commands. Command mode is convenient for anyone familiar with system commands. For more information about MARC refer to the A Series Menu-Assisted Resource Control (MARC) User's Guide.

In a similar fashion, remote-file application programs can be accessed through ON and PASS commands to the remote-file program's window after the program and window have been set up through the COMS Utility program. Like the CANDE and MARC windows described previously, your remote-file window's output will be suspended whenever it is not the current window, unless manually overridden.

<u>SIMPLE REMOTE FILES</u>

To access simple remote files, first load the source file and then compile it, using the CANDE COMPILE command. To initiate the program, enter EXECUTE or RUN <program-name> and transmit. However, there are two choices to using the CANDE EXECUTE command. One choice is to do a MARC RUN. Another choice is to declare Remote File Programs and Windows to COMS through the COMS Utility.

To do a MARC RUN, use the RUN menu selection or the WFL "RUN" command. Using the MARC Run selection results in a series of up to three screens being displayed to gather information to initiate a synchronous task. The first screen in the series (the RUN screen) prompts you for the name of the code file to be executed and for any parameters or task values. The second screen (the FILEDEF screen) allows you to establish file equations and attributes for one or more files. The third screen (the TASKDEF screen) allows you to assign task attributes for the task to be initiated. For more information refer to the A Series Menu-Assisted Resource Control (MARC) User's Guide.

To declare Remote File Programs and Windows, perform the following installation procedure. Note that this procedure must be done for each program. You must be a COMS control-capable user or the station being used must be a COMS control-capable station in order to declare remote file programs.

<u>Declare a Program to COMS</u>

Use the following steps to declare a program to COMS.

1. Enter the command ?ON UTILITY to get onto the COMS Utility Window.

2. Enter either "P" in the Choice field or "GO P" in the Action field of the HOME MENU. This will take you to the Program Activity screen.

3. Enter CReate in the Action field.

4. Enter the program name in the Program Name field.

5. Enter the title as it appears on disk without the usercode in the Title field.

6. Enter the usercode in the Usercode field.

7. Enter "Yes" in the Remote File Interface field.

8. Enter the number of users in the Remote Users field.

9. Transmit from the Home position on the terminal. The system responds with a message indicating that your program has been created. Once your program is created, you must declare a window to COMS.

<u>**Declare a Window to COMS**</u>

Use the following steps to declare a window in COMS.

1. Enter "GO W" in the Action field of any Utility screen and transmit. The Window Activity screen will be displayed.

2. Enter CReate in the Action field.

3. Enter a window name that corresponds with the program name previously declared in the Window Name field.

4. Change the Remote Files field to "Y".

5. Enter the name of the previously created program name in the Remote File Program field and transmit.

This will cause a COMS window to be created, which means a window containing your program is now available for use.

For more information refer to the "A Series Communication Management System (COMS) Planning and Installation Manual."

HOW TO RUN PROGRAMS

If you are running your program in CANDE, enter RUN <program name>. If you are running the remote file program that you just declared in COMS, enter ?ON <window name>. To run a disabled COMS remote file program, enter ?ENABLE PROGRAM <program name> and then ?ON <window name>. For more information refer to the "A Series Communication Management System (COMS) Operators Guide."

BRINGING DOWN PROGRAMS

For declared Remote File Programs, you have the option to set a time limit for the program to run before it automatically ends if no input has been received. This time is specified in the Time Limit field on the Program Activity screen in the COMS Utility. To end a program immediately, enter ?DISABLE PROGRAM <program name> and transmit. For more information refer to the "A Series Communication Management System (COMS) Operators Guide."

B1000 CDs

On A Series, there are two forms of the COBOL74 CD. The first is the standard EBCDIC CD data area, which is compatible with the B1000 COBOL74 CD. The second form of the COBOL74 CD on the A Series is the BINARY CD which is used for the COMS direct window interface.

On the A Series, the COBOL74 compiler builds references to a library called DCILIBRARY whenever a program uses the ACCEPT, DISABLE, ENABLE, RECEIVE, or SEND statements. A sample DCILIBRARY is provided for use with the EBCDIC CDs. We recommend the use of this library, which maps the CD into a remote file. You can also write your own DCILIBRARY. See the A Series COBOL74 Reference Manual for further details.

COMS provides a DCILIBRARY interface for COBOL74 programs that use the BINARY CDs (the COMS direct window interface). See the "A Series COMS Programmer's Guide" for more information on COMS direct windows.

REMOTE FILES WITH SIMPLE HEADERS

The A Series does not handle the function of remote files with simple headers. To approach this problem, you should convert the COBOL program into a remote file without simple headers and determine what functionality is lost by this. Then examine the lost functionality to see if it can be handled by remote file attributes. (For more information about remote file attributes refer to the A Series I/O Subsystem Reference Manual.) If this method is insufficient and there is no file attribute to handle the functionality, the user should go to a COMS direct window interface. (For more information about COMS direct windows refer to the A Series COMS Programmer's Guide.)

USER MCSs - REMOTE FILES WITH HEADERS

User MCSs will have to be evaluated on an individual basis to examine what functions their user MCS allows that COMS will not allow. Knowledge of both COMS and the user's functional requirements is necessary to help determine how to progress to COMS.

JOBS FILE

The following is a list of the features of the Jobs File in SMCS and their COMS equivalent.

SMCS	COMS
PROGRAMID	The window name as declared to the COMS Utility. Used with ?ON and ?PASS commands.
AUTO-START	On the COMS Program Activity screen the minimum copies attribute is set to one. This causes one copy of the program to be executed when COMS is initiated.
COPIES	On the COMS Program Activity Screen, the attribute is MAXIMUM COPIES = <integer>.
EXCEPT	A Series systems does not provide an exception indicator for remote files. COMS supports the "EOF on a station basis" capability.

LOG-ON	There is no COMS equivalent. If the program is a single user program, the Open Notification Text field on the Window Activity Screen can be used to simulate this feature.
MESSAGES	There is no COMS equivalent.
NO-EOF	If Minimum Copies is set to a value greater than zero (0), COMS will not send an EOF indication to the program when all users have signed off.
NO-RR	There is no COMS equivalent.
NO-SCROLL	There is no COMS equivalent.
NO-ZIP	There is no COMS equivalent. This feature is used for Station Transfer only.
USERCODE	In the COMS Utility, on the Usercode Activity screen there is a Valid Window List field. You should specify the windows (programs) that are valid for this usercode.
USERFILE	On the COMS Usercode Activity screen specify the windows (programs) in the Valid Window List field that are valid for this userfile.
HR:MIN:SEC	This is the Time Limit field on the COMS Program Activity screen.
<zip string>	There is no COMS equivalent. COMS does allow a set of file attributes and task attributes to be specified for a program execution, except for those attributes specifically listed on the COMS Utility Program Activity screen.

COMMAND FUNCTIONALITY

The following is a list of SMCS commands and their COMS equivalent. For more information refer to the "A Series COMS Operator's Guide," the "Small Systems Supervisory Message Control System (SMCS) Reference Manual" and the "A Series Operator Display Terminal (ODT) Reference Manual."

SMCS to COMS

SMCS	COMS
ATTACH	The ATTACH command allows a station to be dynamically attached to COMS.
BACKGROUND	There is no functional equivalent in COMS.
BROADCAST	This is the TO ALL command in COMS.
BYE	This is the same in COMS.
CHANGE	There is no functional equivalent in COMS.
CLEAR	There is no functional equivalent in COMS.
CLOSE	There is no functional equivalent in COMS.
CONTINUE	There is no functional equivalent in COMS.
DETACH	The equivalent in COMS is ?CLOSE WINDOW <window name>. It detaches the station from the window. The station can no longer input to that window.
DUMP	There is no functional equivalent in COMS, but the A Series has the ability to dump a program.
DYNAMIC	There is no functional equivalent in COMS.
FILE	First, select FILE from the MARC home menu, then select DETAIL from the FILE screen. Finally, specify the file name you want to see.
HARDWARE	There is no functional equivalent in COMS.
HELP	This is the same in COMS.
ID	There is no functional equivalent in COMS.
INITIATE	When setting up a remote file program in the COMS Utility, set the minimum copies attribute to 1.
JOBS	This is the same in COMS except that the window in which the program is running is also displayed.
LOGON	There is no functional equivalent in COMS.

MACRO	There is no functional equivalent in COMS, but this can frequently be handled by a WFL Job. (Refer to the "A Series Work Flow Language (WFL) Reference Manual.")
MAIL	There is no functional equivalent in COMS.
MAKE	The COMS equivalent of the SMCS MAKE STATION READY command is READY. The COMS equivalent of the SMCS MAKE STATION NOT READY command is SAVE.
MOVE	This is similar to the COMS RECALL command except you can only RECALL ALL. Selective messages can not be recalled from one station to another station.
NEWS	There is no functional equivalent in COMS, but MARC handles this function. A NEWS command can be entered from the MARC window.
PASS	This is the same in COMS except that COMS suspends output to passed windows by default.
PASSWORD	COMS PASSWORD is used to change a password associated with a usercode.
READY	This is the same in COMS.
REMOVE	The COMS PURGE command is the equivalent except all messages must be purged in COMS as opposed an individual message in SMCS.
REPORT	This is the same in COMS.
SEND	The COMS TO command is the equivalent.
SIGN ON	The COMS ON command is the equivalent except you enter ON <window name> instead of ON <program name>.
SIGN OFF	The COMS CLOSE command is the equivalent.
SIGNAL	There is no functional equivalent in COMS.
STATUS	The COMS STATUS command is the equivalent.
STOP	The QUIT COMS command is the equivalent.

SMCS to COMS

SYSTEM	There is no functional equivalent in COMS.
TABS	There is no functional equivalent in COMS.
TERMINATE	There is no functional equivalent in COMS.
TRACE	This is the same in COMS.
TRANSLATE	There is no functional equivalent in COMS. All characters received by COMS will be treated as uppercase characters. COMS makes no distiction between lowercase and uppercase.
USER	The COMS HELLO command is the equivalent. The MARC LOGON screen will appear by default when the system comes up.
ZBACKGROUND	There is no functional equivalent in COMS.
ZIP	This function is handled with the equivalent ODT command processed through the MARC window.
?	This function is handled with the equivalent ODT command processed through the MARC window.

13 GEMCOS/COMS

The Small Systems GEMCOS is a generative system; the A Series COMS is a code file shipped to you ready to run. A Format Support Library (FS-LIB) is provided with COMS to handle the Small Systems GEMCOS TCL Formatting. The rest of the Small Systems GEMCOS TCL statements must be progressed to COMS Utility program input. The Small Systems GEMCOS programs must be manually progressed to the A Series COMS format.

GENERAL INFORMATION ABOUT COMS

COMS is a message control systems (MCS) that controls on-line environments. COMS provides an integrated MCS that supports the needs of multi-program transaction processing as well as single and multi-station remote files.

COMS provides a window feature that allows you to operate a number of program environments independently and simultaneously. Each window has a name that can be specified in commands to perform various functions involving the program environment in that window. For information about COMS windows, see the "Window Feature in COMS" section in the "A Series COMS Operator's Guide."

COMS Utility program is the counterpart to the GEMCOS MCSTIC file. The Utility program is used to define and change specifications in the COMS configuration on-line without bringing down the MCS. COMS also provides a Utility Window where the Utility program runs. The GEMCOS TCL files must be converted into a format that the Utility program understands.

COMS, unlike GEMCOS, does not explicitly handle formatting. However, a Format Support Library is provided to handle the Small Systems GEMCOS TCL formatting. The Small Systems TCL formatting is inserted into a skeleton A Series TCL file. This skeleton is provided with the Format Support Library. The A Series GEMCOS Utility is then run with this TCL as input, creating the input for the Format Support Library. Processing Item. The TCL formats can then be maintained and updated as needed.

Formatting for COMS can also be done through the Screen Design Facility (SDF). SDF is a tool to help programmers design and process forms for applications simply and efficiently. With it, you interactively create form images, and use a series of screens to define form and field attributes. The forms created with SDF are maintained in a generated form library. COMS can be directed to access this library to pre-process input messages as well as post-process output messages. For information about the Screen Design Facility, refer to the "A Series Screen Design

Facility User's Guide." For information about COMS, refer to the "A Series COMS Planning and Installation Guide."

TCL FORMATTING

One of the major problems users encounter when progressing from GEMCOS to COMS is formatting. Progression aids are available so you will not lose any formatting capabilities after progressing to COMS. By using the Format Support Library, you can run TCL formats directly under COMS. This allows the GEMCOS functions and formats to be maintained in current symbolic form. To do this, the functions and formats along with the trancodes and device mappings must be maintained in the A Series GEMCOS TCL, which will be processed by the A Series GEMCOS Utility to produce run-time files. (The GEMCOS Utility must be level 800 or higher and is included with the Format Support Library.) These run-time files will be used by the Format Support Library. This library will perform input formatting, output formatting, and forms requests exactly as A Series GEMCOS performs them.

Maintenance of TCL

In order to have functions, formats, trancodes, and device mappings maintained, you must follow the A Series GEMCOS TCL syntactic conventions. A skeleton TCL file is available for you to insert various pieces of this TCL into the appropriate spaces. The skeleton TCL file is called SYMBOL/COMS/FORMAT/SUPPORT/LIBRARY/EXAMPLE/SKELETON and is available with the Format Support Library.

The following table describes the parts of the skeleton TCL file.

FUNCTIONS and FORMATS	Enter the functions and formats from your GEMCOS TCL.
SYSTEM <name>	Enter a name which corresponds to the COMS window name.
PROGRAM	The dummy program is already provided in the skeleton.
INPUT QUEUE	This is a dummy syntactical item which does not occur in Small Systems TCL. It is provided in the skeleton.
MESSAGE KEYS	These are the same as trancodes in Small Systems TCL. The syntax for message keys on the A Series is "MKE". They must be

	entered under INPUT QUEUE by changing the reserved word TRANCODE to the reserved word MKE.
STATIONS	These are dummy station names. Some are provided in the skeleton. However, there must be one dummy station for each device type. Therefore, you must add one station name for each consecutive device that you want to add to the skeleton TCL.
DEVICE	Insert the TCL Device section from your Small Systems TCL. The devices declared here must correspond exactly to the devices declared in COMS.
STALIST	For each device, there must be at least one dummy station that was declared in the previous STATIONS section.
FORMATSIN	This does not change from Small Systems.
FORMATSOUT	This does not change from Small Systems.

This is your new A Series TCL. Any modifications or enhancements to your formatting will be made to this TCL. (See "Format Update" later in this section.)

Run GEMCOS Utility

The GEMCOS TCL file must be compiled, without any syntax errors, with GEMCOS/UTILITY release 8.0 or higher. This is necessary to produce the run-time files used by the TCL Formatter Processing Item. The GEMCOS/UTILITY and a sample job deck are included with the Format Support Library.

The Format Support Library needs information from the device and format sections in the GEMCOS TCL file. In addition, it needs to know the system names defined in the TCL file so that the correct format can be applied depending on which window the user is on.

The following files must be file equated. The variable <codefile name> is the code file name of the library containing the TCL Format Processing Item. These files must have a security classification of PUBLIC IO.

IQUS	<codefile name>/INPUT
OQUS	<codefile name>/OUTPUT
CQUS	<codefile name>/CONTROL
FMTFILE	<codefile name>/FORMAT

COMS Configuration Items

In addition to the normal configuration information specified in the COMS Utility program, the following steps must be taken to configure the COMS Environment for the Format Support Library.

1. Ensure that the window name defined to COMS through the Utility is the same as the system name specified in the TCL. The COMS Utility input is:

   ```
   CREATE WINDOW <window name>;
   ```

2. Ensure that the trancodes (Message Keys) from the TCL match the COMS Utility input:

   ```
   CREATE TRANCODE <trancode name> OF <window name>
   AGENDA = <agenda name>
   FUNCTION = <integer>;
   ```

   It should be noted that all trancodes must start in the first position.

   For more information see the TRANCODE statement in the TCL subsection later in this section.

3. Ensure that the devices declared in the TCL match the COMS Utility input:

   ```
   CREATE DEVICE_TYPE = <device_type name>;
   CREATE STATION <station name>
        DEVICE_TYPE = <device_type name>;
   ```

4. Create a library. The Format Support Library is specified to COMS through the Utility command:

   ```
   CREATE LIBRARY <library name>;
   ```

5. Create a processing item with an ACTUALNAME of TCL_FORMATTER and specify the previously created library. The processing item is specified to COMS through the following Utility command:

```
    CREATE PROCESSING_ITEM <processing item name>
         ACTUALNAME = TCL_FORMATTER,
         LIBRARY = <library name>;
```

6. Create a processing item list which contains the previously created TCL_FORMATTER processing item or insert the TCL_FORMATTER processing item in an existing processing item list. This processing item list should be applied to each input or output message needing formatting. The processing item list is specified to COMS through the following Utility command:

```
    CREATE PROCESSING_ITEM_LIST
         <processing_item_list name> =
              <processing_item name1, name2, name3,...>;
```

The processing item list is applied to the input messages through the Utility command:

```
    CREATE AGENDA <agenda name> of <window name>
    PROCESSING_ITEM_LIST = <processing_item_list name>,
    DESTINATION = <program name>;
```

FORMSREQUEST

This is the same as in GEMCOS.

Format Update

To update formats, run the GEMCOS Utility against the updated TCL. Then disable the library by entering "?DISABLE LIBRARY <library name>". For more information about updating formats see the "A Series COMS Operations Guide."

Programming Changes

The application programs must be changed to use the COMS direct interface. (See the COMS Programmer's Guide for more information.) For output formats, the FORMID should be placed in the first 6 bytes of the COMS conversation area. If there is no conversation area the processing item will discontinue (DS) with an invalid index.

Screen Design Facility (SDF) Coexistence

The Format Support Library can coexist with SDF. This is allowed by having SDF recognize when a message has already been formatted. To do this, the Format Support Library will set the first 48 bits of the conversation area to all "1"s when formatting occurs. The SDF processing item should be placed after the TCL_FORMATTER processing item in the agenda's processing item list for both input and output messages.

Monitor Output

The Format Support Library will send format exception notification to the COMS monitor stations. This notification is directed to the same window on the monitor station that it occurred on at the user station. Therefore, if the notification is desired, the monitor station must have the same window name (the GEMCOS system name) declared and that window must be open.

TCL

This subsection compares Small Systems GEMCOS statements to the A Series COMS commands. The GEMCOS TCL statements are listed with their COMS equivalent, if any, or with a description of how COMS handles what the GEMCOS statement controlled. In many cases, GEMCOS statements are not applicable to the COMS systems.

The GEMCOS statements are listed according to the TCL section in which they occur.

Compiler Statements

GEMCOS Statements Not Necessary On COMS

CONTROL STATEMENT	This statement has no direct mapping into COMS. The only applicable CONTROL STATEMENT option is REPORT, which maps into the COMS REPORT command in the COMS Utility program.
FORMAT FILE NAME STATEMENT	COMS does not directly do formatting. However, formatting can be done via a processing item in COMS. See "General

Information About COMS" and "TCL Formatting," earlier in this section.

MCSTIC FILE NAME STATEMENT

The GEMCOS MCSTIC file maps into the COMS configuration file. The following shows how to file equate the configuration file when COMS is initiated:

```
FILE CFILE(TITLE = <filename>,
FAMILYNAME = <packname>);
```

Global Section

GEMCOS TCL Statements Not Necessary ON COMS

The following lists those GEMCOS TCL Global Section statements that are not required to operate COMS. Where appropriate an explanation of how COMS handles the TCL statement is given with the TCL statement.

AUDIT FILE FAMILY ID STATEMENT	
AUDIT RECORD SIZE STATEMENT	The size of an audit (transaction trail) record is automatically optimally determined by COMS.
CHANGE REQUESTS STATEMENT	
CHECKPOINT INTERVAL STATEMENT	
CONVERSATION LIMIT STATEMENT	
DATA DUMP STATEMENT	
FORMAT and FUNCTION LIST STATEMENT	COMS does not handle formatting. However, formatting can be done through a processing item in COMS. Refer to "General Information About COMS" and "TCL Formatting" earlier in this section.
TEXT SIZE STATEMENT	
MESSAGE BROADCAST STATEMENT	
MESSAGE RECALL STATEMENT	
MONITOR TRACE STATEMENT	
NAME-STACK ENTRIES STATEMENT	
NCC OK RESPONSE STATEMENT	
OBJECT CODE FILE NAME STATEMENT	This statement refers to the name of the generated GEMCOS MCS code file.

PROGRAM BOJ EOJ STATEMENT
QUEUE BUFFERS STATEMENT
QUEUE DEPTH STATEMENT
QUEUE NAME STATEMENT
RECALL PROGRAM STATEMENT
SIMULATION STATEMENT

SOURCE CODE FILE NAME — This statement refers to the name of the generated GEMCOS MCS source file.

STATUS REPORTS STATEMENT
SUBORDINATE MCS STATEMENT
SYSTEM HALT STATEMENT
VALUE-STACK BITS STATEMENT

GEMCOS TCL Statements With Their COMS Equivalents

The following is a list of GEMCOS TCL statements with their COMS equivalents. Where an example is given, the example may not be syntactically complete; there are other attributes available for these COMS commands but only the relevant information has been given.

AUDIT FILE PACK ID STATEMENT — This GEMCOS statement maps into COMS as an attribute of the data base command of the Utility program as follows:

```
CREATE DATA_BASE <data base name>
FAMILYNAME = <packname>;
```

AUDIT PAGE SIZE STATEMENT — This GEMCOS statement maps into COMS as an attribute of the DATA BASE command of the Utility program as follows:

```
CREATE DATA_BASE <data base name>
AREASIZE = <integer>;
```

COMPILE OPTIONS STATEMENT — There is only an indirect mapping of this statement into COMS. Since COMS is not generative, the need to compile COMS will arise infrequently. Should COMS ever need to be compiled, Appendix C of the "ALGOL Reference Manual" describes the usage of compiler control records.

MONITOR TRACE ON STATEMENT — This function can be accomplished in COMS by use of the VALUE task attribute on the RUN statement when COMS is started. The numeric value of the VALUE attribute depends on what other COMS options controlled by this attribute have been selected. To correctly set the value, add 4 to the value normally selected. The default value is 0.

Definition Section

ACCESS CONTROL STATEMENT

GEMCOS requires that access keys (users) be defined separately from the usercode structure supported by the systems software. COMS works in conjunction with the systems software USERDATAFILE and uses those usercodes. The ACCESS CONTROL statement can be implemented in COMS using the COMS Utility command:

```
MODIFY USERCODE <usercode name>
WINDOW_LIST = <wl name>
SECURITY_CATEGORY_LIST = <scl name>;
```

This assumes <usercode name> is already known to the Utility. If the usercode is not known, then CREATE is used instead of MODIFY. <wl name> is a previously defined Utility Window List element that contains a list of all valid windows for this usercode. This usercode then has access to all programs belonging to these windows. <scl name> is a previously defined Security Category List element that contains a list of all valid security categories. The usercode then has access to all trancodes associated with these security categories. The GEMCOS keyword ALL maps into COMS as a replacement of <wl name> and <scl name>.

Definition Section (Program)

When a program is defined to GEMCOS, a program classification must be included (ASSIGNMENT, UTILITY, USER, Pass, or PORT). COMS program classifications are based on the window to which they are assigned. If a program uses the remote-file interface, it belongs to a remote-file window and is considered a remote-file program. If the program is an MCS, it belongs to an MCS window. If the program is a TBR program (it uses trancodes) or requires pre- or post-processing of its messages, it belongs to a direct window. The USER classification maps into a direct COMS window program. The other two classifications need to be handled on a program-by-program basis.

GEMCOS TCL Statements Not Necessary On COMS

This subsection lists those GEMCOS TCL Program Section statements that are not required to operate COMS. Where appropriate an explanation of how COMS handles the TCL statement is given with the TCL statement.

AP300STATUS STATEMENT	
ATTACH MESSAGE STATEMENT	
AUDIT OUTPUT STATEMENT	Since COMS audits are taken at mid-transaction, there is no copy of the after transaction image (output transaction) available to audit.
COMMON SIZE STATEMENT	
DETACH MESSAGE STATEMENT	
OPEN MESSAGE STATEMENT	
PLMPROGRAM STATEMENT	The function of this GEMCOS statement is to facilitate the ability to connect a given station to another host system. In GEMCOS, this is accomplished by executing the designated program and entering a CONNECT command to this program. In COMS, the CONNECT statement is a COMS command and can be directly entered from a station without the use of any program.
RESIDENCE STATEMENT	
RESTART PROGRAM STATEMENT	
SUPPRESS GOOD DAY MESSAGE STATEMENT	
TRANSACTION CODE POSITION STATEMENT	Whenever COMS does trancode routing from a program, the starting position of the trancode is always assumed to be the first character of the message.

GEMCOS TCL Statements With Their COMS Equivalents

The following is a list of GEMCOS TCL statements with their COMS equivalents. Where an example is given, the example may not be syntactically complete; there are other attributes available for these COMS commands but only the relevant information has been given.

AUDIT TRANSACTIONS and AUDIT ASSIGNMENT STATEMENTS	In GEMCOS, a distinction is made between messages with trancodes and messages

without trancodes and you can specify which messages to audit. In COMS, all update transactions are automatically audited. You can elect to also have inquiry transactions audited (the default does not include the audit of inquiry transactions).

To have inquiry transactions audited along with the update transactions, use the Utility command:

```
CREATE DATA_BASE <data base name>
AUDIT = Y;
```

CONVERSATIONSIZE STATEMENT

In GEMCOS, the conversation area is passed as part of the message itself. In COMS, the conversation area is passed as part of the input and output CD. The input and output CD are formatted by the user according to the needs of each program. The CD field names associated with the conversation area are COMS-IN-CONVERSATION and COMS-OUT-CONVERSATION.

DATA BASE NAME STATEMENT

The name of the data base associated with a program is specified in COMS through the Utility program as:

```
CREATE PROGRAM <program name>
DATA_BASE = <data base name>;
```

The DATA BASE NAME statement is used only with SYNCHRONIZED recovery.

EXECUTE STATEMENT

The three GEMCOS EXECUTE options (ONDEMAND, BOJ, and MANUAL) map onto COMS as follows:

If EXECUTE is ONDEMAND, the program must belong to a direct window and the MIN_COPIES attribute of the PROGRAM definition command must be set to zero. The program will start upon receipt of the first message directed to the program.

If EXECUTE is BOJ, then the MIN_COPIES attribute of the PROGRAM command must be set to a value greater than zero. EXECUTE BOJ is window independent.

If EXECUTE is MANUAL, then the program must be either an MCS (MCS window) or remote-file (remote window) program. The receipt of the first ON <window name> COMS command will start the program. The MIN_COPIES attribute must be set to zero.

INTERFACE STATEMENT

The three GEMCOS INTERFACE options (MCS, PARTICIPATION, and NONPARTICIPATION) map onto COMS as follows:

If INTERFACE is MCS, then the program belongs to an MCS window and is defined to the Utility as follows:

```
CREATE WINDOW <window name> MCS = Y,
TITLE = <program title>;
```

Once the window <window name> is defined, the user can switch to the MCS program by entering the COMS command:

```
ON <window name>.
```

IF INTERFACE is PARTICIPATION, then the program belongs to a direct window and is defined to the Utility as:

```
CREATE PROGRAM <program name> REMOTE_FILE =
N;
CREATE WINDOW <window name>;
CREATE AGENDA <agenda name> OF <window
name>
DESTINATION = <program name>;
```

In COMS, direct windows require a program to be assigned as the destination of an agenda and the agenda to be assigned to a direct window. Additionally, trancodes can be assigned to agendas. This provides the capability of having multiple programs assigned to the same direct window since each agenda in the direct window specifies a destination program.

If interface is NONPARTICIPATION, the program belongs to a remote-file window and is defined to the Utility as:

GEMCOS/COMS

```
CREATE PROGRAM <program name>
REMOTE_FILE = Y;
CREATE WINDOW <window name>  REMOTE_FILE  =
Y,
REMOTE_PROGRAM = <program name>;
```

MAXIMUM ASSIGNERS STATEMENT

While GEMCOS is structured in terms of the number of stations that can concurrently talk to a program, COMS is structured in terms of the number of users that can concurrently talk to a window, thus to a program. In addition, COMS restricts the usage of windows not of programs. For remote-file windows, there is always one program for each window. For direct windows, which can contain more than one program, access is limited to the number of users of the window. This GEMCOS statement maps onto COMS through the Utility program with the command:

```
CREATE WINDOW <window name>
MAX_USERS = <integer>;
```

MAXIMUM COPIES STATEMENT

The MAXIMUM COPIES statement in COMS is functionally the same as in GEMCOS. The GEMCOS statement maps into COMS through the Utility program with the command:

```
CREATE PROGRAM <program name>
MAX_COPIES = <integer>;
```

PROGRAM TITLE STATEMENT

The GEMCOS PROGRAM TITLE STATEMENT maps onto COMS through the Utility program command as:

```
CREATE PROGRAM <program name>
TITLE <program title>;
```

The USERCODE and FAMILY attributes can also be specified with this command to determine the usercode and family assignment for the program.

RECOVERY STATEMENT

While there are three types of recovery available with GEMCOS, COMS currently has only one type of recovery. This is a synchronized DMSII recovery similar to the synchronized recovery available in GEMCOS.

The GEMCOS Queuerestoration recovery is not available on COMS. The GEMCOS SYNCHRONIZED recovery maps into the COMS SYNCHRONIZED recovery. The GEMCOS DATABASE recovery also maps into the COMS SYNCHRONIZED recovery. The COMS SYNCHRONIZED recovery is a recovery functionally equivalent to the GEMCOS SYNCHRONIZED recovery.

Recovery is specified in COMS through the Utility command:

```
CREATE PROGRAM <program name>
DATA_BASE = <data base name>;
```

TRANCODE STATEMENT

The GEMCOS TRANCODE statement maps onto COMS through the Utility program command as:

```
CREATE TRANCODE <trancode name> OF
<window name>
AGENDA = <agenda name>, FUNCTION =
<integer>;
```

In COMS, a trancode always belongs to a given window. This window must have been previously defined to COMS. In addition, an agenda must also have been defined to COMS and it must name a previously defined program as its destination (this is the program with which the trancode is associated). The two GEMCOS trancode

indices map into the FUNCTION attribute of the COMS command. The <integer> is passed to the destination program as the input CD field COMS-IN-FUNCTION-INDEX; the default integer is zero.

Definition Section (Station)

Station definition in GEMCOS is accomplished by declaring the station and its attributes in the TCL. In COMS, it is done by defining stations and attributes through the CREATE STATION command in the Utility program.

GEMCOS TCL Statements Not Necessary On COMS

This section lists those GEMCOS TCL Station Section statements that are not required to operate COMS. Where appropriate an explanation of how COMS handles the TCL statement is given with the TCL statement.

CONVERSATIONAL STATEMENT
HOST ACCESS KEY STATEMENT
SCREEN SIZE STATEMENT

SIGN ON STATEMENT

In COMS all stations require a user to sign-on. However, COMS also provides a feature called "auto logon" which will automatically log a user onto a station. To automatically log a user onto the station, use the following Utility command:

```
CREATE STATION <station name>
DEFAULT_USERCODE = <usercode name>;
```

STATION HOST NAME STATEMENT
TRANCODE STATEMENT (Station)
TRANSACTION MODE STATEMENT

TYPE STATEMENT

COMS supports the concept of device types. This is done through the assignment of a device type to every station declared to COMS. Then, for routing through a direct window, processing items (entry points in a library) can be invoked based on the device type. This allows user-written routines to be called when and where required based on the station's physical characteristics. To use this feature, the following Utility commands are required:

```
CREATE DEVICE_TYPE = <device_type name>;
CREATE DEVICE_TYPE_LIST <device_type_list
       name>=<device_type name>;
CREATE STATION <station name>
DEVICE_TYPE = <device_type name>;
CREATE PROCESSING_ITEM <processing_item
       name>
DEVICE_TYPE_LIST =<device_type_list name>;
CREATE PROCESSING_ITEM_LIST
      <processing_item_list name>=
      <processing_item name>;
CREATE AGENDA <agenda name> OF <window
      name>
```

```
PROCESSING_ITEM_LIST =
      <processing_item_list name>;
```

This example is not complete since it only specifies the required information to handle device types. This example also assumes the existence of a window named <window name>. The user-written library, including the entry point <processing_item name>, can be called on output to a station to format the data according to the physical requirements of the station.

This feature is not available for remote-file windows.

VIRTUAL STATION STATEMENT

GEMCOS TCL Statements With Their COMS Equivalents

The following is a list of GEMCOS TCL statements with their COMS equivalents. Where an example is given, the example may not be syntactically complete; there are other attributes available for these COMS commands but only the relevant information has been given.

CONTINUOUS LOG ON STATEMENT

This feature is the same for both GEMCOS and COMS. Continuous logon is specified in COMS using the utility command:

```
CREATE STATION <station name>
CONTINUOUS_LOGON = Y;
```

CONTROL STATION STATEMENT

This feature is the same for both GEMCOS and COMS. Control Station is specified in COMS using the Utility command:

```
CREATE STATION <station name>
CONTROL = Y;
```

MONITOR STATION STATEMENT

This feature is functionally equivalent in both COMS and GEMCOS, however, in COMS, it is declared in a different way. To declare a list of stations as Monitor Stations in COMS, a station list named MONITOR must be created. The COMS MONITOR command is used to control which COMS activities are monitored.

TRANSACTION CODE POSITION STATEMENT	This statement is functionally equivalent in GEMCOS and COMS. In COMS, this function is accomplished through the Utility command:
	
	`CREATE STATION <station name>`
	`TRANCODE_POSITION = <integer>;`

VALID ACCESS KEYS STATEMENT	In GEMCOS, for each station a list of valid users is specified. In COMS, a list of valid stations for each user is specified. While the implementations are different, they are functionally equivalent. The list of valid stations for a usercode are assigned with the utility command:
	
	`CREATE USERCODE <usercode name>`
	`STATION_LIST = <station-list name>;`
	
	<station-list name> must be a previously declared COMS Utility element that is a list of stations.

Device Section

In GEMCOS, the device section is only used in conjunction with formatting. COMS does not do formatting itself, so this section has no direct equivalent in COMS.

However, TCL Formatting can continue to be used with COMS via the Format Support Library. If you continue to use TCL Formatting, the input from the device sections is required. (See "TCL Formatting" earlier in the section.) COMS does support the concept of devices or device types apart from formatting. See the TYPE statement.

GEMCOS TCL Statements Not Necessary On COMS

This section lists those GEMCOS TCL Device Section statements that are not required to operate COMS. However, these statements are used by the Format Support Library. (See "TCL Formatting" earlier in this section.)

INPUT FORMATS STATEMENT
OUTPUT FORMATS STATEMENT
STATION LIST STATEMENT

Mess Code Section

COMS supports the concept of user-written code in a more flexible manner than GEMCOS. This is accomplished through user-written libraries that can be called to pre- or post-process a message that is handled through a direct COMS window. For more information, refer to Section 7 of the "A Series COMS Programmer's Guide."

COMMON-AREA HEADER

The concept of a message header is accomplished in COMS through the COBOL74 CD interface. The fields of both the input and output CD are described as to how COMS sends and receives information through this interface.

COBOL Interface Differences

The following section discusses the differences in the COBOL interface between GEMCOS and COMS. Emphasis is placed on the GEMCOS message header and the corresponding CD interface in COMS.

Message Header

The message header in GEMCOS is the common-area header and is a minimum of 60 bytes with a maximum of 200 bytes possible. In COMS, the message header is in the form of a COBOL74 input CD on a receive message and an output CD on a send message.

```
     GEMCOS COMMON-AREA HEADER LAYOUT
     --------------------------------

     01     COMMONAREA.
            05  MSGDESTINATION          PIC 9(1).
            05  LSN                     PIC 9(3).
            05  PGMNBR REDEFINES LSN    PIC 9(3).
            05  MTSMSGTYPE              PIC S9(1).
            05  SEQNO                   PIC 9(6).
            05  NDLTIME                 PIC 9(7).
            05  TEXTSIZE                PIC 9(4).
            05  TERMTYPE                PIC 9(2).
            05  MSGID                   PIC X(6).
            05  INDEX1                  PIC 9(2).
            05  INDEX2                  PIC 9(2).
```

GEMCOS/COMS

```
05  ERROR                    PIC 9(1).
05  FMTERR                   PIC 9(1).
05  MCSTYPE                  PIC 9(2).
05  INPUTADDR                PIC 9(9).
05  RETRYCOUNT               PIC 9(1).
05  RECOVERYSTATUS           PIC 9(1).
05  OUTPUTADDR               PIC 9(9).
05  CONVERSATIONSTATUS       PIC 9(1).
05  CONVERSATIONBOJEOJ       PIC 9(1).
05  USERAREA                 PIC X(n).
```

The size of the USERAREA (n) is a number between 1 and 140 when a user area is required.

__CD Interface__

The following is an example of a COMS CD interface.

```
INPUT CD
     COMMUNICATION SECTION.
     CD COMSIN USAGE BINARY ; FOR INPUT.
     01   CD-ARRAY-IN.
          03  COMS-IN-PROGRAM            PIC S9(11) USAGE BINARY.
          03  COMS-IN-FUNCTION-INDEX     PIC S9(11) USAGE BINARY.
          03  COMS-IN-USERCODE           PIC S9(11) USAGE BINARY.
          03  COMS-IN-SECURITY-DESG      PIC S9(11) USAGE BINARY.
          03  COMS-IN-DATE               PIC S9(11) USAGE BINARY.
          03  COMS-IN-TIMESTAMP          PIC S9(11) USAGE BINARY.
          03  COMS-IN-STATION            PIC S9(11) USAGE BINARY.
          03  COMS-TEXT-LENGTH           PIC S9(11) USAGE BINARY.
          03  COMS-IN-END-KEY            PIC S9(11) USAGE BINARY.
          03  COMS-IN-STATUS-KEY         PIC S9(11) USAGE BINARY.
          03  COMS-IN-RST-LOCATOR        PIC S9(11) USAGE BINARY.
          03  COMS-IN-CONVERSATION.
              05  COMS-IN-CONV-FLD-1     PIC X(10).
              05  COMS-IN-CONV-FLD-2     PIC 9(4).
              05  COMS-IN-CONV-FLD-3     PIC X(50).

OUTPUT CD

     COMMUNCIATION SECTION.
     CD COMS-OUT USAGE BINARY; FOR OUTPUT.
     01   CD-ARRAY-OUT.
          03  COMS-OUT-COUNT             PIC S9(11) USAGE BINARY.
          03  COMS-OUT-TEXT-LENGTH       PIC S9(11) USAGE BINARY.
          03  COMS-OUT-STATUS-KEY        PIC S9(11) USAGE BINARY.
          03  COMS-OUT-CONVERSATION.
```

```
05  COMS-OUT-CONV-FLD-1        PIC X(10).
05  COMS-OUT-CONV-FLD-2        PIC 9(4).
05  COMS-OUT-CONV-FLD-3        PIC X(50).
```

Common-area Header Compared To CD Interface

In COMS, many of the same functions present in the GEMCOS common-area header are in the COMS CD interface. The following is a list of the functions in the GEMCOS common-area compared to the COMS interface.

	GEMCOS	COMS
Restart Data Set Audit Locator	INPUTADDR	COMS-IN-RST-LOC
Source of Input	LSN	COMS-IN-STATION
Conversation Information	Start of the text	COMS-IN-CONVERSATION and COMS-OUT-CONVERSATION
Error Indication	MCSTYPE (recovery message)	COMS-IN-FUNCTION-INDEX
Time	NDLTIME	COMS-IN-TIMESTAMP
Text Length	TEXTSIZE	COMS-TEXT-LENGTH and COMS-OUT-TEXT-LENGTH
Function Index	INDEX1 and INDEX2	COMS-IN-FUNCTION-CODE

The usage of the additional fields in the input and output CDs are explained in Section 3 of the "A Series COMS Programmer's Guide."

NETWORK CONTROL COMMANDS

In GEMCOS, the Network Control Commands (NCC) consist of a signal character, a short mnemonic command code, and in some cases, one or more parameters. The Network Control Commands allow the user to communicate with the system through GEMCOS. In COMS, there are several categories of commands, called COMS commands, which accomplish many of the features of the GEMCOS Network Control Commands. In some instances, no COMS equivalent exists. In other instances, COMS provides certain functions that GEMCOS does not.

GEMCOS/COMS

The following is a list of GEMCOS commands and their COMS equivalents, if any. The COMS commands are entered interactively.

GEMCOS	COMS
HELP	To get HELP in COMS, you must be on the MARC menu selection screen. Once on the selection screen, move the cursor to either the "CC" or the "COMS" selection, then press "Spcfy". This will display a one or two-line explanation. If more information is needed, press "Spcfy" a second time for a detailed explanation.

Security Control Commands

DETACH FROM REMOTE FILE (DFR)	There is no exact COMS equivalent, however the COMS RELEASE and MCS commands can be used to simulate this function. Use these commands to release the station to another MCS.
DISABLE USER (DUS)	There is no COMS equivalent.
ENABLE USER (EUS)	There is no COMS equivalent.
SIGN OFF (BYE)	The COMS BYE command is used to end a session.
SIGN ON (SGN)	The COMS HELLO command is used to start a new session. If a session was already in progress, HELLO terminates the session and starts a new session. In GEMCOS, the usercode is given in the TCL and is not maintained by the Usercode Data File. In COMS, the usercode and password pairs are maintained by the system. In addition, COMS displays a secured line for the entry of the password.

Program Control Commands

EXECUTE PROGRAM (EX)	This GEMCOS command is handled in COMS via the MARC menu selection screen. Once on the MARC menu selection screen, enter RUN in the choice field.
FREE STATION FOR EXECUTION (FRE)	There is no COMS equivalent.
HALT APPLICATION PROGRAM (HAP)	The COMS DISABLE PROGRAM is used to terminate a program. Once a program is terminated, it cannot be run until it is enabled.
PROGRAM PASS COMMAND (PASS)	The GEMCOS and COMS PASS commands are functionally equivalent. However, COMS passes messages to windows rather than to programs.

MCS Control Commands

AUDIT OK (AOK)	There is no COMS equivalent.
HALT KILL	The COMS QUIT COMS NOW command is equivalent to the GEMCOS HALT KILL command. COMS also has a QUIT COMS DUMP command that terminates the MCS and produces a dump of what has occurred.
HALT SYSTEM (HLT)	The COMS QUIT COMS command is used to halt the COMS MCS.

Message Control Commands

BROADCAST (BRC)	The COMS TO command replaces the GEMCOS BROADCAST command. The ODT option of the TO command replaces the GEMCOS SPO (ODT in the 7.0 release) option. Both COMS and GEMCOS allow either a station name or a station LSN. To BROADCAST to all stations in GEMCOS, no options are specified; in COMS, the option ALL must be specified. COMS also allows

messages to be sent to either a window or usercode.

POP QUEUE (PQ)

COMS does not support all of the functions of the GEMCOS POP QUEUE command. However, subsets of the POP QUEUE command are equivalent to the COMS RECALL and PURGE commands.

COMS supports the capability to recall all of any station's messages and deliver them to another station. The example below shows the syntax of the COMS RECALL command that is equivalent to the GEMCOS POP QUEUE command.

```
GEMCOS
------
*PQ station1 ALL SEND station2

COMS
----
RECALL station1 TO station2
```

The other subset of the GEMCOS POP QUEUE command is equivalent to the COMS PURGE command. The PURGE command discards all messages queued for the specified station. The example below shows the syntax of the GEMCOS POP QUEUE that is equivalent to the COMS PURGE command.

```
GEMCOS
------
*PQ  station1 ALL

COMS
----
PURGE STATION station1
```

__REPORT Commands__

REPORT DATA DUMP (RDM)	There is no COMS equivalent.
REPORT FILE STATUS (RFS)	There is no COMS equivalent.
REPORT PROGRAM STATUS (RPS) and REPORT PROGRAM COUNTERS (RPC)	The COMS STATUS PROGRAM command is a combination of both of these GEMCOS commands. All of the information returned by the GEMCOS commands is returned by the

	COMS command, along with the window associated with the program and the name of the data base, if any.
REPORT STATION STATUS (RSS) and REPORT STATION COUNTERS (RSC)	The COMS STATUS STATION command is nearly equivalent to these two GEMCOS commands. The information returned by the two GEMCOS commands is returned by the COMS command, along with the usercode logged on, if any, and the window name. COMS also has a REPORT STATION command that gives a briefer report about the station.

CHANGE Commands

CHANGE MONITOR FLAG (CMF)	The COMS TRACE command is the equivalent to this GEMCOS command, with a few variations. GEMCOS traces all procedures of the MCS, COMS traces on either a station-by-station basis or a feature basis. Options for this command include: tasks, DCWRITE, and DCRESULT.
CHANGE STATION ADDRESS (CSD)	There is no COMS equivalent.
CHANGE STATION DIAGNOSTIC (CDS)	There is no COMS equivalent.
CHANGE STATION MAXIMUM RETRY (CSM)	There is no COMS equivalent.
CHANGE STATION READY (CSR)	This GEMCOS command is handled by the COMS commands READY and SAVE. The COMS READY command is used to make a station ready, the SAVE command is used to make a station not ready.
CHANGE STATION TRANSMISSION NUMBER (CST)	There is no COMS equivalent.
FORMAT UPDATE COMMAND (UPD)	There is no COMS equivalent; formatting is not directly handled by COMS. However, processing items that do formatting can be updated by disabling the library after an updated version is available. For example, DISABLE LIBRARY <name>. See the "Format Update" subsection for more information.

AUDIT And RECOVERY Commands

CLEAR DISABLED PROGRAM (CLE)	The COMS ENABLE PROGRAM is nearly equivalent to this GEMCOS command. However, the GEMCOS command enables the program and initiates a recovery cycle. The COMS command only enables the program, no recovery cycle is initiated.
REFRESH COMMAND (REF)	There is no COMS equivalent. Since COMS does not have an audited output message, there is no way to recall the last message for a station.
RECOVER DATA BASE (REC)	COMS automatically recovers affected data bases so there is no COMS command equivalent. However, if a manual recovery in COMS is necessary, follow these steps: 1. Disable the data base. 2. Make sure that the data base files and audit trails are loaded and valid. 3. Enable the data base.
RESET BUSY STATUS (RBS)	There is no COMS equivalent. Since COMS does not currently support the "transaction mode" capability for a station, there is no current need for an equivalent to this command.

ADDITIONAL COMS COMMANDS NOT IN GEMCOS

The following section describes commands that are available in COMS but not in GEMCOS.

PASSWORD Command

The PASSWORD command is used to change or delete the password associated with a given usercode or to alter a list of passwords if more than one password is associated with the usercode.

ON Command

In COMS, the ON command is used to move from one window to another.

CLOSE Command

The CLOSE command is used to close any dialog of a window or all dialogs of all windows at your station.

SUSPEND Command

The SUSPEND command is used to suspend messages from one or more window dialogs at your station. When messages from a window dialog other than the current window dialog are suspended, all messages from that dialog are tanked (not displayed) until that window dialog is resumed or becomes the current dialog again. For example, if you suspend dialog 1 of the CANDE window, all messages for dialog 1 of the CANDE window are suspended unless the CANDE window is your current window dialog. Refer to the "A Series COMS Operator's Guide" for a discussion about Window Dialogs.

RESUME Command

The RESUME command is used to resume the display of messages from one or more window dialogs at your station.

CONTROLLING STATIONS

The following seven commands control stations in the COMS network.

ADDSTA COMMAND	The ADDSTA command adds stations to a line.
ATTACH COMMAND	The ATTACH command controls stations that are not currently attached. Attaching a station allows COMS to send messages (output) to that station. When the attached station is enabled, all input from that station is received by COMS.
DISABLE COMMAND	The DISABLE command disables stations in the COMS network. When a station

	is disabled, it will not be polled for input, but it can still receive output.
ENABLE COMMAND	The ENABLE command enables stations in the COMS network. When a station is enabled, it will be polled for input if it is in the ready status.
MOVE STATION COMMAND	The MOVE STATION command moves stations to another line.
SUBTRACT COMMAND	The SUBTRACT command removes stations from the lines they are on.
SWAP LINE COMMAND	The SWAP LINE command swaps a line for another line.

COMS Command

The COMS command is used to control the kind of information to be written to COMS transaction trails, close the current COMS transaction trail and open a new one, and inquire about the current COMS transaction trail status.

DATABASE Command

The DATABASE command is used to control the kind of information to be written to transaction trails for a given data base, close the current transaction trail for a given data base and open a new one, and inquire about the current transaction trail status for a given data base.

JOBS Command

The JOBS command is used to display a list of programs that are running, together with the associated COMS window.

MONITOR Command

The MONITOR command is used to set the monitoring of COMS activities at all monitor stations. This command sets and resets various monitor attributes. A monitor message is sent to all monitor stations whenever any of the set attributes is encountered by COMS.

WINDOW Commands

COMS supports a number of commands that deal with windows. These commands are: WINDOWS, ENABLE WINDOW, DISABLE WINDOW, REPORT WINDOW, STATUS WINDOW, and JOBS IN WINDOW.

DISABLE WINDOW	This command: 1. Terminates all programs in the window. 2. Disables the window after the programs terminate. 3. Closes the window. 4. Places all stations that were on the window on dialog 1 of the MARC window.
ENABLE WINDOW	This command allows users to access dialogs of the specified window by using the ON command.
JOBS IN WINDOW	This command displays a list of the programs that are running in a specified window.
REPORT WINDOW	This command displays COMS network-related information about all stations that have open dialogs in the specified window.
STATUS WINDOW	This command displays the condition of the window specified (enabled or disabled), the user count for the window, and the number of messages from the window that are being held.
WINDOWS	The WINDOWS command is used to display the current COMS window environment for a particular station.

RECOVERY DIFFERENCES

This section describes the recovery differences between GEMCOS and COMS. Refer to the "Synchronized Recovery" section of the "A Series COMS Programmer's Guide" for a detailed explanation of synchronized recovery in COMS. We strongly suggest that the person responsible for converting application programs from GEMCOS to COMS read the "Synchronized Recovery" section of the "A Series COMS Programmer's Guide" and Section 7 of the "B 1000 GEMCOS User's Manual."

Recovery Specification

In GEMCOS, recovery is specified in the Program section. In COMS, recovery is specified on the PROGRAM menu by choosing the Data Base Name attribute. The acceptable values for this attribute are: <data base name> and NONE.

In both GEMCOS and COMS, NONE specifies that no recovery is to be performed on the given program. Specifiying a data base name on the PROGRAM screen in COMS is a request for synchronized recovery for that program. Synchronized recovery in GEMCOS and COMS is functionally similar but the methods of implementation are vastly different. COMS does not support the QUEUERESTORATION option available in GEMCOS.

DATABASE

GEMCOS DATABASE recovery is a subset of GEMCOS Synchronized recovery. Data base recovery in GEMCOS maps into COMS Synchronized recovery in the same manner as GEMOCS Synchronized recovery.

QUEUERESTORATION

Since QUEUERESTORATION is not available on COMS, you will have to maintain your own form of recovery for those programs that do not belong to a data base.

Synchronized Recovery

Synchronized recovery is a COMS function that resubmits transactions to the data base after a transaction-state abort, system crash, or rollback. First, DMSII recovery restores the data base to the last point in time when no programs were in transaction state. Next, COMS resubmits all completed transactions that occurred beyond the DMSII recovery point.

RECOVERY-RELATED CONVENTIONS

The following are conventions required in COMS to successfully perform a synchronized recovery on a data base.

1. All instructions must be grouped into transactions.

2. The program must enter transaction state, perform update activity in transaction state, then exit transaction state.

3. Each transaction must be two-phase. Two-phase transactions are those transactions that lock records but do not free any in the first phase and free records but do not lock any in the second phase.

4. All programs must be restartable, in other words, able to resume processing where COMS tells them after an interruption such as abnormal termination.

5. A restart data set must be created for every data base that is to be recoverable.

6. Routines must be written to handle:

   a. Setting up the input and output CD.

   b. Receiving messages.

   c. Normal data base close.

   d. Handling aborts and exceptions.

   e. Sending messages.

   f. Setting up the restart data set.

   g. Passing the message header at begin-transaction.

   h. Storing a restart locator in mid-transaction phase.

   i. Recovery considerations at end-transaction.

   j. Recovery consideration when sending a message.

   k. Using exception-condition statements.

<u>The Recovery Sequence (GEMCOS And COMS)</u>

Listed below are the logical steps for transactions and recovery in GEMCOS and COMS.

```
              GEMCOS                            COMS
              ------                            ----

     SET UP THE RESTART DATA SET       SET UP THE RESTART DATA SET
     PROCESSING:                       PROCESSING:
         BEGIN TRANSACTION                 BEGIN TRANSACTION
                                           MID-TRANSACTION
         END TRANSACTION                   END TRANSACTION
     CLOSE DATA BASE                   CLOSE DATA BASE
```

Though these logical steps are the same, the constructs that perform this logic are very different. The following sections describe how each of these logical steps differ in GEMCOS and COMS and give examples of the constructs to perform the recovery.

Creating the Restart Data Set

In GEMCOS, the layout for the Restart Data Set is:

```
     RESTARTAREA RESTART DATA SET (
       GEMOCS-LITERAL                   ALPHA(6);
       GEMCOS-PGM-NBR                   NUMBER(3);
       GEMCOS-MULTI-NBR                 NUMBER(3);
       GEMCOS-DATE-TIME                 NUMBER(12);
       GEMCOS-DATA                      NUMBER(9)
                                   )
       POPULATION = 100;

       RESTARTSET ORDERED SET OF RESTARTAREA
         KEY IS (
              GEMCOS-LITERAL,
              GEMCOS-DATE-TIME,
              GEMCOS-PGM-NBR,
              GEMCOS-MULTI-NBR
                );
```

In COMS, the layout for the Restart Data Set is:

```
RESTART-DS DATA SET
   (
    RDS-IS                         ALPHA(6)
                                     INITIALVALUE "ONLINE" COMS-ID;
    RDS-PROG                       REAL COMS-PROGRAM;
    RDS-LOCATOR                    REAL COMS-LOCATOR;
   );
```

Initialize the Restart Data Set

Initialization in GEMCOS and COMS opens the data base for recovery and opens the restart data set.

```
GEMCOS
------


<open the remote file>
<open the data base>
<read a record from the remote file>
MODIFY <restart data set> AT
   GEMCOS-LITERAL    = "GEMCOS"                    AND
   GEMCOS-DATE-TIME  = <GEMCOS-HEADER-date-time> AND
   GEMCOS-PGM-NBR    = <GEMCOS-HEADER-pgm-nbr>   AND
   GEMCOS-MULTI-NBR  = <GEMCOS-HEADER-multi-nbr>
ON EXCEPTION
   CREATE <restart data set name>
      MOVE "GEMCOS"                      TO GEMCOS-LITERAL
      MOVE <GEMCOS-HEADER-date-time>     TO GEMCOS-DATE-TIME
      MOVE <GEMCOS-HEADER-pgm-nbr>       TO GEMCOS-PGM-NBR
      MOVE <GEMCOS-HEADER-multi-nbr>     TO GEMCOS-MULTI-NBR
      MOVE 0                             TO GEMCOS-DATA.
```

(The field names that start with GEMCOS-HEADER are received as text on the first message read.)

```
COMS
----

MOVE ATTRIBUTE NAME OF ATTRIBUTE EXCEPTIONTASK
     OF MYSELF TO <work area name>.
CHANGE ATTRIBUTE TITLE OF "DCLILIBRARY" TO <work area name>.
ENABLE INPUT <input CD name> KEY "ONLINE".
OPEN UPDATE <data base name>
```

```
     ON EXCEPTION
          DISPLAY <DMSII exception error>
          CALL SYSTEM DMTERMINATE.
CREATE <restart data set name>.
MOVE "ONLINE" TO RDS-ID.
MOVE <COMS-in-program field name> to RDS-PROG.
MOVE <COMS-in-restart-locator field name> to RDS-LOCATOR.
```

(COMS-in-program field name is word 0 (zero) of the input CD and the COMS-in-restart-locator field name is word 10 of the input CD.)

Processing

Processing in both GEMCOS and COMS follows these steps:

1. Receive a message.
2. Make all preparations for the update.
3. Enter transaction state (begin transation).
4. Perform the update activity.
5. Send the result to the originator of the transaction.
6. End transaction state.
7. Return to step 1 to receive another message.

In COMS, the processing section can contain as many transations as necessary. Each transaction has a definite starting and ending point and performs only one operation per group of protected (locked) records.

Begin Transaction

Before the first transaction, in both COMS and GEMCOS, you must lock the data records. Once the records are locked, the transaction can be processed. Below is an example of GEMCOS and COMS begin transaction syntax.

```
GEMCOS
------

BEGIN-TRANSACTION NO-AUDIT <restart data set name>
    ON EXCEPTION <exception handling code>.
```

COMS

```
BEGIN-TRANSACTION <input CD name> USING <message area name>
   NO-AUDIT <restart data set name>
   ON EXCEPTION
       <exception handling code>.
```

Begin Transaction Abort Handling

In GEMCOS, if a program aborts, all the programs using the failed data base are sent a Type 20 message from GEMCOS. Each program then sends GEMCOS a Type 21 message and ignores all input messages until a Type 22 message is received. Also, if a program detects an abort on BEGIN-TRANSACTION, it sends a Type 21 message to GEMCOS. This sequence of messages is required by GEMCOS to be able to initiate recovery.

In COMS, when an abort occurs, the system automatically notifies COMS of the abort. You must include code in the begin transaction area of the program to stop current processing. This means the program should go back to receive another message rather than continue processing the current message. An indication of the abort could also be sent as part of the output message. For example,

```
...
ON EXCEPTION
     GO TO RECEIVE-NEXT-MESSAGE.
```

Mid-Transaction

COMS has a mid-transaction point in the recovery process that is not available in GEMCOS. Mid-transaction is the period between when the last record is locked and the first record is freed. Only the protected records (locked) can be accessed during this phase. It is also during this phase that COMS performs the transaction audit.

GEMCOS users do not have to explicitly perform mid-transaction processing since GEMCOS does not allow locked records to be freed during transaction state. For complete information about the mid-transaction phase, see the A Series COMS Programmer's Guide.

End Transaction

The end transaction frees the records that were locked for processing. Below is an example of the GEMCOS and COMS end transaction syntax.

```
GEMCOS
------

END-TRANSACTION AUDIT <restart data set name>
   ON EXCEPTION
       <exception handling code>.


COMS
----

END-TRANSACTION <output CD name> USING <output message area name>
   AUDIT <restart data set name>
   ON EXCEPTION
       <exception handling code>.
```

End-Transaction Abort Handling

If an end transaction abort occurs in COMS, follow the same steps that are used to handle begin transaction aborts. After Synchronized recovery is complete, COMS automatically resubmits the current transaction to the program. In GEMCOS, the program must initiate the Type 21/22 message sequence.

Close The Data Base

In GEMCOS, a Type 24 message is sent to the program instructing it to close the data base. When this message is received, the program closes the data base. If the data base close is successful, the program sends a Type 25 message to GEMCOS. GEMCOS then instructs the program to terminate. If the data base was not closed successfully, the program must initiate the Type 21/22 message abort detect "handshake" to recover. In other words, the program will send a message to GEMCOS saying that the data base was not closed successfully.

In COMS the receipt of a message with the input CD field COMS-IN-STATUS-KEY equal to 99 instructs the program to perform the end-of-job routine which closes the data base. We also recommend that the program explicitly store the restart record in the restart data set before closing the data base. This ensures that COMS and DMSII can synchronize their recovery operations if the data base has to be rolled back. The following is an example of COMS code that closes the data base and stores the restart data set.

```
BEGIN-TRANSACTION NO-AUDIT <restart data set name>.
RECREATE <restart data set name>.
STORE <restart data set name>.
END-TRANSACTION NO-AUDIT <restart data set name>.
CLOSE <data base name>.
STOP RUN.
```

Recovered Message Resubmittal

In GEMCOS, when a message is received, the common area header field RECOVERYSTATUS indicates whether the message is a normal or recovery message. A value of 0 (zero) indicates a normal message, any other value indicates a recovery message. In addition, RETRYCOUNT indicates how many times the message was resubmitted.

In COMS, whan a message is received, the input CD field COMS-IN-STATUS-KEY indicates the status of a message. A value of 0 (zero) indicates a normal message. The value 92 indicates a message resubmitted during synchronized recovery. The value 93 indicates that the transaction is being resubmitted because it caused the program to fault the last time it was submitted.

ARCHIVAL RECOVERY

In GEMCOS, archival recovery is a stand-alone event that must be run and then terminated and a new copy of the MCS started before normal processing can proceed.

In COMS, to perform the equivalent of a GEMCOS archival recovery, the following steps must be performed:

1. Disable the data base through COMS (DISABLE command).

2. Load the most recent backup of the data base, or if possible, roll back the current data base to an earlier (valid) data base.

3. Apply the audit trails for the data base up to the last quiet point before the problem occurred.

4. Make sure the current COMS transaction trail, and possibly previous transaction trails, are present on disk.

5. Enable the data base through COMS, (ENABLE command). This causes COMS to automatically roll the data base forward from where the DMSII recovery left off to the end of the transaction trail(s).

When the transaction trail or trails have been applied, the result should be recovery of the data base with no loss of systems-acknowledged transactions.

14 SORT

The SORT compiler is designed to meet all the A Series sorting requirements, as well as provide a tool to ease the progression from Small Systems. As a progression tool, the A Series SORT compiler accepts most Small Systems SORT syntax. It does not implement Small Systems SORT UTILITY. This section documents the differences between the A Series SORT compiler and the Small Systems SORT program.

SORT ONLY AS A COMPILER

The A Series SORT is implemented only as a compiler. You may do a "compile to library" and then execute the compiled program, or you may do a "compile and go." Unlike the restrictions associated with the Small Systems SORT, there are no restrictions associated with compiled sort programs.

STATEMENTS NOT NECESSARY IN THE A SERIES SORT

The following are Small Systems statements accepted by the A Series SORT compiler, but ignored. A warning is issued by the compiler when one of these statements is encountered. You may remove these statements or leave them without affecting the A Series SORT.

```
BIAS
TIME
TIMING
```

STATEMENTS NOT SUPPORTED IN THE A SERIES SORT

The following are Small System statements that cause a syntax error when encountered by the A Series SORT compiler. These statement must be removed.

```
COLLATE
DUPCHECK
INPLACE
OVERRIDE
SEQUENCE
TAGCOBOL
TAGRPG
ZIP
```

A SERIES REPLACEMENTS FOR SMALL SYSTEMS SORT STATEMENTS

The Small Systems TEACH command must be replaced by the A Series $ REFORMAT compiler control option.

The Small Systems NOPRINT statement must be changed to the A Series $ RESET LIST compiler control option.

EMBEDDED COMMENTS

The way Small Systems handle embedded comments is not supported by the A Series SORT compiler. On the A Series, comments must follow a "%" or ":" on an input record.

FILE STATEMENT

The Small System SORT program allows up to 16 Small Systems file input parts to be sorted. The A Series SORT compiler allows up to eight input files to be merged and from 1 to 99 input files to be sorted.

VARIABLE LENGTH RECORDS

The A Series SORT compiler recognizes the Small Systems syntax for variable length records, but issues a syntax error because the variable length record capability is not implemented.

FILE NAMES

On Small Systems, "*(USERCODE)/A" overrides the default pack specification and looks for the file on the systems's disk. There is no equivalent syntax on the A Series. The SORT compiler treats "*(USERCODE)/A" as though the "*" was not there.

Small Systems SORT program uses the default pack of the usercode named in the file title. The A Series SORT compiler uses the default pack (family substitution) of the usercode running the program.

For example,

> Usercode A has a default pack X (on Small Systems)
> or
> FAMILY DISK = X ONLY (on A Series).
>
> Usercode B has a default pack Y (on Small Systems)
> or
> FAMILY DISK = Y ONLY (on A Series).

A program running under usercode "A" opens file "(B)Q".

The Small Systems SORT would open the file on pack "Y". The A Series SORT opens the file on pack "X".

DATA TYPE DIFFERENCES

The types SA and RSA are not handled the same on the two systems. On the Small Systems SORT, all the bits are used in the comparison. On the A Series, only the digits are used. The A Series sees C1C2C3 as equal to F1F2F3, while the Small Systems do not.

There is no exact equivalent of SA or RSA on the A Series.

MEMORY STATEMENT

By default, the Small Systems SORT uses 20,000 bytes of memory (8000 bytes prior to the 10.0 release). By default, the A Series SORT uses enough memory to hold 1200 records. This is usually substantially more memory than was used on the Small Systems, but it is the amount recommended for a fast sort. See the SORT section in the "A Series System Software Utilities Reference Manual."

The MEMORY statement on Small Systems specifies memory in bytes. The MEMORY statement on the A Series specifies memory in words. One word equals six bytes. If a large amount of memory was specified on Small Systems, six times that much memory might be too much on the A Series. Refer to the SORT section in the "A Series System Software Utilities Reference Manual" to determine the appropriate amount of memory you really need. Since the default memory will probably be much greater, you should consider removing the MEMORY statement.

INCLUDE AND DELETE STATEMENTS

The IN option of the Small Systems INCLUDE and DELETE statements is not supported.

The Small Systems SORT specifies that expressions are evaluated strictly from left to right and that the first false expression causes the evaluation to end. The A Series SORT compiler (like the Small Systems SORT/UTILITY) specifies that there is a precedence order of NOT, AND, and OR. Some of the Small Systems SORT programs will need to be modified to add explicit parentheses to obtain the desired evaluation of the condition. For example, on Small Systems, the statement:

```
     INCLUDE 1 EQL "A" AND 2 EQL "B" OR 3 EQL "C".
```

does not include the record AXC. On A Series, the record is included.

With Small Systems SORT, if DELETE is followed by INCLUDE, and both of these statements select the same record, the record is retained. The A Series SORT compiler uses the rule, like the Small Systems SORT/UTILITY, that each successive INCLUDE or DELETE acts to further subset the stream of input records. For example, on Small Systems, the statements:

```
     DELETE 1 EQL "A"
     INCLUDE 1 EQL "A"
```

includes the record AXC. However, it is not included on the A Series.

TAGSORT STATEMENT

Small Systems SORT creates 4-byte (8-digit) index records when TAGSORT is specified. The A Series SORT compiler creates this type of record if a type of PACKED and a length of eight are specified. By default, however, a 1-word record is created.

The ADDROUT files supported by the RPG compiler on the A Series consist of 1-word records. This means no change to either the RPG program or the SORT is required for conversion of ADDROUT files from the Small Systems to the A Series. However, the ADDROUT file used on the A Series must be created by the A Series SORT.

15 REPORTER III

All REPORTER products run on both Small Systems and the A Series. However, there are some minor differences in the way the products run. Complete explanations of these differences are covered in these REPORTER manuals:

1. On-Line REPORTER User's Guide, form 1185220.
2. Vocabulary Language (VOCAL) User's Guide, form 1180428.
3. AUDIT-REPORTER Language User's Guide, form 1180486.
4. REPORTER II and REPORTER II (Advanced) User's Guide, form 1185121.
5. On-Line REPORTER III User's Manual, form 1177151.
6. REPORTER III Vocabulary Language (VOCAL) User's Manual, form 1177177.
7. REPORTER III Report Language User's Guide, form 1177185.

The same manuals are used for both Small Systems and the A Series.

Listed below are the differences for REPORTER III and in what REPORTER manual it is discussed.

If you are using REPORTER II, REPORTER II (Advanced), or AUDIT-REPORTER, we recommend that you upgrade to REPORTER III before progressing. For upgrading to REPORTER III, see the REPORTER III distribution letter.

VOCABULARY LANGUAGE (VOCAL) USER'S MANUAL

External File Name

On Small Systems, the external file name contains a maximum of three identifiers. Each identifier may contain up to 10 characters, generally without special characters.

On the A Series, the external file name is a series of identifiers, these may include a usercode, file directory identifiers, a file identifier, and a family name. Each identifier may contain up to 17 characters (including slashes).

DATA SET Statement

On Small Systems, the DATA SET statement is used to add a data set to the vocabulary, to give an alternate name to a disjoint data set, or to assign a TOTAL POPULATION to a data set. The data set name must be a valid Small Systems data set.

On the A Series, the DATA SET statement is used to give an alternate name to a disjoint data set or to assign a TOTAL POPULATION to a data set. The data set name must be a valid A Series data set in the DB-INVOKE listing.

EXCLUDE Statement

On Small Systems, the EXCLUDE statement is used to exclude elements in a disjoint data sets from the vocabulary.

On the A Series, this statement is used to delete data sets and associated elements from the vocabulary. The A Series also has two additional options, LINK ITEM and CONTROL ITEM.

SET Statement

On the A Series, the SET statement has the additional option DECIMAL-POINT IS COMMA. This option is used to force the building of the vocabulary files with the COBOL SPECIAL-NAMES.

DMSII Language Statements

On Small Systems, the data base statement informs RP3VOC that the specifications refer to a Small Systems data base. RP3VOC processes Small Systems statements by accessing the data-set COBOL library files created by DASDL. The format for the library file name is <data base

name>/<disjoint-data-set-name>. The data sets must be individually specified to the RP3VOC. They can be specified in any order.

On the A Series, the data base statement identifies an A Series data base. RP3VOC processes the statement for the data base by referencing a directory file. A utility program, RP3VDM, creates this directory. RP3VDM must be run once for each physical or logical data base that RP3VOC uses. RP3VOC automatically executes RP3VDM to create the required directory. You also have the option of running RP3VDM manually.

USING Clause

The USING clause is not available on the A Series since FORTE and FORTE2 files are not used.

Storage Media And File Attributes

The following tables show the storage media and file attributes for Small Systems and the A Series.

Media	File Attributes	
	Small Systems	A Series
Cards	80-column card	80-column card
Disk	Card image disk file	Characters per record and blocking is assigned DEFAULT
Library	COBOL library file	Characters per record and blocking is assigned DEFAULT
Tape	Tape file	80-character/record tape, blocked 9
Interchange disk pack	Interchange card-image file on disk pack	
Disk pack		Characters per record and blocking is assigned DEFAULT

Files Required For Execution

The files required for execution have changed on the A Series. The names of the files are given in the REPORTER III documentation.

Unrecognized COBOL Constructs

The RP3VOC portion of the REPORTER III Systems analyzes user-supplied ANSI-74 COBOL source programs. However, there are difference in Small Systems and the A Series constructs that are accepted by the RP3VOC. These difference are explained in Appendix B (A Series) and Appendix D (Small Systems) of the "REPORTER III Vocabulary Language User's Manual."

Defaults And Limits

The defaults and limitations of the A Series are different than those for Small Systems. The REPORTER III report language limits and defaults are listed in Appendix B of the "Reporter III Report Language User's Manual."

COBOL74 Code

Additional COBOL74 code added to the vocabulary, either with SOURCE FILE FOLLOWS or as part of an input procedure, has to conform to the COBOL74 syntax of the A Series.

REPORTER III REPORT LANGUAGE USER'S MANUAL

External File Name

The difference between Small Systems and the A Series external file names is described previously. It also affects the SAVE statement, the SAVE LISTING statement, and the VOCABULARY statement. Its effects on these statements is given in the REPORTER III documentation.

REPORTER III

Maximum Characters For a PIC Clause

The maximum length of a COBOL PIC specification on Small Systems is 18, for the A Series, it is 22.

EXTRACT FILE AREASIZE Statement

On Small Systems, the maximum number of records per area is 16,777,216. On the A Series, the maximum number of records per area is 1,048,575.

EXTRACT FILE Statement

The default number of areas is 20.

On Small Systems, the maximum number of areas is 105. On the A Series, the maximum number of areas is 1000.

Sample Statement Parameters Limit

Parameters	Limits
Sample size	1 to 99,999,999
Sample size percent	1 to 100
Strata population	Sample size to 99,999,999
Seed	0 to 99,999,999
Sum of all strata population	A Series: 1,080,000,000 Small Systems: 108,000,000

Process Options SET Statement

Statements	Default	Limits
INTEGER-SIZE	12	Small Systems: 1 to 18, minus FRACTION-SIZE A Series: 0 to 22, minus FRACTION-SIZE

FRACTION-SIZE	5	Small Systems: 0 to 18, minus INTEGER-SIZE A Series: 0 to 22, minus INTEGER-SIZE
STRING-SIZE	30	PAGE-WIDTH
NULL-NUMERIC	0	---
NULL-STRING	SPACES	---
NULL-BOOLEAN	FALSE	---
MODE	Default is ON-LINE if Report Language Analysis Program (RP3REP) is run via On-Line REPORTER III; otherwise, default is BATCH.	

ON-LINE REPORTER III USER'S MANUAL

The differences between Small Systems and the A Series operation is discussed in Appendix C of the "On-Line Reporter User's Manual."

EXECUTION OF REPORTER III

There are also differences in the way REPORTER III is executed. Whether you are executing Vocabulary or the Report Language of REPORTER III, it will need to be modified to conform to the method in which A Series jobs are executed.

Cards Or Pseudo Reader

On Small Systems, REPORTER III is executed using the syntax:

```
?EX RP3REP
?DATA RP3CRD
  <language statements in columns 8 - 72)
?END
```

REPORTER III

On the A Series, use the following syntax to execute REPORTER III:

```
?RUN RP3REP
?DATA RP3CRD
  <language statements in columns 8 - 72)
?END
```

ODT

To execute REPORTER III on the Small Systems, the syntax is:

```
EX RP3REP;FILE RP3CRD NAME
<file name> DISK DEF
```

On the A Series, REPORTER III is executed using this syntax:

```
RUN RP3REP;FILE RP3CRD(TITLE=
<filename>, KIND=DISK)
```

<u>16</u> <u>ODESY</u>

The Small Systems On-Line Data Entry SYstem (ODESY) user will find a relatively easy path for progression to the A Series. A Series ODESY is available and provides functional and visual equivalence to Small Systems ODESY version 2.2.

A program for progressing Small Systems ODESY 2.2 format dump files for loading by Format Maintenance on the A Series is provided. Users of Small Systems ODESY 2.1 should first upgrade to Small Systems ODESY 2.2, then to the A Series version.

In addition to progressing format dump files, you will be required to make changes to your programs, provided you are currently using this facility. If you have SDL/UPL user programs you must rewrite the programs in COBOL74 or RPG, since the SDL/UPL language does not exist on A Series. If you have COBOL74 or RPG user programs you should progress the programs as you would any application (with the aid of Burroughs translators), then make two minor changes to the message header of the program. These changes are described in the A Series ODESY Manual.

17 ISAM FILES

Small Systems have three different types of ISAM files: B-Indexed files TAG files, and ISAM files. A Series has two types of ISAM files, ISAM and KEYEDIO. Since the A Series ISAM file access method is somewhat primitive and available only for COBOL(68) and PL/1, we recommend KEYEDIO files since they are available for COBOL74 and RPG. This section describes how to transfer Small Systems ISAM files to A Series KEYEDIO files.

DATA TRANSFER

All Small Systems ISAM files are easily converted to A Series files. However, because of format difference, these data files cannot be transferred from Small Systems to A Series using only B1000COPY or B6000COPY.

B-Indexed Files

B-Indexed files are available only in RPG. Use SYSTEM/COPY and B1000COPY to transfer the B-Indexed file, then write an A Series program to read the data file and create a KEYEDIO file. There is an A Series COBOL74 sample program called LOADISAMS available on the A Series 3.6 Release BTA360 Migration Aids tape to help you write the program.

TAG Files

TAG files are available in RPG and COBOL(68). Use SYSTEM/COPY and B1000COPY to transfer the data portion of the TAG file, then write an A Series program to read the data file and create a KEYEDIO file. There is an A Series COBOL74 sample program called LOADISAMS available on the A Series 3.6 Release BTA360 Migration Aids tape to help you write the program.

ISAM Files

ISAM files are available in RPG and COBOL74. To transfer the Small Systems files, write a Small Systems program that reads the ISAM file and creates a sequential data file. If the data file is on disk, use SYSTEM/COPY and B1000COPY to transfer the file to the A Series.

Then write a A Series program to read the tape or disk sequential file and create a KEYEDIO file. There are COBOL74 sample programs on the Small Systems BTA350 Conversion Tape and A Series 3.6 Release BTA360 Migration Aids tape. The sample programs are called DUMPISAMS and LOADISAMS, respectively. These sample programs will assist you when writing your unload and load programs.

A SERIES KEYEDIO FILES

KEYEDIO is available in COBOL74 and RPG. It is a multi-user (multiple inquiry and multiple update), multi-keyed method of file access. The data and keys on A Series are stored in one physical file.

FILEORGANIZATION

The File Attribute called FILEORGANIZATION has two values that pertain to KEYEDIO:

1. INDEXED.
2. INDEXEDNOTRESTRICTED.

If a value for FILEORGANIZATION is not specified, the default values are assigned. The default values are:

1. INDEXED for COBOL74.
2. INDEXEDNOTRESTRICTED for RPG.

When you want a value other than the default value, INDEXEDNOTRESTRICTED for COBOL74 and INDEXED for RPG, the line indicated by the arrow in the following example should be inserted in the source code.

```
    COBOL74
    -------


    FD   ISAM-FILE
         RECORD CONTAINS                180 CHARACTERS
         BLOCK CONTAINS                 10 RECORDS
   ----> VALUE OF FILEORGANIZATION IS INDEXEDNOTRESTRICTED
         AREASIZE IS                    5000
         AREA IS                        25.
```

RPG

```
     04110F*************
     04120F*     EXAMPLE:    ONE ATTRIBUTE FOR THE FILE
     01430FFILE1    IP     80  80             DISK
---> 01440A  FILEORGANIZATION "INDEXED"
     01450F*************
```

To change the FILEORGANIZATION of an existing file, the file must be recreated.

INDEXED

If the file was created with the organization of INDEXED, any program that accesses this file must describe the file as INDEXED, and if keys are declared, they must match the keys exactly as they were defined in the existing file. This organization conforms to the ANSI-74 standards.

An indexed file may not be sorted and may not be listed with the utility SYSTEM/DUMPALL.

INDEXEDNOTRESTRICTED

We recommend that your file have a FILEORGANIZATION of INDEXEDNOTRESTRICTED. If a file is created with a FILEORGANIZATION of

INDEXEDNOTRESTRICTED, accessing this file may be defined with any of the following organizations:

1. SEQUENTIAL.

2. INDEXED.

3. INDEXEDNOTRESTRICTED.

Such a program may declare none of the keys, all of the keys, or any number of the keys. Remember that any key declared in the program accessing the KEYEDIO file must match the key in the declaration of the file when the file was created.

If the file is accessed sequentially, the program will get one copy of each record. It is not guaranteed that the program will get the records in any given order. Also, the speed in which you receive these records will not be as fast as if the program were getting records from a purely sequential file. This file may also be accessed via the relative record number (actual key construct). If either of these access methods are being used, the program may update and/or add records to the ISAM file without corrupting any of the keys. They will always be properly maintained.

Because this file may be accessed sequentially, programs such as SORT and DUMPALL may open a file created with the FILEORGANIZATION of INDEXEDNOTRESTRICTED.

Recovery Of A KEYEDIO File

When creating a KEYEDIO file (any KEYEDIO file opened output), if the system should fail or the program DS, that file and any records written to the file will be lost. To prevent the file from being lost, a permanent disk file can be created by opening the file OUTPUT, closing the file SAVE, and then reopening the file I-O before writing to it. Thus, if a program is DSed or the system should fail, the file and the records written to the file will be on disk. However, performance will be significantly slower.

Once the file has been created, any additions, changes, or deletions are always recovered up to the last record. This happens because once a WRITE is completed, the record is on disk. The record is not stored in the buffer for any length of time, and no other statement will be executed until the WRITE statement is completed.

If recovery is needed, the system will automatically initiate it. If the program was DSed and recovery is needed, it will begin as the program goes to EOJ. If the program is executing and the system fails, recovery will take place the next time the file is opened.

Example

```
     WRITE ISAM-RECORD.
     MOVE 1 TO ISAM-COUNTER.
```

When the MOVE statement is executed, the ISAM record is on disk.

If the system failed in the middle of the WRITE statement, the next time the file is opened, KEYEDIO automatically restores the integrity of the file. This means there will never be a case when a key points to a non-existent record or a record without a key.

Header Information

The Header or Control Information is located in the first record(s) of the file.

Data and Indices

The Data Blocks and Indexed Blocks are mixed throughout the remainder of the file.

Each index is organized as a B-TREE in a manner similar to Small Systems DMSII and Small Systems ISAM. For more information, refer to the "KEYEDIO" section in the "A Series System Software Support Reference Manual."

Performance Considerations

Changing the block size of the KEYEDIO file can cause considerable changes in performance. Remember that the block size can be changed only before the file is created.

Each index table is the same size as the user specified block size. Therefore, if you increase the block size you increase the table size. Increasing the table size means fewer tables and fewer disk I-Os. This results in improved performance. When calculating the block size, remember that these larger block sizes only give you an advantage when you are:

1. Accessing a file randomly.

2. Deleting records from or randomly adding records to a file.

3. Updating a file and a given key is changed.

The block size has little effect on a file that is accessed sequentially.

Block sizes that are too large can degrade performance by using too much memory. By default, KEYEDIO locks at least 10 blocks in SAVE memory for each KEYEDIO file open. The number of buffers used by the KEYEDIO library in processing an indexed file may be controlled by the user. A program may indicate the number of buffers KEYEDIO is to use by setting the value of the BUFFERS attribute of the indexed file.

Normally the best performance occurs when there are two levels of tables; a root table which points to a fine table which points directly to data.

Example

If each root table holds 10 records, then the root table can point to 10 fine tables, and each fine table can point to 10 data records. The result is a file which holds 100 records.

ISAM Files

```
                       FINE TABLES           DATA RECORDS

                       -----                      --
                   /->|  1  |-------------------->|10|
                  /    -----                      --
                 /             -----                 --
ROOT TABLE      / /---------->|  2  |--------------->|10|
               / /             -----                 --
 -----------  / /
|           |/ /       -----                      --
|-----1-----| / /---->|  3  |-------------------->|10|
|           |/ /       -----                      --
|---- 2 ----| /
|           |/                 -----                 --
|---- 3 ----| /-------------->|  4  |--------------->|10|
|           |/                 -----                 --
|---- 4 ----|
|           |          -----                      --
|---- 5 ----|-------->|  5  |-------------------->|10|
|           |          -----                      --
|---- 6 ----|
|           | \                 -----                 --
|---- 7 ----|  \-------------->|  6  |--------------->|10|
|           |\                  -----                 --
|---- 8 ----|  \
|           |\   \        -----                       --
|---- 9 ----|  \   \---->|  7  |--------------------->|10|
|           |\   \        -----                       --
|--- 10 ----|  \   \
|           |\   \   \
|           |  \   \   \             -----                --
 -----------|    \   \   \--------->|  8  |-------------->|10|
                   \   \             -----                --
                     \   \
                       \   \  -----                      --
                         \   |  9  |-------------------->|10|
                           \  -----                      --
                             \
                               \ ----                 --
                                | 10 |--------------->|10|
                                 ----                 --
```

If more than 100 records are entered into this file, or if the records are entered in an unorderly manner which causes some of the tables to be filled, then a third level of tables will be automatically allocated.

In this case, the root table points to a coarse table, the coarse table points to the fine table, and the fine table points to the data records.

Example

```
                      COARSE          FINE       DATA
                      TABLES         TABLES     RECORDS

                      -----          ----        ---
                   /->|  1  |------->|1-10|------>|100|
                  /   -----          ----        ---
                 /       -----           ----          ---
ROOT TABLE      / /----->|  2  |--------->|1-10|------->|100|
               / /       -----           ----          ---
------------  / /
|           |/ /      -----          ----        ---
|-----1-----| / /---->|  3  |------->|1-10|------>|100|
|           |/ /      -----          ----        ---
|---- 2 ----| /
|           |/           -----           ----          ---
|---- 3 ----| /--------->|  4  |--------->|1-10|------->|100|
|           |/           -----           ----          ---
|---- 4 ----|
|           |         -----          ----        ---
|---- 5 ----|-------->|  5  |------->|1-10|------>|100|
|           |         -----          ----        ---
|---- 6 ----|
|           |\            -----           ----          ---
|---- 7 ----|  \--------->|  6  |--------->|1-10|------->|100|
|           |\            -----           ----          ---
|---- 8 ----|  \
|           |\   \        -----         ----         ---
|---- 9 ----|  \   \----->|  7  |------>|1-10|------->|100|
|           |\   \        -----         ----         ---
|--- 10 ----|  \   \
|           |\   \   \        -----           ----          ---
 -----------|  \   \   \----->|  8  |--------->|1-10|------->|100|
                 \   \            -----           ----          ---
                   \   \
                     \   \      -----        ----         ---
                       \   \|  9  |----->|1-10|----->|100|
                         \      -----        ----         ---
                           \
                            ------       ----         ---
                            |  10  |------>|1-10|------>|100|
                            ------       ----         ---
```

As illustrated, each coarse table points to 10 fine tables and each fine table points to 10 data records. The file can now hold 1000 records. The extra table will cause additional disk IOs, thus poorer performance.

How To Calculate BLOCKSIZE

The following formula will help you calculate optimum BLOCKSIZE:

1. Calculate the number of records the file will contain over its lifetime.

2. Compute the square root of the number of records. Then, multiply this value by an adjustment factor to allow for partially filled tables. The result of this computation is the desired number of keys per block.

   The value of the adjustment factor is determined by the way the file is to be created and updated. If the file is created sequentially with entries for all keys in ascending order and few records will be added later, you can use a small adjustment factor of 1.1. If the file is created sequentially and more records will be added later, use an adjustment factor of 1.3 (or greater, if many records will be added). If file is created with entries for some of the keys occurring in random order, use an adjustment factor of 2.0.

3. Compute the size of the largest key entry by performing the following steps:

   a. Find the size of the largest key in the record.

   b. If the key size is not already a multiple of six characters, round this size up to the next multiple of six characters.

   c. Add six characters to allow space for the key entry's pointer to the data record.

4. Compute the desired block size by multiplying the desired number of keys per block (from step 2) by the size of the largest key entry (from step 3).

5. Round this desired block size up to the next multiple of the record size if it is not already a multiple of the record size. This last step ensures that the block size chosen is suitable for storing the data records as well as the keys.

The block size calculated as a result of this procedure provides two level access. However, its impact on the system must be determined prior to deciding that this is the correct block size to use. The effect of the block size on memory usage must be considered. The buffers used by KEYEDIO occupy SAVE memory. The amount of SAVE memory that will be used for a given indexed file can be approximated by multiplying the actual block size by the number of buffers to be used for the file. By default, KEYEDIO keeps eight blocks in memory for each physical file opened and two additional blocks for each user. However, this value may be changed using the BUFFERS file attribute.

If the SAVE memory requirements for a particular block size are too great, a new block size that provides three or four level access should be calculated. This can be done by using the algorithm given above. However, at step 2, compute the cube root or fourth root of the number instead of the square root.

We suggest a block size no larger than 5400 bytes.

Example 1

Assume you have a file with 10,000 records where the largest key is 10 bytes, and a record size of 120.

1. 10,000 records.
2. The square root of 10,000 is 100.

   100 * 1.3 = 130
3. a. 10 bytes.

   b. 12 bytes.

   c. 12 + 6 = 18 bytes.
4. 130 * 18 = 2340
5. BLOCKSIZE = 2400

Example 2

Assume you have a file with 100,000 records, a record size of 120, and two keys: a 30-byte key and a 6-byte key. For the best performance, when memory is not a concern, calculate block size using the 30-byte key. Also, assume the file was created with the entries for some of the keys occurring in random order.

1. 100,000 records.

2. The cube root of 100,000 is 47.

   47 * 2.0 = 94

3. a. 30 bytes.

   b. 30 is already a multiple of 6.

   c. 30 + 6 = 36 bytes.

4. 94 * 36 = 3384.

5. BLOCKSIZE is 3480.

If the 30-byte key is rarely accessed other than sequentially, and the 6-byte key is consistently accessed randomly, calculate block size using the 6-byte key. This is because the block size calculated using the 30-byte key uses more memory than the block size calculated using the 6-byte key. If the program accessing the file is doing a sequential access, there is no a performance advantage in comparison to the amount of memory being used. Thus, the larger block sizes are only give an advantage in performance when using random access, deleting or randomly adding records to your file, or updating a file and a given key is changed.

For more information, refer to the "KEYEDIO - Block Size Calculations for Indexed Files" in the "A Series System Software Support Reference Manual."

PROGRAM CONVERSION

COBOL74 Programs

There are no changes required when converting COBOL74 programs, however, we recommend adding the following line to the value clause in your FD.

```
VALUE OF FILEORGANIZATION IS INDEXEDNOTRESTRICTED
```

COBOL(68) Programs

Use the CTA filter to convert your COBOL(68) programs.

RPG Programs

The following describes some of the necessary changes to convert RPG files.

1. For ISAM files there are no changes.
2. For TAG files, the KEYEDIO file should be created with a key for each tag used on the file. That way, all the tags are available all the time and they will not have to be created.
3. For B-Indexed files, since a B-Indexed file is always ordered based on the key, some programs may access the file sequentially knowing that the records are in order. These programs will require a change to describe and access the file as INDEXED.

ADDITIONAL FEATURES

KEYEDIO has many features available that Small Systems B-Indexed style and TAG files do not. These features are:

1. KEYEDIO is multi-user (multiple inquiry and multiple update).
2. KEYEDIO is multi-keyed.

ISAM Files

3. KEYEDIO supports the DELETE verb.

4. KEYEDIO has automatic full recovery to the last record.

5. KEYEDIO files may be accessed either:

   a. SERIALLY.

   b. SEQUENTIALLY by KEY.

   c. RANDOM by KEY.

   d. RANDOM by RELATIVE RECORD NUMBER.

18 QUEUE/PORT FILES

The A Series equivalent to queue files is port files. This section compares queue files and port files, explains the subfile matching process for port files, details the COBOL74 interface to port files, and discusses port file attributes.

GENERAL INFORMATION

Queue files are used on Small Systems to communicate between two or more programs.

A port file provides communication paths between two programs. All records written from one program go into one path, while all reads get records from the other path. On a Burroughs Network Architecture (BNA) network, port files are used to communicate between programs on different hosts.

The following queue file example illustrates how a queue file is used to communicate between two programs. Program 1 writes records into the queue file, which are then read by Program 2.

```
Queue File Example
------------------

                                  Queue File
+--------------+           +------------+          +--------------+
|              | Writes    |  |  |  |  |  Reads    |              |
|  Program 1   |---------->|  |  |  |  |--------->|   Program 2  |
|              |           |  |  |  |  |          |              |
+--------------+           +------------+          +--------------+


Equivalent Port File Example
----------------------------

                                  Port File
                          +-------------+
+--------------+          |             |          +--------------+
|              |          |             |          |              |
|  Program 1   | Writes   |             | Reads    |  Program 2   |
|              |--------->|             |--------->|              |
|              |          |             |          |              |
+--------------+          |             |          +--------------+
                          +-------------+
```

The port file provides 2-way communication between the programs, however, the converted programs may not need both paths.

A program on the Small Systems can read and write records to a queue file as soon as the program opens the queue file. A program opening a port file on the A Series cannot read or write to the file until a second program opens another port file that is matched to the first port file.

When a program closes a queue file, other programs linked to that file can continue to use it. The program can reopen the queue file without the other programs making any changes. When a program closes a port file, the other program linked to that port file can read the remaining messages in the port file or close it. In order for the two programs to re-use the port file, the remaining program must close the port file and both programs must reopen it.

On Small Systems, queue files can be used for job spawning. Messages about the program are returned via the queue specified in the ZIP statement. Port files do not have this feature. Task capabilities are available in COBOL74 that can be used to simulate Small Systems job spawning. Also, WFL jobs can be started by programs.

The following diagrams and text illustrate some of the possible configurations of programs using queue files and the A Series equivalent using port files.

The simplest configuration is when a queue file is used for tanking messages as shown below.

Queue/Port Files

Message Tanking Queue File Example

```
            +---------------+
            |     Reads     |
            |               |
            V             +---------+
+-------------+         Q |---------| F
|             |         u |---------| i
|  Program 1  |         e |   :  :  | l
|             |         u |---------| e
+-------------+         e |---------|
            |             +---------+
            |               A
            |    Writes     |
            +---------------+
```

Equivalent Port File Example

```
                           Port File
                        +------------+
+-------------+ Writes  |            |  Reads +--------------+
|             |-------->|            |-------->|              |
|  Program 1  |         |            |         |  Program 2   |
|             | Reads   |            |  Writes |              |
|             |<--------|            |<--------|              |
+-------------+         |            |         +--------------+
                        +------------+
```

The only function of Program 2 is to read messages from the port file and immediately write them back.

So far, this section has explained port files that are used for communication between only two programs. A port file actually consists of one or more subfiles (also referred to as subports). Each subfile provides communication between two programs (via the two 1-way paths). The previous examples of a port file actually consisted of a port file with one subfile.

The following diagrams omit the "Reads" and "Writes" headings for clarity. Any line with an arrow pointing into a queue or subfile box is a write path and any line with an arrow pointing away from a queue or subfile box is a read path. Also, the examples will show communication in only one direction for clarity, but a subfile allows communication in both directions.

A more complex, but more common, configuration is where many programs write to a queue file and one program reads from that queue file.

```
     Many to One Queue File Example
     ------------------------------

     +-------------+
     |             |
     |  Program 1  |-----+
     |             |     |
     +-------------+     |
                         |     Queue File
     +-------------+     |    +-----------+          +-------------+
     |             |     +--->|  |  |  |  |          |             |
     |  Program 2  |---------->|  |  |  |  |--------->|  Program 4  |
     |             |     +--->|  |  |  |  |          |             |
     +-------------+     |    +-----------+          +-------------+
                         |
     +-------------+     |
     |             |     |
     |  Program 3  |-----+
     |             |
     +-------------+
```

Equivalent Port File Example

```
                           Port File
                       +---------------+
+-------------+        |   Subfile 1   |
|             |        | +-----------+ |
|  Program 1  |--------->|  |  |  |  |----+
|             |        | +-----------+ |  |
+-------------+        |               |  |
                       |               |  |
+-------------+        |   Subfile 2   |  |    +-------------+
|             |        | +-----------+ |  +--->|             |
|  Program 2  |--------->|  |  |  |  |--------->|  Program 4  |
|             |        | +-----------+ |  +--->|             |
+-------------+        |               |  |    +-------------+
                       |               |  |
+-------------+        |   Subfile 3   |  |
|             |        | +-----------+ |  |
|  Program 3  |--------->|  |  |  |  |----+
|             |        | +-----------+ |
+-------------+        |               |
                       +---------------+
```

Programs 1, 2, and 3 each declare one port file with one subfile. Program 4 declares one port file with three subfiles. Program 4 can open all subfiles at once or one at a time. Program 4 can also perform a general read that will duplicate a read on a Small Systems queue file. For more information, refer to OPEN and READ in "COBOL74 Interface," later in this section.

A program can be linked to an existing queue file. The programs already linked to the queue file can communicate with the new program without performing any extra processing. However, to add a program to a port file, both programs must open a subfile before any communication can occur. If the number of programs communicating through the port file changes while the program is running, the program's logic must be modified to open or close subfiles as other programs open or close their subfiles.

In the previous queue file example, the number of writing programs can change without the reading program performing extra processing. If the number of writing programs changes in the port file example, the reading program must open or close its subfiles accordingly.

Another configuration occurs when one program writes to a queue file and several programs read from that queue file.

One to Many Queue File Example

```
                                                    +-------------+
                                                    |             |
                                               +--->|  Program 2  |
                                               |    |             |
                                               |    +-------------+
                               Queue File      |
     +-------------+          +-----------+    |    +-------------+
     |             |          |  |  |  |  |----+    |             |
     |  Program 1  |--------->|  |  |  |  |-------->|  Program 3  |
     |             |          |  |  |  |  |----+    |             |
     +-------------+          +-----------+    |    +-------------+
                                               |
                                               |    +-------------+
                                               |    |             |
                                               +--->|  Program 4  |
                                                    |             |
                                                    +-------------+
```

Equivalent Port File Example

```
                           Port File
                       +----------------+
                       |   Subfile 1    |      +--------------+
                       | +-----------+  |      |              |
                   +---->|  |  |  |  |-------->|  Program 2   |
                   |   | +-----------+  |      |              |
                   |   |                |      +--------------+
                   |   |                |
+-------------+    |   |   Subfile 2    |      +--------------+
|             |----+   | +-----------+  |      |              |
|  Program 1  |---------->|  |  |  |  |-------->|  Program 3   |
|             |----+   | +-----------+  |      |              |
+-------------+    |   |                |      +--------------+
                   |   |                |
                   |   |   Subfile 3    |      +--------------+
                   |   | +-----------+  |      |              |
                   +---->|  |  |  |  |-------->|  Program 4   |
                       | +-----------+  |      |              |
                       |                |      +--------------+
                       +----------------+
```

Program 1 declares one port file with three subfiles. Programs 2, 3, and 4 each declare one port file with one subfile. The major difference between queue files and port files in this example is that program 1 will now have to choose which subfile to write records into. This can be done by choosing them in order (first 1, then 2, then 3, and back to 1). Another way is to have programs 2, 3, and 4 write a message back to program 1 when they are ready for another message.

The following example illustrates a queue file with several programs writing records into it and several programs reading records from it.

Many to Many Queue File Example

```
+-------------+                                +-------------+
|             |                                |             |
|  Program 1  |------+                    +--->|  Program 4  |
|             |      |                    |    |             |
+-------------+      |                    |    +-------------+
                     |    Queue File      |
+-------------+      |   +-----------+    |    +-------------+
|             |      +-->|  |  |  |  |----+    |             |
|  Program 2  |---------->|  |  |  |  |--------->|  Program 5  |
|             |      +-->|  |  |  |  |----+    |             |
+-------------+      |   +-----------+    |    +-------------+
                     |                    |
+-------------+      |                    |    +-------------+
|             |      |                    |    |             |
|  Program 3  |------+                    +--->|  Program 6  |
|             |                                |             |
--------------+                                +-------------+
```

Equivalent Port File Example

```
                          Port File 1
                       +---------------+
                       |   Subfile 1   |
                       | +-----------+ |
                   +---->|  |  |  |  |-----+
+-------------+    |   | +-----------+ |   *      +-------------+
|             |----+   |   Subfile 2   |   +----->|             |
|             |        | +-----------+ |          |             |
|  Program 1  |---------->|  |  |  |  |-------+ +->|  Program 4  |
|             |        | +-----------+ |     * .  |             |
|             |----+   |   Subfile 3   | +------->|             |
+-------------+    |   | +-----------+ | :   * .  +-------------+
                   +---->|  |  |  |  |-----+ * .
                       | +-----------+ | : * * .
                       +---------------+ : * * .
                                         : * * .
                          Port File 2    : * * .
                       +---------------+ : * * .
                       |   Subfile 1   | : * * .
                       | +-----------+ | : * * .
                   +---->|  |  |  |  |---------+
+-------------+    |   | +-----------+ | : * *    +-------------|
|             |----+   |   Subfile 2   | : * +--->|             |
|             |        | +-----------+ | : *      |             |
|  Program 2  |---------->|  |  |  |  |----------->|  Program 5  |
|             |        | +-----------+ | : *      |             |
|             |----+   |   Subfile 3   | : * +--->|             |
+-------------+    |   | +-----------+ | : * :    +-------------+
                   +---->|  |  |  |  |---------+
                       | +-----------+ | : * : .
                       +---------------+ : * : .
                                         : * : .
                          Port File 3    : * : .
                       +---------------+ : * : .
                       |   Subfile 1   | : * : .
                       | +-----------+ | : * : .
                   +---->|  |  |  |  |---+ * : .
+-------------+    |   | +-----------+ |   * : .  +-------------+
|             |----+   |   Subfile 2   |   +----->|             |
|             |        | +-----------+ |     : .  |             |
|  Program 3  |---------->|  |  |  |  |-------+ +->|  Program 6  |
|             |        | +-----------+ |          |             |
|             |----+   |   Subfile 3   |   +----->|             |
+-------------+    |   | +-----------+ |   :      +-------------+
                   +---->|  |  |  |  |-----+
                       | +-----------+ |
                       +---------------+
```

Each of the six programs declare a port file with three subfiles. The writing programs have to decide which subfile to write, as explained in the previous example titled, "One to Many."

In general, the writing programs declare a port file with the number of subfiles equal to the number of reading programs. The reading programs declare a port file with the number of subfiles equal to the number of writing programs.

QUEUE FILE FAMILIES

Small Systems also allow a queue file to be made up of one or more subqueues. This is called a queue file family. Each subqueue acts like a queue.

A queue file family can complicate the progression to port files. If each subqueue is used only to communicate between two programs (one-to-one) then it can be changed directly into one port file with the same number of subfiles as subqueues. If any subqueue in the queue file family is used in a many-to-one, one-to-many, or many-to-many situation, each subqueue has to be changed into a separate port file with the appropriate number of subfiles. As an alternative method, one port file can be used, however, the program has to know which subfiles are grouped together to correspond to the old subqueues.

SUBFILE MATCHING

Small Systems queue files can have one or two 10-character file names. If the queue file is a part of a queue file family, you can specify only one 10-character file name. The system will generate the second 10-character name. When a queue file is opened, the system checks to see if a queue file with that name already exists. If the queue file does exist, the program is linked to that queue. If the queue file does not exist, a new queue file is created.

A Series Port Files use several different file attributes to determine when to match (link) two subfiles. Some apply to the port file as a whole, others apply to each subfile separately. When a program offers a subfile for matching (via an OPEN statement), if another subfile is already offered with matching attributes, the system makes the connection and sets certain file attributes to tell the programs that the subfile is now open. If a matching subfile is not found, the action taken depends upon the open option used in the OPEN statement. See the "COBOL74 Interface" later in this section, for more information.

Throughout the following discussion, the offered subfile is the subfile for which the system is trying to find a match. The complementary subfile is the subfile that is being checked to see if it matches the offered subfile. (The complementary subfile would have been previously offered by another program.)

When an attribute applies to a file as a whole, the attribute can only be set or interrogated on the file level. If the attribute applies to each subfile, it can be set or accessed for each file via a subfile index. For more information, see "Queue File Attributes Compared To Port File Attributes," later in this section.

The following attributes are used when the A Series match subfiles:

MYNAME	MYNAME is a string of one to 100 characters. The MYNAME attribute applies to the file. For two subfiles to match, MYNAME of the offered subfile must equal YOURNAME of the complementary subfile. If MYNAME is null (a value of ".") then it will only match a null value for YOURNAME.
YOURNAME	YOURNAME is a string of one to 100 characters. YOURNAME applies to each subfile. YOURNAME of the offered subfile must equal the value of MYNAME for the complementary subfile for the two subfiles to match. If YOURNAME is null (".") then it will match any value for MYNAME (including ".").
TITLE	TITLE can be up to 17 characters long and cannot include a slash (/). The TITLE applies to the port file. The TITLE of the port file for the offered subfile must match the TITLE of the port file for the complementary subfile. TITLE defaults to the internal file name of the file.
SECURITYTYPE	This attribute applies to the port file. A Series applies security checking each time it tries to match two subfiles. If SECURITYTYPE is PUBLIC for the offered

	subfile, then security checking is always successful. If SECURITYTYPE is PRIVATE for the offered subfile, then the usercode of the program offering the complementary subfile must equal the value of YOURUSERCODE for the offered subfile.
	Small Systems do no security checking for queue files. On A Series, whenever a program is run under a usercode, the default value for SECURITYTYPE is PRIVATE. This causes security checking to be performed for port files. In order to minimize changes to source code, the port file should be declared with a SECURITYTYPE of PUBLIC. (The "COBOL74 Interface" subsection shows the syntax needed to accomplish this.) If all programs using the port file are run under the same usercode, SECURITYTYPE can be left PRIVATE.
YOURUSERCODE	This attribute applies to each subfile. YOURUSERCODE specifies the usercode under which the program opening the complementary subfile must be running if the value of SECURITYTYPE is PRIVATE for the offered subfile. YOURUSERCODE can be one to 17 characters long. The default value is the usercode of the program offering the subfile.

The following two attributes are used in the matching process only when the subfiles will be linked between two hosts on a BNA network. If the programs will be run on only one host, it is not necessary to use these attributes.

HOSTNAME	HOSTNAME is a string of one to 17 characters that must begin with a letter. This attribute applies to each subfile. HOSTNAME specifies the name of the host on which the program with the complementary subfile is running. The value of HOSTNAME for the subfile must equal the value of MYHOSTNAME for the complementary subfile in order for the two subfiles to match. A null value for HOSTNAME (".") matches any value of MYHOSTNAME.
MYHOSTNAME	The MYHOSTNAME attribute is a read-only attribute and it applies to the port file

as a whole. MYHOSTNAME is the name of the local host on which the program is running.

MYHOSTNAME is set by the system when the program is first run. The value of MYHOSTNAME must match the value of HOSTNAME for the complementary subfile for the two subfiles to match.

The following examples further illustrate the A Series matching process.

```
Port File Matching Example 1
----------------------------

        Port File 1                            Port File 2

+-------------------------+             +-------------------------+
|  USERCODE - A           |             |  USERCODE - A           |
|  TITLE - PFA            |             |  TITLE - PFA            |
|  MYNAME - "."           |             |  MYNAME - "."           |
|  SECURITYTYPE - PRIVATE|              |  SECURITYTYPE - PRIVATE|
|                         |             |                         |
|       SUBFILE 1A        |             |       SUBFILE 2A        |
| +--------------------+  |             | +--------------------+  |
| : YOURNAME - "."     :  |<- Subfile ->| : YOURNAME - "."     :  |
| : YOURUSERCODE - A   :  |<- Matches ->| : YOURUSERCODE - A   :  |
| +--------------------+  |             | +--------------------+  |
|                         |             |                         |
+-------------------------+             +-------------------------+
```

Example 1 is the simplest example of subfiles that match. Both subfiles were offered by programs running under the same usercode. The user did not specify values for any of the port file attributes in the program, therefore, the default values were used. Both YOURNAME and MYNAME defaulted to a value of "." (null). The TITLE of both port files was the same. While SECURITYTYPE was PRIVATE, both programs ran under the same usercode, so YOURUSERCODE matched the other program's usercode. YOURNAME matched MYNAME for both port files.

Port File Matching Example 2

```
        Port File 1                            Port File 2

+-------------------------+            +-------------------------+
|  USERCODE - A           |            |  USERCODE - H           |
|  TITLE - PFA            |            |  TITLE - PFA            |
|  MYNAME - "ABC"         |            |  MYNAME - "123"         |
|  SECURITYTYPE - PUBLIC  |            |  SECURITYTYPE - PUBLIC  |
|                         |            |                         |
|       SUBFILE 1A        |            |       SUBFILE 2A        |
| +---------------------+ |            | +---------------------+ |
| : YOURNAME - "."      : |<- Subfile ->| : YOURNAME - "ABC"    : |
| : YOURUSERCODE - A    : |<- Matches ->| : YOURUSERCODE - H    : |
| +---------------------+ |            | +---------------------+ |
|                         |            |                         |
+-------------------------+            +-------------------------+
```

In the second example, the subfiles are matched because the TITLE of each file is the same, SECURITYTYPE is PUBLIC for both files (the different usercodes do not matter), and YOURNAME of subfile 1A is null. Therefore, it matches any MYNAME for port file 2, and YOURNAME (ABC) of subfile 2A matches MYNAME (ABC) of port file 1.

Port File Matching Example 3

```
        Port File 1                            Port File 2

+------------------------+             +------------------------+
|  USERCODE - A          |             |  USERCODE - A          |
|  TITLE - PFA           |             |  TITLE - PFA           |
|  MYNAME - "ABC"        |             |  MYNAME - "123"        |
|  SECURITYTYPE - PUBLIC |             |  SECURITYTYPE - PUBLIC |
|                        |             |                        |
|       SUBFILE 1A       |             |       SUBFILE 2A       |
| +--------------------+ |             | +--------------------+ |
| : YOURNAME - "ABC"   : |<-   No    ->| : YOURNAME - "ABC"   : |
| : YOURUSERCODE - A   : |<-  Match  ->| : YOURUSERCODE - H   : |
| +--------------------+ |             | +--------------------+ |
|                        |             |                        |
|       SUBFILE 1B       |             |       SUBFILE 2B       |
| +--------------------+ |             | +--------------------+ |
| : YOURNAME - "123"   : |<- Subfile ->| : YOURNAME - "ABC"   : |
| : YOURUSERCODE - A   : |<- Matches ->| : YOURUSERCODE - H   : |
| +--------------------+ |             | +--------------------+ |
|                        |             |                        |
+------------------------+             +------------------------+
```

In Example 3, subfile 1A was not matched to subfile 2A because YOURNAME (ABC) of subfile 1A did not match MYNAME (123) of port file 2. (Remember, MYNAME applies to the port file as a whole.) Subfile 1B was matched with subfile 2B because TITLE was equal, SECURITYTYPE was PUBLIC, and YOURNAME of each subfile equaled MYNAME of the opposite port file.

In this example, the system could have matched and linked either subfile 1B and subfile 2B, or subfile 1B and subfile 2A. The subfile that is matched depends on which subfile was open, or, if subfiles 2A and 2B were open at the same time, the first subfile opened.

Leaving MYNAME and YOURNAME with their default values can cause problems by linking the wrong subfiles together. Using the One to Many example from the previous subsection, the subfiles from two of the reading programs could get linked together, leaving two subfiles from the writing program with no complementary subfiles (subfiles in the same port file are never linked together). This situation depends on the order in which the subfiles were opened and whether the writing program had performed an OPEN OFFER. For more information, see "COBOL74 Interface," later in this section.

To eliminate this possibility and to avoid elaborate startup procedures for programs, we recommend that the port file's MYNAME be set to the name of the program. (If there is more than one port file in a program, use a unique MYNAME for each port file.) The subfile's YOURNAME should be set to the name of the program to which it is to be linked. Because MYNAME and YOURNAME can be up to 100 characters long, the values could be the object code name of the program or even a descriptive title. MYNAME can be set in the COBOL74 FD and YOURNAME can be set for each subfile at the beginning of the program. For more information, see the "COBOL74 Interface," later in this section.

If two sets of the same programs are running at the same time, the subfiles may be cross-connected between the two systems of programs. To avoid this, keep the value of SECURITYTYPE as PRIVATE and run each system of programs under a different usercode.

The second program in the Message Tanking example previously shown can be eliminated by using two port files in program 1. By using appropriate values for MYNAME and YOURNAME, a subfile in the first port file can be linked to a subfile in the second port file to provide a simulation of message tanking. Messages can be written into one subfile and read out of the other subfile.

COBOL74 INTERFACE

This subsection discusses the conversion of Small Systems COBOL(68) and COBOL74 queue file constructs to the equivalent A Series COBOL74 syntax. The discussion is divided into the three parts based on where the constructs occur in the COBOL program: the Environment Division, the Data Division, and the Procedure Division.

Except where noted, the discussion of the Small Systems syntax applies to both COBOL(68) and COBOL74. The A Series syntax applies to COBOL74.

Environment Division

The SELECT statement is the only statement that requires change in the Environment Division.

Small Systems Examples

```
                                            (COBOL74)

     SELECT QFILE ASSIGN TO QUEUE       SELECT QFILE ASSIGN TO QUEUE
         RESERVE 3 AREAS.                   ACTUAL KEY IS ACT-KEY
                                            FILE STATUS IS QFILE-STATUS.
```

To change the SELECT statement

1. Change QUEUE to PORT.

2. Remove the RESERVE clause, it is not necessary in the A Series since all messages are in memory.

The ACTUAL KEY and STATUS clauses are the same. On A Series, the ACTUAL KEY is required if there is more than one subfile.

All other statements remain the same. The resulting A Series program is:

```
                                            (COBOL74)

     SELECT QFILE ASSIGN TO PORT.       SELECT QFILE ASSIGN TO PORT
                                            ACTUAL KEY IS ACT-KEY
                                            FILE STATUS IS QFILE-STATUS.
```

FILE STATUS Values

The A Series values for FILE STATUS are similar to those on the Small Systems. The values are two characters long, the first character is status key 1 and the second character is status key 2.

The following is a list comparing the Small Systems values and A Series values for FILE STATUS.

Small	A Series	Meaning
00	00	Sucessful completion of I/O.
10	10	On Small Systems, a status key 1 value of "1" indicates a read was attempted on an

		empty queue file with no other programs connected to this queue by way of an OPEN OUTPUT or OPEN I-O.
		On A Series, a status key of "1" indicates that a READ was attempted when there was no next logical record and the connection between the two subfiles was severed.
		Functionally, these two values are equivalent.
20	20	On Small Systems, a status key 1 value of "2" indicates that a READ or WRITE occured on a queue file when the content of the ACTUAL KEY data item was less than zero or greater than the number of subfiles in the queue file family (Q.FAMILY.SIZE).
		On A Series, a status key 1 value of "2" indicates that a READ or WRITE statement had an ACTUAL KEY data item with a value of less than zero or greater than the number of subfiles in the port file (MAXSUBFILES).
		Functionally, these two values are equivalent.
34	34	On both Small Systems and A Series, this value indicates a boundary violation. This means that the value of the ACTUAL KEY data item was less than zero or greater than MAXSUBFILES.
-	81	This A Series value means that the subfile was not successfully opened (matched) after an OPEN OFFER or OPEN AVAILABLE statement.
-	82	This A Series value means that when a CLOSE was attempted on a subfile, an error occurred.
94	94	On both Small Systems and A Series this value means that a READ WITH NOWAIT was attempted but no data (messages) was available.

95	95	On both Small Systems and A Series this value means that a WRITE WITH NOWAIT was attempted but no buffer was available.

Data Division

The Data Division requires changes to the FILE CONTAINS and MAXSUBFILES.

Small Systems Examples

```
(COBOL(68))
FD  QFILE
    VALUE OF Q-MAX-MESSAGES IS 20.
01  QFILE-REC           PIC X(80).

(COBOL(68))
FD  QFILE
    FILE CONTAINS 3 QUEUES
    VALUE OF Q-MAX-MESSAGES IS 4.
01  QFILE-REC           PIC X(80).

(COBOL74)
FD  QFILE
    VALUE OF MAXSUBFILES IS 3
    MAXCENSUS IS 10.
01  QFILE-REC           PIC X(80).
```

To change the Data Division

1. Change Q-MAX-MESSAGES to MAXCENSUS.
2. COBOL(68): Change the FILE CONTAINS <n> QUEUES to MAXSUBFILES IS <n>.
3. Change MAXSUBFILES to reflect the number of subports rather than subqueues.

The resulting A Series program is

```
FD  QFILE
    VALUE OF MAXCENSUS IS 20.
01  QFILE-REC          PIC X(80).
```

```
FD  QFILE
    VALUE OF MAXSUBFILES IS 3
    MAXCENSUS IS 4.
01  QFILE-REC         PIC X(80).

FD  QFILE
    VALUE OF MAXSUBFILES IS 3
    MAXCENSUS IS 10.
01  QFILE-REC         PIC X(80).
```

Other file attributes can be set in the VALUE clause in the COBOL74 FD. If the attribute is a subfile attribute, declaring the attribute in the FD gives the same value for each subfile in the port file. For examole:

```
FD PORTFILE
     VALUE OF MAXCENCUS IS 10
              MAXSUBFILES IS 5
              MYNMAE IS "PROGRAM1"
              YOURNAME IS "PROGRAM2"
              SECURITYTYPE IS PUBLIC.
```

The attributes MAXCENSUS and YOURNAME are subfile attributes. Therefore, values declared here apply to each of the five subfiles in the port file. The attributes specified in the FD can be changed with the CHANGE statement as explained in "Procedure Division," the next subsection.

Procedure Division

The Procedure Division requires changes in

1. The Declaratives Section.
2. The OPEN statement.
3. The READ statement.
4. The WRITE statement.
5. The CLOSE statement.
6. The CHANGE statement.
7. The IF statement.

8. The MOVE statement.

9. The DISPLAY statement.

The DECLARATIVES Section

Small Systems COBOL(68) allows the DECLARATIVES section to contain a USE statement that applies to queue files. The Small Systems USE statement has the options FOR Q-FULL and FOR Q-EMPTY available. A Series COBOL74 has no equivalent USE options. Therefore, the Q-FULL and Q-EMPTY options must be removed. Though A Series does not have these options, their functions can be simulated.

To simulate the Small Systems Q-EMPTY option, access the A Series CENSUS attribute in an IF statement before a READ.

To simulate the Small Systems Q-FULL option, access the A Series OUTPUTEVENT in an IF statement before a WRITE.

For more information, see the discussion about the IF statement, later in this subsection.

The OPEN Statement

The A Series OPEN statement is similar to the Small Systems OPEN statement but there are differences that may need conversion.

On Small Systems, when an OPEN is done for a queue file family, all subqueue files are opened. On A Series, the value of the ACTUAL KEY data item determines which subfiles are opened. When the value is zero, all subfiles whose FILESTATE is CLOSED are opened. When the value is greater than zero but less than or equal to MAXSUBFILES, only the subfile with the specified value is opened. When the value is less than zero or greater than MAXSUBFILES, an error results. Change your Small Systems programs accordingly.

Small Systems COBOL allow OPEN INPUT, OPEN OUTPUT, and OPEN I-O. A Series allow OPEN I-O, OPEN OFFER, and OPEN AVAILABLE. In most cases the OPEN option should be changed to I-O. The A Series COBOL74 OPEN options are explained below:

I-O

The program offers the subfile for matching and the program is suspended until the matching subfile is found. Use this option to progress your program because no I/O can be done on a subfile until it is matched with another subfile and then opened. Your Small Systems program may be written in such a way that it will read or write from a queue file without requiring an existing queue file.

If the subfile being opened will be connected to a subfile on a remote host, there is a condition you should be aware of. During an OPEN I-O for a subfile, if the host specified by the HOSTNAME for that subfile becomes unreachable during the open operation, the program will resume execution. The FILESTATE of the subfile will be AWAITINGHOST and the SUBFILEERROR for the subfile will be UNREACHABLEHOST. If it is unacceptable to have the program resume execution in this case, the program must be changed to detect this condition.

The following COBOL74 program excerpt shows how to detect this error. The port file used has only one subfile (no subfile index is necessary).

```
                        OPEN I-O PFILE.
error in open ---->     IF PFILE-STATUS = "81"
                            IF ATTRIBUTE FILESTATE OF
                                 PFILE = VALUE AWAITINGHOST
                               AND
                               ATTRIBUTE SUBFILEERROR OF
                                 PFILE = VALUE UNREACHABLEHOST
                            DISPLAY "UNREACHABLE HOST ERROR.  "
                                    "PORT FILE NOT OPENED"
                            STOP RUN
                        ELSE
                            DISPLAY "ERROR ON PORT FILE OPEN.  "
                            DISPLAY "FILESTATE =" ATTRIBUTE
                                FILESTATE OF PFILE
                            DISPLAY "SUBFILEERROR = " ATTRIBUTE
                                SUBFILEERROR OF PFILE
                            STOP RUN.
```

	PFILE-STATUS is the data item specified in the FILE STATUS clause. The IF statement is explained in more detail below.
OFFER	The program offers the subfile for matching and continues execution. The program does not wait for matching subfiles to be found. If a read or write is attempted for a subfile that was open offered and no match was found, an end-of-file condition will be returned. The attribute FILESTATE can be used to determine the result of the OPEN.
AVAILABLE	The program offers the subfile for matching and continues execution. If a matching subfile is not found, the subfile is not left offered and cannot be matched to any subsequently offered subfile. (The subfile must be opened again.) The SUBFILEERROR attribute can be checked for a NOFILEFOUND value if a matching subfile was not found. Also, the value of FILE STATUS will be "81" if this occurs.

The value of the FILE STATUS data item will be "81" if an error occurred on the OPEN. Note that it is not an error when a subfile is open offered and no matching subfile was found. The FILE STATUS value will be "00" in this case.

The Small Systems LOCK and LOCK ACCESS options are not allowed for port files.

Recommended Progression

1. Change OPEN INPUT and OPEN OUTPUT to OPEN I-O. The program is suspended until the matching subfile is found.

2. Remove the LOCK and LOCK ACCESS options.

3. If an ACTUAL KEY data item was declared for the port file and the OPEN statement occurs several times in the program, add a MOVE 0 TO <ACTUAL KEY data item> before each OPEN. This statement is needed because the value of the ACTUAL KEY data item can be changed by previous READ and WRITE statements.

The READ Statement

The READ statement is almost the same on A Series as it is on Small Systems but there are differences in the way it works.

On Small Systems, the value of the ACTUAL KEY data item determines which subqueue to read in a queue file family. On A Series, when the READ statement is executed, the contents of the ACTUAL KEY data item determine the subfile read. (If no ACTUAL KEY was declared or the file contains one subfile, the program does not have to maintain the value of the ACTUAL KEY.) If the ACTUAL KEY is zero, then a non-selective READ is performed. On Small Systems, when a non-selective READ is done on a queue file family, the system checks each subqueue in order, starting at one, until it finds a non-empty subqueue. On A Series, the subfiles are checked in order starting with the subfile indexed by the value of the LASTSUBFILE attribute plus one, until a non-empty subfile is found. If the last subfile in the port file is empty, the check will proceed to the first subfile. The value of the ACTUAL KEY data item is updated to the number of the subfile read.

If the value of the ACTUAL KEY is greater than zero, the subfile specified by the value is be read. If the value of the ACTUAL KEY is out of range, then an error occurs and the FILE STATUS will be "34".

If a queue file is progressed into a port file with multiple subfiles, insert a MOVE 0 TO <ACTUAL KEY data item> before each READ. This simulates the read from the queue file on Small Systems. Remember that the ACTUAL KEY value is updated after each READ and WRITE, if the ACTUAL KEY is not set to zero before each READ, some subfile may never be read.

The Small Systems NO WAIT option has the same meaning on A Series. When a READ is executed with a NO WAIT option, the program is not suspended waiting for a message from the system. The value of FILE STATUS will indicate if the READ WITH NO WAIT was unsuccessful.

If a READ was done and NO WAIT was not specified, the program will be suspended until a subfile has a message in it. This is the same as Small Systems.

If a READ (with or without NO WAIT) is done on a subfile and no messages are in the subfile and the program writing messages to the subfile has closed it, the system terminates the READ with an end-of-file condition. (The AT END or INVALID KEY branch is taken.) If there are still messages in the subfile that was closed by the program writing messages to it, the subfile is kept open until all the messages are read, then an end-of-file occurs.

If the program previously did a READ and is suspended waiting for a message and the other program closes the subfile being read, the system resumes execution of the program and returns an end-of-file condition. (The AT END or INVALID KEY branch is taken.) If the program did a general READ, the AT END branch is taken when all the subfiles are closed by the other programs.

Recommended Progression

The READ statements have to be changed only when the ACTUAL KEY data item is declared for the port file and there is more than one subfile. If the READ statements need to be changed, add MOVE 0 TO <ACTUAL KEY data item> before each READ statement.

The WRITE Statement

The syntax for the WRITE statement is the same on the A Series as it is on the Small Systems, however, there are operational differences.

On Small Systems, the value of the ACTUAL KEY data item specifies the subqueue in a queue file family to which the data is to be written. On A Series, the ACTUAL KEY determines which subfile.

On Small Systems, the ACTUAL KEY data item can never be zero; on A Series, it can. When the ACTUAL KEY is zero, the same record will be written to all open subfiles. When the ACTUAL KEY is greater than zero, data is written to the specified subfile. When the ACTUAL KEY data item is out of range, an error occurs.

Unless the port file in the program contains only one subfile, a MOVE statement must be inserted before the WRITE statement to set the value of the ACTUAL KEY data item. If the program does not do a MOVE, the program will either do a broadcast WRITE or a write to whatever subfile is specified by the ACTUAL KEY data item. If the last I/O statement was a READ, the WRITE will be to the same subfile that was just read. This may be acceptable in your program's logic, but check to be sure.

If a program does a write, with or without the NO WAIT option, and the program connected to the subfile closed it, an end-of-file condition is returned. (The INVALID KEY branch is taken.)

If the program is suspended because no buffers are available for the WRITE (the NO WAIT option was not specified), the system resumes execution of the program and an end-of-file condition is returned. (The INVALID KEY branch is taken.)

Recommend Progression

To progress the program, unless the port file in the program contains only one subfile, it must be decided from the program's logic as to whether:

1. The ACTUAL KEY from the last READ or WRITE can be used again,

   or

2. Whether a MOVE <number> TO <ACTUAL KEY data item> must be inserted before each WRITE statement.

The WAIT Statement

The options of the WAIT statement must be changed to port file syntax. Each of the Small Systems options are listed below with their A Series equivalents.

WRITE-OK	The port file equivalent is OUTPUTEVENT. A WAIT on OUTPUTEVENT suspends the program until a buffer is available for a write to a particular subfile. A subfile index is required in the port file unless there is only one subfile in the port file.
READ-OK	The port file equivalent is INPUTEVENT. When a WAIT on INPUTEVENT is executed, the program waits for a message in a subfile. A subfile in the WAIT statement causes the program to wait for a message in that particular subfile. If no subfile index is specified, the program resumes when there is a message in any of the subfiles.

Queue/Port Files

USING	The USING option is not allowed on the A Series.
GIVING	The A Series equivalent is GIVING and it works on the A Series just as it does on the Small Systems.

For COBOL(68) programs, when an arithmetic expression is used as the first item in the event list, it represents, in tenths of seconds, the amount of time the program will be suspended. For Small Systems and A Series COBOL74, the arithmetic expression represents the number of seconds the program will be suspended. The Burroughs to Burroughs Translator (CTA) for COBOL(68) to COBOL74 properly converts this.

On Small Systems, when a program is waiting on READ-OK and all other writing programs using the queue file close it, the READ-OK is made TRUE. The next read on the queue file gets an end-of-file. This is not the case for A Series programs waiting on INPUTEVENT. When the program is waiting for the INPUTEVENT for a subfile to happen, it is possible for the other program using the subfile to close it. The program waiting on the INPUTEVENT will wait forever.

In order to prevent this from happening, another event file attribute called CHANGEEVENT should be added to the WAIT statement. A WAIT on the CHANGEEVENT for the file, or on one of its subfiles, will suspend the program until the FILESTATE attribute changes for the file or the subfile respectively. The FILESTATE attribute will change when a program closes the subfile. The CHANGEEVENT attribute should come after the INPUTEVENT attribute in the WAIT statement.

Because the READ statement will probably be a general read (any subfile), the result of the WAIT statement must be checked to see which event happened. The following example shows this for a converted queue file QFILE.

Example

```
     Small Systems
     -------------

     WAIT UNTIL READ-OK ON QFILE.
     READ QFILE AT END
         CLOSE QFILE.

     Large Systems COBOL74 (one subfile in the port file)
     ---------------------

     WAIT ATTRIBUTE INPUTEVENT OF QFILE,
          ATTRIBUTE CHANGEEVENT OF QFILE GIVING WAIT-NO.
     IF WAIT-NO = 1
         PERFORM MESSAGE-HANDLER
     ELSE IF WAIT-NO = 2
         IF ATTRIBUTE FILESTATE OF QFILE =
                                VALUE DEACTIVATED
              CLOSE QFILE.

     Large Systems COBOL74 (more than one subfile)
     ---------------------

     WAIT ATTRIBUTE INPUTEVENT OF QFILE,
          ATTRIBUTE CHANGEEVENT OF QFILE GIVING WAIT-NO.
     IF WAIT-NO = 1
         PERFORM MESSAGE-HANDLER
     ELSE IF WAIT-NO = 2
         MOVE ATTRIBUTE CHANGEDSUBFILE OF QFILE TO SUB-NO
         IF ATTRIBUTE FILESTATE OF QFILE (SUB-NO) =
                                VALUE DEACTIVATED
              MOVE SUB-NO TO QFILE-ACTUAL-KEY
              CLOSE QFILE.
```

The WAIT statement is executed, and when one of the events happens, the ordinal number of that event is put into WAIT-NO. If WAIT-NO equals 1 then the event INPUTEVENT happened and the program can read the port file (the program will not hang on the read). If WAIT-NO = 2 then the event CHANGEEVENT happened and should be checked to see if a subfile closed. In the second Large Systems COBOL74 example, the attribute CHANGEDSUBFILE is the subfile index of an arbitrary subfile whose CHANGEEVENT has "happened." Because the event could be other than the closing subfile, an IF is added to check. If the value of FILESTATE is DEACTIVATED the subfile can be closed.

For more information on the file attributes CHANGEEVENT, CHANGEDSUBFILE, and FILESTATE, see "Port File Attributes" later in this section. Detailed information on the IF and MOVE statements is presented later in this subsection.

When a program is waiting on an OUTPUTEVENT for a subfile, a similar situation can occur. If the other program closes the subfile, then the program waiting on the OUTPUTEVENT for that subfile will wait indefinitely.

To prevent this from happening the CHANGEEVENT file attribute should be added to the WAIT statement. The CHANGEEVENT attribute should follow the OUTPUTEVENT attribute in the WAIT statement. The example below shows the use of CHANGEEVENT in a WAIT on OUTPUTEVENT statement.

Example

```
Small Systems
-------------
WAIT UNTIL WRITE-OK ON QFILE.
WRITE QFILE-REC.

Large Systems COBOL74 (one subfile)
---------------------

WAIT ATTRIBUTE OUTPUTEVENT OF QFILE,
     ATTRIBUTE CHANGEEVENT OF QFILE
     GIVING WAIT-NO.
IF WAIT-NO = 1
    PERFORM MESSAGE-WRITER
ELSE IF WAIT-NO = 2
    IF ATTRIBUTE FILESTATE OF QFILE =
                               VALUE DEACTIVATED
        CLOSE QFILE.

Large Systems COBOL74 (more than one subfile):
---------------------

WAIT ATTRIBUTE OUTPUTEVENT OF QFILE (SUB-NO),
     ATTRIBUTE CHANGEEVENT OF QFILE (SUB-NO)
     GIVING WAIT-NO.
IF WAIT-NO = 1
    PERFORM MESSAGE-WRITER
ELSE IF WAIT-NO = 2
    IF ATTRIBUTE FILESTATE OF QFILE (SUB-NO)
                           = VALUE DEACTIVATED
        MOVE SUB-NO TO QFILE-ACTUAL-KEY CLOSE QFILE.
```

The data item SUB-NO is used as a subfile index in the preceding example.

When the program is resumed after the WAIT statement, WAIT-NO will contain the number of the event that happened. If WAIT-NO = 1 then a write can be done without the program hanging on a no-buffers-available condition. If WAIT-NO = 2 then the FILESTATE changed for the subfile. If the FILESTATE is DEACTIVATED then the subfile can be closed.

The check for the FILESTATE DEACTIVATED is present because there are other FILESTATEs that could have occurred. If the same subfile is being used for 2-way communication and there are some messages left to be read, then the FILESTATE is DEACTIVATIONPENDING. In this case the program can read the rest of the messages from the subfile before closing it. If BNA is used, the FILESTATEs SHUTTINGDOWN and BLOCKED can occur. If the subfile is only being written to and BNA is not used, then the following statement can be deleted:

```
    IF ATTRIBUTE FILESTATE OF QFILE (SUB-NO) =
                                 VALUE DEACTIVATED
```

Deleting this statement causes the subfile to be closed without further checks into why the CHANGEEVENT occurred.

Recommended Progression

1. Change the WAIT WRITE-OK ON <queue file name> to WAIT ATTRIBUTE OUTPUTEVENT OF <port file name> (subfile index).

2. Change WAIT READ-OK ON <queue file name> to WAIT ATTRIBUTE INPUTEVENT OF <port file name>.

3. Remove the USING option.

4. Change the arithmetic expression from tenths of seconds to seconds. If you run the program through the Burroughs to Burroughs COBOL(68) to COBOL74 translator(CTA), the correct conversion is done.

5. Add CHANGEEVENT to a WAIT INPUTEVENT or WAIT OUTPUTEVENT statement. If OUTPUTEVENT is specified in the WAIT statement, use a subfile index for CHANGEEVENT. Code must be added after the WAIT statement to check for the event that terminated the WAIT.

The CLOSE Statement

The CLOSE statement on Small Systems is the same on A Series except for the action of the ACTUAL KEY and the NO WAIT options.

On Small Systems, CLOSE closes all subqueues in a queue file family. On A Series, CLOSE closes only the subfile specified by the ACTUAL KEY data item. A value of zero closes all open subfiles. A value greater than zero but less than or equal to MAXSUBFILES closes only the specified subfile. A value out of range causes an error.

Closing a subfile may take a significant amount of time if the complementary subfile is on a different host. Because the program waits until the close is finished, the delay may be unacceptable. The NO WAIT option allows the program to continue while the system closes the subfile. Unless the subfile is used again in the program, we recommend the use of the NO WAIT option when the complementary subfile is on another host.

Recommended Progression

1. If the port file will not be used again, insert a MOVE 0 TO <ACTUAL KEY data name> just before the CLOSE. This will cause all open subfiles to be closed.

2. Add the NO WAIT option on the CLOSE statement if the other subfile is on a different host.

The CHANGE Statement

The CHANGE statement is the same on A Series as it is for Small Systems COBOL74; it allows you to change port file attributes.

If the file attribute being changed is not a subfile attribute (such as MYNAME), specifying a subfile index causes a syntax error.

For those attributes that are subfile attributes (such as YOURNAME), a subfile index must be specified. If the index is zero, the change will apply to all subfiles in the port file. If there is only one subfile in the port file, the subfile index may be omitted.

For attributes that are both file and subfile attributes (such as MAXRECSIZE), if a subfile is not specified, the change applies to the file. If a subfile index is used, the change applies to the specified subfile.

Some port file attributes may only be changed at certain times, such as before the file is opened (like MYNAME). If the attribute is changed at the wrong time, an attribute error occurs.

Most file attribute errors are non-fatal. A fatal error will DiScontinue (DS) the program. If a non-fatal attribute error occurs, the value of the attribute that was to be changed is left at its previous value.

The IF Statement

Port file attributes can be accessed through the COBOL74 IF statement.

The use of the subfile index with the IF statement is just like that used for the CHANGE statement.

Some file attributes have valid values only after the file is opened.

The file attributes are of different types as explained in "Port Files Attributes," later in this section. Alphanumeric attributes can be compared with alphanumeric data items or string literals. Numeric attributes can be compared with numeric data items or numeric literals.

Mnemonic attributes can be compared with numeric literals or mnemonics. Use of mnemonics is recommended to improve the maintainability of a program. Boolean attributes can only be compared to the numbers 1 (TRUE), 0 (FALSE), VALUE TRUE, or VALUE FALSE. Event attributes can be used as simple conditions.

The MOVE Statement

The values of port file attributes can be moved to other data items.

File attributes cannot be used as the receiving field in the MOVE statement.

The use of the subfile index is just like that for the CHANGE statement.

The rule for the MOVE statement, given in the COBOL74 Reference Manual, applies to file attribute moves. Moving a boolean file attribute results in a 1 (TRUE) or a 0 (FALSE) being moved. Event file attributes cannot be moved.

The DISPLAY Statement

The DISPLAY statement allows the values of port file attributes to be displayed. The values for numeric, mnemonic, and Boolean file attributes can be displayed. A value of TRUE for Boolean items is displayed as +000001, FALSE is displayed as +000000. The values for alphanumeric file attributes cannot be displayed directly with a DISPLAY statement. The value must first be moved to an alphanumeric data item and then displayed. Event attributes cannot be displayed.

The use of the subfile index is just like that for the CHANGE statement.

QUEUE FILE ATTRIBUTES COMPARED TO PORT FILE ATTRIBUTES

The following is a list of the Small Systems queue file attributes with their A Series port file equivalents.

Small Systems	A Series
BUFFERS (BUF)	This Small Systems attribute is not applicable on A Series port files. To achieve a similar function on the A Series, use the MAXCENSUS attribute, described later in this section.

INPUT.SELECTIVITY (ISL)	There is no A Series equivalent. The A Series MAXSUBFILES attribute, described later in this section, is similar.
Q.FAMILY.SIZE (QFS) MAXSUBFILES (COBOL74)	The A Series MAXSUBFILES attribute is the equivalent of this Small Systems attribute. MAXSUBFILES is used to specify the number of subfiles in a port file.
Q.MAX.MESSAGES (QMX) Q-MAX-MESSAGES (COBOL(68)) MAXCENSUS (COBOL74)	MAXCENSUS is the A Series equivalent of this Small Systems attribute. MAXCENSUS is used to specify the number of messages that can be stored in a subfile before a "NO BUFFER AVAILABLE" error message is displayed. The maximum number of messages that can be stored is 63. For additional information, see the SUBFILEERROR attribute, described later in this section.

PORT FILE ATTRIBUTES WITH NO QUEUE FILE EQUIVALENTS

The following is a list of A Series port file attributes that have no Small Systems equivalents.

Each attribute description is followed by "file" or "subfile" to indicate whether the attribute applies to the entire file or to each subfile. Where the attribute is used for a subfile, a subfile index is necessary when checking or setting the attribute for port files with multiple subfiles. For additional information, see the "COBOL74 Interface", earlier in this section.

The file or subfile indicator is followed by the type of file attribute. There are five types that are valid for A Series COBOL74:

1. Alphanumeric. These file attributes are similar to an elementary alphanumeric DISPLAY item.

2. Numeric. These file attributes are similar to an elementary numeric DISPLAY item (signed with six digits, PIC S9(6)).

3. Mnemonic. These file attributes can be accessed by a mnemonic rather than a numeric value because the actual value is unrelated to its meaning (see FILESTATE and SUBFILEERROR shown later). Also, using mnemonics improves the readability of a program.

4. Boolean. These file attributes can be accessed the same as numeric attributes where 1 represents TRUE and 0 represents FALSE.

5. Event. These file attributes have two states: "happened" and "not happened." The two states are similar to the Boolean values TRUE and FALSE, respectively. The system will set the event file attribute to "happened" whenever the event described occurs. Event file attributes can be used as conditional expressions.

For more information on file attribute types, see Section 4 of the "A Series COBOL74 Reference Manual."

BLOCKSTRUCTURE (FILE, MNEMONIC)

The values of FIXED and EXTERNAL for the BLOCKSTRUCTURE attribute apply to port files. BLOCKSTRUCTURE is only meaningful for a READ. If BLOCKSTRUCTURE has a value of FIXED then the buffer is filled with blanks. If BLOCKSTRUCTURE has a value of EXTERNAL, only the data received is put into the buffer (there is no blank fill to the end of the buffer). The actual length of the data can be found by using the CURRENTRECORD attribute. The default value for BLOCKSTRUCTURE is FIXED.

CENSUS (FILE, SUBFILE, NUMERIC)

The CENSUS attribute returns the total number of messages queued for all subfiles in a port file or the number of messages queued for a specified subfile. This attribute is read-only (it cannot be set).

CHANGEDSUBFILE (FILE, NUMERIC)

This read-only attribute returns the subfile index of an arbitrary subfile whose CHANGEEVENT has "happened." If no subfile's CHANGEEVENT has happened, a 0 (zero) is returned. This attribute can be useful in a general handler routine for many subfiles.

CHANGEEVENT (FILE, SUBFILE, EVENT)

The CHANGEEVENT for a subfile is caused whenever the value of the FILESTATE attribute (described later) changes for that subfile. The CHANGEEVENT for the file has the value "happened" (TRUE) as long as the CHANGEEVENT for any subfile has the value "happened" (the CHANGEEVENT was caused). When the program checks the subfile's FILESTATE, the system resets the CHANGEEVENT for that subfile. When all subfile CHANGEEVENTs are reset, the system resets the file's CHANGEEVENT. This attribute is read-only.

COMPRESSION (SUBFILE, BOOLEAN)

Data sent between two subfiles can be compressed. This can be done only when the programs containing the subfiles are on different hosts. Support for compressing data is negotiated when the subfiles are opened and matched (each host must be able to handle compressed data or compression is not possible). If data compression is supported, then setting COMPRESSION to TRUE when the subfile is open will compress data written to the complementary subfile. The value for COMPRESSION can be changed at any time the subfile is open to selectively compress records. The value of COMPRESSION will be FALSE even after setting it to TRUE, if compression of data is not supported.

Compressing data involves a trade-off in terms of processor and data transfer time. Compressing data increases processor time, but decreases data transfer time. If the connection between the two hosts involves a slow transfer rate, data compression may be worth investigating.

CURRENTRECORD (SUBFILE, NUMERIC)

This attribute returns the length, in FRAMESIZE units, of the last record read or written. If the BLOCKSTRUCTURE attribute has a value of FIXED, then CURRENTRECORD will equal MAXRECSIZE.

FILESTATE (SUBFILE, MNEMONIC)

This read-only attribute indicates the logical state of the subfile. The logical state is specified using either the mnemonic or its equivalent value (0 through 8) as listed below.

Checking a subfile's FILESTATE causes the system to reset the subfile CHANGEEVENT.

Mnenonic	Description
CLOSED (0)	Indicates that the subfile is closed.
AWAITINGHOST (1)	Indicates that the host specified by the subfile's HOSTNAME attribute cannot be reached. The subfile remains in this state until the host is reachable. All I/O performed on this subfile returns an end-of-file.
OFFERED (2)	Indicates that an open was attempted on this subfile, but no matching subfile was found. All I/O performed on this subfile returns an end-of-file. If a different host was specified (via HOSTNAME), then that host was reached to check for a matching subfile.
OPENED (3)	Indicates that the subfile is now open and ready for input or output.
SHUTTINGDOWN (4)	Indicates that the system operator wants to end communication between the host on which the program is running, and the host to which the subfile is linked. The connection will not be severed until all subfiles between the two hosts are closed. This allows the program to close the subfile (finish processing messages, etc.). The port file remains open and all I/O operations are valid.
BLOCKED (5)	Indicates that the remote host to which the subfile linked is temporarily unreachable. The subfile remains open and all I/O operations are valid.
CLOSEPENDING (6)	Indicates that this program closed the subfile, but the complementary subfile has not been closed by the other program. When the complementary subfile is closed, the FILESTATE is changed to CLOSED.
DEACTIVATIONPENDING (7)	Indicates that the complementary subfile was closed but there are still messages

	queued for input by this program. Writes to this subfile return end-of-file, but reads are still valid.
DEACTIVATED (8)	Indicates that the complementary subfile was closed and there are no messages queued for input. The subfile cannot be read or written to, it can only be closed.

The FILESTATE values AWAITINGHOST, SHUTTINGDOWN, and BLOCKED will only occur for those subfiles that are linked to a subfile on a different host.

FRAMESIZE (FILE, NUMERIC)

The FRAMESIZE attribute specifies the number of bits transferred as a unit of data during an I/O operation. For port files the system actually transfers data in 8-bit units, but the user program can access the data in any of the sizes available in FRAMESIZE. The default value is 48 bits (six characters).

HOSTNAME (SUBFILE, ALPHANUMERIC)

The HOSTNAME attribute specifies the host on which the program with the complementary subfile is running. HOSTNAME is a string of one to 17 characters and must begin with a letter. HOSTNAME is used during the subfile matching process. See "Subfile Matching," earlier in this section.

INPUTEVENT (FILE, SUBFILE, EVENT)

If accessed for the port file, INPUTEVENT returns "happened" (TRUE) if the CENSUS file attribute is greater than zero. If a subfile index is used, INPUTEVENT returns "happened" if the CENSUS subfile attribute is greater than zero for the specified subfile. This attribute is similar to the Small Systems READ-OK. See the "COBOL74 Interface," earlier in this section. This is a read-only attribute.

LASTSUBFILE (FILE, NUMERIC)

The subfile index of the last subfile that was used for a successful I/O operation is contained in this attribute. This value is updated only if the last I/O was successful. LASTSUBFILE is useful after a general read (no subfile was specified to read from) to tell from which subfile the message came.

MAXCENSUS (SUBFILE, NUMERIC)

This attribute specifies the maximum number of input messages that can be stored in a subfile before the writing program gets a "NO BUFFER AVAILABLE" indication for that subfile. See "SUBFILEERROR," later in this subsection. The maximum value for MAXCENSUS is 63 messages.

MAXRECSIZE (FILE, SUBFILE, NUMERIC)

The MAXRECSIZE attribute can only be set for the port file as a whole. It is read-only for each subfile and can also be read for the port file. When the system matches two subfiles and links them together, the new MAXRECSIZE for each subfile is the smaller of the two original MAXRECSIZEs. MAXRECSIZE is set by default to the largest record size in the port file's COBOL FD. The value for MAXRECSIZE is in FRAMESIZE units.

MAXSUBFILES (FILE, NUMERIC)

This attribute specifies the maximum number of subfiles in the port file. Each subfile has an index numbered from 1 to MAXSUBFILES, inclusive. Once this attribute is set, its value cannot be decreased, only increased.

MYHOSTNAME (FILE, ALPHANUMERIC)

MYHOSTNAME specifies the name of the host on which the program is running. MYHOSTNAME is used in the subfile matching process. (See the preceeding subsection, "Subfile Matching.") This attribute is a read-only attribute.

MYNAME (FILE, ALPHANUMERIC)

MYNAME is a string from one to 100 characters. It is used during the subfile matching process. (See the preceding subsection, "Subfile Matching.")

OUTPUTEVENT (SUBFILE, EVENT)

The OUTPUTEVENT attribute for a subfile returns "happened" (TRUE) whenever output buffers are available for that subfile. The system resets OUTPUTEVENT whenever no output buffers are available. The OUTPUTEVENT attribute is similar to the Small Systems WRITE-OK. (See the "COBOL74 Interface" subsection.) This is a read-only attribute.

SECURITYTYPE (FILE, MNEMONIC)

SECURITYTYPE can be either PUBLIC or PRIVATE and determines the level of security checking performed during subfile matching. For more information, see the "Subfile Matching" subsection.

SUBFILEERROR (SUBFILE, MNEMONIC)

This read-only attribute is set after each READ, WRITE, OPEN, or CLOSE operation that affects that subfile. A list of the valid values is given below. Either the mnemonic or the numeric value, as listed below, can be used to specify the value. For additional information, see the "COBOL74 Interface" subsection.

Mnemonic	Description
NOERROR (0)	No error occurred.
DISCONNECTED (1)	Communication with the complementary subfile was lost. This happens when the connection with a remote host is severed without allowing the subfiles to close.
DATALOST (2)	The subfile may have closed (due to a failure in the BNA link) before transmission of data to the other subfile was complete. This can only happen when the two subfiles are on different hosts.
NOBUFFER (3)	A write with the NO WAIT option to this subfile failed because no buffer space was available. If NO WAIT was not specified in the WRITE, then the error will not occur (the program will hang on the write until a buffer is available).
NOFILEFOUND (4)	An open operation on this subfile failed because a matching subfile was not found. This error will only occur when the AVAILABLE option was specified in the OPEN statement (see the "COBOL74 Interface" subsection).
UNREACHABLEHOST (5)	The remote host became unreachable during the open operation. The subfile is not open and no I/O can be performed.
UNSUPPORTEDFUNCTION (6)	An attempted open on this subfile resulted in a request for an unsupported function.

TITLE (FILE, ALPHANUMERIC)

The TITLE attribute is a string between one and 17 characters long. The TITLE for a port file is used in matching subfiles. (See the subsection, "Subfile Matching.") The default for TITLE is the internal name of the file. TITLE cannot contain a slash (/).

YOURNAME (SUBFILE, ALPHANUMERIC)

YOURNAME is a string from 1 to 100 characters long. YOURNAME is used in the subfile matching process. (See the preceding subsection, "Subfile Matching.")

YOURUSERCODE (SUBFILE, ALPHANUMERIC)

The YOURUSERCODE attribute specifies the usercode under which the program opening the complementary file must be running. It is used only when SECURITYTYPE is PRIVATE. (See the subsection, "Subfile Matching," for more information.)

Further information about these file attributes can be found in the "A Series I/O Subsystem Reference Manual."

A SMALL SYSTEMS AND A SERIES REFERENCE MANUALS

The following is a list of Small Systems and A Series reference manuals.

SMALL SYSTEMS MANUALS

1108982	System Software Operation Guide, Volume 1.
1108966	System Software Operation Guide, Volume 2.
1057197	COBOL Reference Manual.
1108883	COBOL74 Reference Manual.
1090586	CANDE User's Manual.
1163920	GEMCOS User's/Reference Manual.
1106531	GEMCOS Formatting Guide.
1127222	DMSII Reference Manual.
1108875	DMSII Inquiry Reference Manual.
1057189	RPG Reference Manual.
1090594	SORT Reference Manual.
1108859	Communications Module (SYCOM) Reference Manual.
1073715	Network Definition Language (NDL) Reference Manual.

A SERIES MANUALS

5014574	DCALGOL Reference Manual.
5011760	ALGOL Reference Manual.
5014582	BINDER Reference Manual.
1169653	Burroughs Network Architecture (BNA) Program Agent User's Guide.
5011778	Remote Job Entry (RJE) Reference Manual.
5014301	Sort Reference Manual.

5014541	COBOL Reference Manual.
5014442	COBOL ANSI-74 Reference Manual.
5012222	DMSII Inquiry Software Operation Guide.
5001480	DMSII DASDL Reference Manual.
5012230	DMSII User Language Interface Software Operation Guide.
5001803	DMSII Utilities and Operations Guide.
5012263	Report Program Generator (RPG) Reference Manual.
5014483	I/O Subsystem Reference Manual.
5014541	CANDE Reference Manual.
5014962	CANDE Operations Manual.
1154481	Generalized Message Control System (GEMCOS) User's Reference Manual.
5011828	Network Definition Language (NDL) Reference Manual.
5011828	Network Definition Language II (NDL) Reference Manual.
5014426	System Software Utilities Reference Manual.
5014434	System Software Support Reference Manual.
5014418	System Software Site Management Reference Manual.
5014491	Operator Display Terminal (ODT) Reference Manual.
5011794	Work Flow Language (WFL) Reference Manual.
1154465	Advanced Data Dictionary Systems (ADDS) User's Guide.
1185337	Communications Management System (COMS) Migration Guide.
1154523	Communications Management System (COMS) Operator's Guide.

Small Systems and A Series Reference Manuals

1185238	Communications Management System (COMS) Planning and Installation Manual.
1154531	Communications Management System (COMS) Programmer's Guide.
1164027	Extended Retrieval With Graphic Output (ERGO) User's Guide.
1169521	Interactive Datacom Configurator (IDC) User's Guide.
1169588	Menu-Assisted Resource Control (MARC) User's Guide.
1154440	Screen Design Facility (SDF) User's Guide.
1169919	Printing Subsystem Overview
1169950	Printing Utilities User's Guide

GLOSSARY

This glossary contains definitions for many of the terms used in this manual. The terms are defined in alphabetical order.

ASCII

See "American Standard Code for Information Interchange."

audit file

Files that are produced for each Network Support Processor (NSP) and Data Communications Data Link Processor (DCDLP) by the datacomm subsystem procedures of the Master Control Program (MCP). The audited items are specified with a Menu Assisted Resource Control (MARC) menu selection or through an operator command.

audit trail

A file produced by the Accessroutines that contains various control records and a sequence of before-update and after-update record images resulting from changes to the data base. The audit trail is used to recover the data base and supply restart information to programs after a hardware or software failure has occurred.

backup

The copying of information to disk or tape to provide a means of restoring the information on the system as required.

BCL

See "Burroughs Common Language."

Burroughs Common Language (BCL)

A code using 6-bit character representation. BCL is not available on A Series and most B 5000/B 6000/B 7000 Series systems (such as B 5900 systems).

CD

See "Communication Description."

checkpoint

A place in a program where the program is to be stopped so that the current state of the program can be written to disk. After the program's state has been recorded, execution of the program is resumed where it left off. If the system halts unexpectedly before the program finishes, the program can be restarted at the point of its most recent checkpoint instead of at the beginning of the program.

checksum

A value used to detect certain classes of input/output errors. A checksum is computed for each data base file block by applying an equivalence operator to each word in the block. When the block is physically written, the checksum value is stored in a checksum word appended to the end of the block. When the block is read, the checksum is recomputed and the result is compared to the stored value. A checksum error occurs if the two values are not equal.

COBOL

COmmon Business-Oriented Language.

COBOL74

ANSI-74 COBOL.

collating sequence

The sequence in which a computer recognizes characters for purposes of sorting, merging, and comparing.

Communication Description (CD)

A message header that is passed with the message data received and sent by application programs. There are two versions of the CD in COBOL74, the standard EBCDIC CD (like B 1000 COBOL74 CDs) and the Binary CD that is used for the direct programmatic interface to the Communications Management System (COMS). The Binary CD provides routing information about the message data and allows use of COMS security and recovery functions, processing items, and routing by trancode.

communication description entry

In COBOL, an entry in the COMMUNICATION SECTION of the DATA DIVISION that is composed of the level indicator CD, a cd-name, and a set of clauses as required. It describes the interface between the Message Control System (MCS) and the COBOL program.

Communications Management System (COMS)

A general Message Control System (MCS) that supports a network of users and provides them with a consistent, on-line interface between the Data Communications Processor (DCP), Network Support Processor (NSP), or Data Communication DLP (DCDLP) and application programs that process transactions associated with remote terminals. COMS supports the processing of multi-program transactions as well as single-station and multi-station remote files.

compiler

A computer program that translates computer instructions written in a source language, such as COBOL, into machine-executable object code.

compiler control image (CCI)

A record in the source instructions beginning with a dollar sign ($) that contains one or more options that control various compiler functions. These specifications can appear anywhere in the source program, unless otherwise specified. A compiler control image is also referred to as a "compiler control record."

compiler control options

Individual compiler directions that appear on a compiler control image.

compiler dollar options

Individual compiler directions that appear on a compiler control image. Compiler dollar option are also referred to as a compiler control options.

COMS

See "Communications Management System."

COMS Control

A Communications Management System (COMS) internal library that initiates a Data Base library (DB library) for each data base that uses synchronized recovery, and a Transaction Processing library (TP library) for non-data base, transaction-processing programs that do not use synchronized recovery.

COMS transaction trail

A file, generated by the Transaction Processing (TP) library, that reflects such information as beginnings of jobs and ends of jobs. The file optionally provides a journal of query transactions not associated with any data base. The file also optionally provides statistical information on a transaction-by-transaction basis and can be used for security and accounting.

COMS Utility

The Communications Management System (COMS) program that defines and maintains the specifications stored in the COMS configuration file.

COMS window environment

The status of the windows currently available to a given station. The status of a window can be open, closed, suspended, or disabled.

conditional expression

In COBOL, a simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement.

A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program depends on this truth value.

conditional variable

In COBOL, a data item for which at least one value has a condition-name assigned to it.

Glossary

control file

In Data Management System II (DMSII), a file containing data file coordination information, audit control information, and dynamic data base parameter values. The control file provides data base interlock control; that is, it allows functions, such as recovery, to have exclusive use of the data base. In addition, the control file provides program-to-file and file-to-file compatibility.

control station

In CANDE, a station that allows CANDE network control commands to be entered.

In the Communications Management System (COMS), a control-capable station: a station with no restrictions on the use of COMS commands. A control station is either defined through the COMS Utility to be control-capable or in the Network Definition Language II (NDLII) or by using the Interactive Datacomm Configurator (IDC) to set its station attribute SPO to TRUE.

CREATIONDATE

The file attribute that gives the date on which a file was first locked on a disk or disk pack.

DASDL

See "Data and Structure Definition Language."

Data and Structure Definition Language (DASDL)

In Data Management System II (DMSII), the language used to describe a data base logically and physically, and to specify criteria to ensure the integrity of data stored in the data base. DASDL is the source language that is input to the DASDL compiler, which creates or updates the data base description file from the input.

data base

An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single Data and Structure Definition Language (DASDL) source file. A data base is considered a global entity that several applications can access and update concurrently. Ideally, all the permanent data pertinent to a particular application will reside in a single data base.

data set

In Data Management System II (DMSII), a collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets can contain other data sets, sets, and subsets.

The data in a data set is formatted in a special manner so that it can be accessed by DMSII software. Application programs are expected to access a data set through the DMSII software rather than directly as files.

data type

An interpretation applied to a string of bits.

Data types can be classified as structured or scalar. Scalar data types include real, integer, double precision, complex, logical (also called "Boolean"), character, pointer, and label. Structured data types are collections of individual data items of the same or different data types. An array is a data type that is a collection of data items of the same type. Records, structures, or files are data types that are collections of data items of one or more data types.

Most programming languages provide a declaration statement or a standard convention to indicate the data type of the variable used.

density

The number of bits per inch on a magnetic tape.

device

Any piece of I/O hardware, such as a Data Link Processor (DLP) or a peripheral unit.

disabled

In CANDE, the state of a station in which messages from the line it represents are being ignored by the system.

In COMS, the condition of being rendered incapable of exercising normal communication with the Communications Management System (COMS).

disk

A data storage device consisting of one or more circular platters that contain bits of information stored in concentric circles called tracks.

disk file

A file stored on a disk or disk pack.

disk pack

A disk that consists of multiple platters stacked vertically on a central spindle. Data on a disk pack is accessed by movable read/write heads. Some disk packs are removable. A disk pack is also referred to as a "pack."

embedded

In Data Management System II (DMSII), a data set, set, or subset contained within another data set. "Embedded" is the opposite of "disjoint."

enabled

The condition of being capable of normal communication with the Communications Management System (COMS).

extract file

A data file produced by extracting information from a data base using the Data Management System II (DMSII) Inquiry EXTRACT command. The format of the file is specified in the EXTRACT command.

FAST

See "File Access Structure Table."

File Access Structure Table (FAST)

A special file that is part of the access structure the system uses to locate disk files. The FAST contains a pointer to each disk file's header in the flat directory of each family.

file attribute

A parameter that describes a characteristic of a file and contains information the system needs to handle the file. Examples of file attributes are the file title, file kind, record size, number of areas, and date of creation.

footing

Text that appears at the bottom of each page of a document.

format

A specific arrangement of a set of data.

GEMCOS

See "Generalized Message Control System."

Generalized Message Control System (GEMCOS)

A Burroughs Message Control System (MCS) developed for on-line, multi-task systems. GEMCOS is transaction-oriented.

header

A sequence of characters preceding the text of a message, containing routing or other communications-related information.

hexadecimal literal

A character-string bounded by at signs (@). The string of characters must consist of one or more characters chosen from the set of hexadecimal digits consisting of the digits 0 through 9 and the characters A through F. The characters A through F are the hexadecimal digit representations for the decimal values 10 through 15, respectively.

host name

The name associated with a particular host. A host name consists of 1 to 17 alphanumeric characters, inclusive.

index random set

A structure of records allocated to particular tables based on a hashing function of the key.

Indexed Sequential Access Method (ISAM)

A method that provides efficient, yet flexible, random access to fixed-length records identified by multiple keys stored in disk files.

initialvalue

The value assigned to an item in a newly created record. An initialvalue can be explicitly specified for each item when it is declared in the Data and Structure Definition Language (DASDL). If no initialvalue is specified, a default initialvalue is assigned by DASDL.

ISAM

See "Indexed Sequential Access Method."

KIND

The file attribute that indicates the type of hardware device on which the file is stored.

manual subset

In Data Management System II (DMSII), a subset that has no condition specifying which data set records are to be included in the subset. The user must add and delete manual subset entries, using the INSERT and REMOVE statements.

In Extended Retrieval with Graphic Output (ERGO), a collection of indices or pointers to records in one data set. A manual subset is used to access selected members of that data set and to represent relationships between data records. The user must maintain the entries in a manual subset because these subsets do not specify WHERE conditions normally used by Data Management System II (DMSII) to maintain the sets and subsets.

MARC

See "Menu-Assisted Resource Control."

MAXRECSIZE

In CANDE, a file attribute that gives the maximum size, in frames, of records in a logical file. For port files, MAXRECSIZE specifies the maximum text size for all subfiles in the port file.

MCS

See "Message Control System."

memory

A temporary storage area where data and programs are placed while being processed.

Menu-Assisted Resource Control (MARC)

A menu-driven interface and transaction processor for users and operators of Burroughs A Series and B 5000/B 6000/B 7000 Series systems.

Message Control System (MCS)

A program that controls the flow of messages between terminals, application programs, and the Master Control Program (MCP). Burroughs MCSs include GEMCOS/MCS, SYSTEM/CANDE, SYSTEM/RJE, SYSTEM/COMS, SYSTEM/APL, and SYSTEM/DIAGNOSTICMCS.

message header

A sequence of characters preceding the text of a message, containing routing or descriptive information for the message.

monitor station

In the Communications Management System (COMS), a station that has been defined to display COMS activities specified as monitor options.

next record

The record that logically follows the current record of a file.

nonnumeric item

A data item whose contents can be composed of any combination of characters taken from the computer's character set. Certain categories of non-numeric items can be formed from more restricted character sets.

nonnumeric literal

A character string bounded by quotation marks ("). The string can include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

ODT

See "Operator Display Terminal."

open mode

In COBOL, the state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

Operator Display Terminal (ODT)

The system console device that allows the operator to enter commands directly to the operating system to perform various functions.

optional word

A reserved word included in a programming language to improve the readability of a source statement. The user can include or omit an optional word.

ordered

An adjective meaning maintained in a user-specified sequence.

processing-item library

In the Communications Management System (COMS), a user-written ALGOL library containing a set of procedures called processing items. A processing-item library can be called only by the Agenda Processor library of COMS to preprocess and postprocess messages as they are received and sent by programs.

queue

A logical collection of messages awaiting transmission or jobs awaiting processing.

queue name

A symbolic name that indicates to the Message Control System (MCS) the logical path by which a message or a portion of a completed message is accessible in a queue.

recovery

In Data Management System II (DMSII), a data base routine that is initiated following a hardware, software, or operations failure while the data base is in update mode. Recovery backs out any partially completed transactions by applying audit-trail images to the data base to restore it to its proper state. In addition, recovery passes restart information to the programs accessing the data base.

In Communications Management System (COMS), reconstruction of a data base after a system failure.

remap

In the Data Management System II (DMSII), a logical data record that redefines a physical data set record by omitting, reordering, or renaming the items.

reorganization

In Data Management System II (DMSII), the process of reordering or reformatting data sets, sets, or subsets. Reorganization can restore space in files, reorder data sets for more efficient retrieval, and reformat data set records when items are added, deleted, or changed.

report file

In COBOL, an output file with a description entry that contains a REPORT clause. The contents of a report file consist of records that are written under control of the Report Writer Control System (RWCS).

SECURITYTYPE

The file attribute that specifies the type of access allowed to a file.

sequential statement

In Pascal, a structured statement in which the subcomponent statements are executed in the order in which they appear, without conditions or repetitions.

serial number

The six-digit number an installation assigns to a disk or magnetic tape to uniquely identify it. The serial number is stored on the label of the disk or tape.

structure

In the Data Management System II (DMSII), any physical entity in a data base; particularly, a data set, set, subset, access, or remap.

table

In COBOL, a set of logically consecutive items of data defined in the DATA DIVISION by means of the OCCURS clause.

tag file

A file created during an Inquiry sort that contains key items and addresses of selected records and is passed to the system sort intrinsic.

unordered data set

In the Data Management System II (DMSII), a collection of related data records stored in a file in which the records are either fixed-format or variable-format.

variable format

In the Data Management System II (DMSII), a record that consists of two parts: a fixed part and a variable-format part. A single record description exists for the fixed part. The variable-format part can describe several variable parts. An individual record is constructed by using the fixed part alone, or by joining the fixed part with one of the variable parts.

WFL

See "Work Flow Language."

Work Flow Language (WFL)

The Burroughs language used to write jobs that control the flow of programs and tasks on the operating system.

Glossary

ZIP

An ALGOL statement that causes the Work Flow Language (WFL) compiler to be initiated. It is commonly used to initiate compilations automatically.

Index

Index

Index

Index

Index

Index

Index