UNISYS

ALLY® Software Development Environment

Application Migration Developer Notes

June 1988

Printed in U S America

Priced Item

UP–14220

Notice

Preface

This manual describes ALLY release 2.0.

The ALLY Software Development Environment can run on many computer systems, operating systems, and data access methods. Therefore, the ALLY manuals are generic—they describe the system-independent features of ALLY.

These developer notes tell you how to move an ALLY application among different data access methods, operating systems, or computer systems. These notes are a supplement to the standard set of manuals provided with ALLY.

These notes include information about:

- the AFILE Migrator and the Data Migrator
- differences among operating systems on which ALLY runs
- differences among access methods on which ALLY runs
- the general steps that you take to transport an application
- an example of how an application was moved from one environment to another

We assume that you are familiar with the ALLY Software Development Environment and the documentation conventions that are provided in the preface of the *Dialog User's Guide* (FCS002-3007). We also assume that you have read the *Utilities User's Guide* (FCS002-3014) and the developer notes that describe the access methods that you are using with your ALLY application. The developer notes include:

- *C-ISAM Developer Notes* (UP-12970)
- *ALLYpc Developer Notes* (UP-12639), which describes how ALLY works with dBASE III-compatible files
- *ORACLE Developer Notes* (UP-12640)
- *UNIFY Developer Notes* (UP-12969)

End of Preface

Contents

Figures

Tables

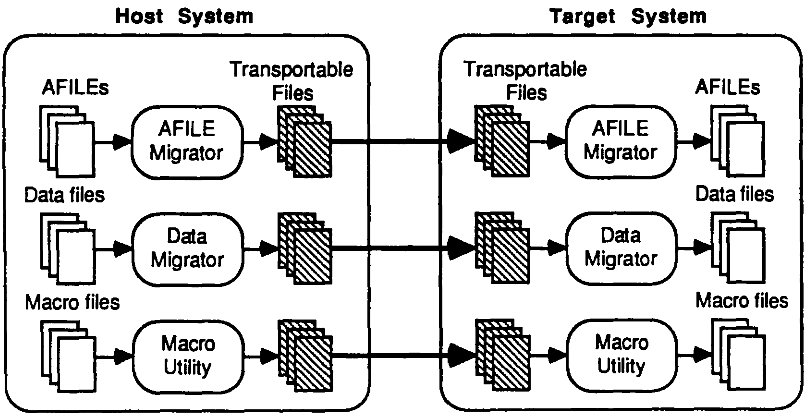
Application Migration Developer Notes

The ALLY Software Development Environment allows you to transport an *application*, its *data*, and its *macro-command files* among machines that use the UNIX operating system and the MS-DOS operating system.

You can also transport an application and its data among different access methods. The access methods ALLY applications can use are:

- C-ISAM
- dBASE III
- FX (fixed sequential)
- ORACLE
- UNIFY

Figure Migration-1 shows how you transport an ALLY application by moving its three basic parts: AFILEs, data files or data tables, and macro-command files. The utilities that you use to transport these parts are the AFILE Migrator, the Data Migrator, and the Macro Utility.



F002-0788-00

Figure 1. Transporting ALLY Applications

Using These Developer Notes

In these developer notes, we tell you how to move an ALLY application among data access methods, operating systems, or computer systems. We assume that you are familiar with ALLY, the Dialog, and the access methods that your application uses. We also assume that you have read the *Dialog User's Guide*, the *Utilities User's Guide*, and the developer notes for your application's access methods.

These developer notes include:

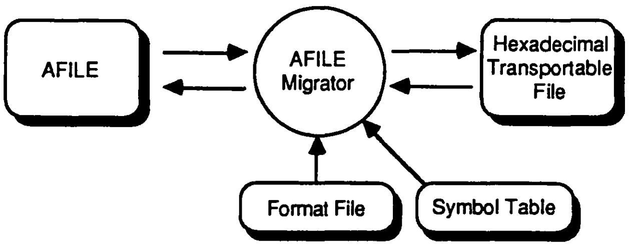
- an introduction to the utilities that you use to transport applications
- a summary of the steps that you must take to transport an application
- a discussion of how differences among computer systems, operating systems, and access methods affect the way that you transport applications
- an example of how to transport an application that was built on a UNIX system (with the ALLY FX access method) to a Unisys PC/IT (or compatible, with the MS-DOS operating system and the dBASE III access method)

The Application Migration Utilities

In this section, we introduce the three utilities that you use to transport applications.

The AFILE Migrator

The AFILE Migrator allows you to transport an application's AFILEs among different systems. You can use the AFILE Migrator, shown in Figure 2, to make a transportable version of an AFILE. Then, you can rebuild the AFILE from the transportable file—on the same system or a different system.

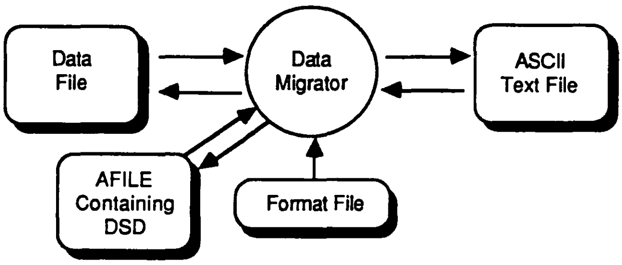


F002-0819-01

Figure 2. The AFILE Migrator

The Data Migrator

The Data Migrator (Figure 3) allows you to transport your application's data files from one ALLY system to another and among different access methods.



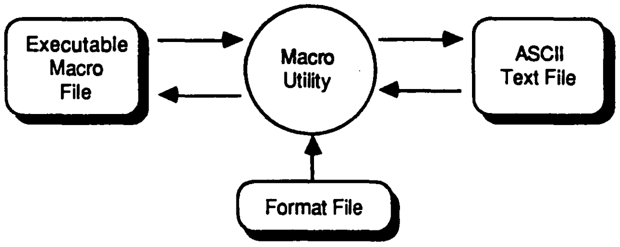
F002-0267-02

Figure 3. The Data Migrator

When the Data Migrator translates a data file to a transportable text file, it takes information about the data from a Base Data Source Definition (Base DSD) in the application's AFILE.

Macro Utility

While you are using an ALLY application, including the Dialog, you can store macro commands in files. A macro command contains a sequence of keystrokes and ALLY commands. The Macro Utility (Figure 4) allows you to transport macro-command files to different computers or operating systems. The Macro Utility allows you to construct a text version of a macro-command file on your host system. With the same utility, you can read a macro text file and construct an executable macro-command file on your target system.



F002-0580-00

Figure 4. The Macro Utility

Steps for Migrating an Application

In this section, we provide the general steps you must follow to transport an application from a host system to a target system.

The terms "host" and "target" do not imply that you are transporting an application to another computer system—you may just be transporting it to a different access method on the same system. In that case, you can migrate just the application's data files.

Before You Start

When you transport an application, the number of steps that are required and the way that you perform the steps can be different, depending on which operating system and access methods your host and target systems are using.

Therefore, before you transport an application, you must answer the following questions:

- Is the operating system on the target system different from the operating system on the host machine? If it is, you should read the "Operating System Differences" section of these developer notes.
- Will the application use a different access method after you transport it? If it will, you should read the "Access Method Issues" section, which tells you about the way that ALLY applications interact with different access methods. You should also read the developer notes that describe your target system's access methods.
- What are the names of your application's AFILEs, Base DSDs, and macro-command files? You will need these names when you construct transportable files with the ALLY utilities.
- Are you updating your application to run with a newer version of ALLY? If you are, you should read the "ALLY Revision Differences" section, which tells you how to transport an application from an older version of ALLY to a more recent version.
- Does your application use any external programs? If it does, you must transport the code for these programs to the target system and recompile the programs.

When you have answered these questions, you can transport your application. The next two sections describe the steps that you take on the host system and on the target system.

On the Host System

The general steps that you take on the host system to transport an application are summarized in the following list.

1) Use the AFILE Migrator to construct a transportable version of the application's AFILE. If the AFILE has an external symbol table, the AFILE Migrator will place it into the transportable version of the AFILE. You cannot migrate a symbol table by itself.

2) Use the Data Migrator to construct a transportable text file for each data file or table that the application uses.

3) If your application has macro commands, use the Macro Utility to construct a transportable text file for the application's macro-command files.

4) Move the transportable files to the target system. If your application uses external programs, move the source code for the programs. Make sure that you do not write over any files on the target system that have the same names as the files that you are transporting.

On the Target System

On the target system, take the following general steps to reconstruct an application.

1) Use the AFILE Migrator to reconstruct the application's AFILE. You must perform this step first because some of the other steps require information from this reconstructed AFILE.

2) If the application is using a different access method, use the Dialog to change the type of each Base DSD. You must perform this step before you use the Data Migrator to reconstruct the application's data files or data tables.

3) If the application is running on an operating system that is different from the host's operating system, use the Dialog to change the paths to the application's help and error message AFILEs and to the default printer.

4) Use the Data Migrator to reconstruct the application's data files or data tables.
5) If the application uses macro commands, use the Macro Utility to reconstruct the application's macro-command files.
6) If the computer systems or operating systems are different, you must change any operating-system-dependent calls in your external programs and recompile the programs.

Differences Among ALLY Systems

If you are transporting your application to a system that uses the same operating system and access methods as your host system, you can follow the general steps that we listed in the previous section, skipping the operating-system-dependent and access-method-dependent steps (steps 2, 3, and 6 on the target system).

When you transport an ALLY application to a different operating system or access method, you must be aware of the effect of these differences on the way that you use the ALLY utilities. In the next two sections, we tell you about these differences.

Operating System Differences

The UNIX operating system and the MS-DOS operating system handle file names differently. These differences affect the path names that an application uses to access its external files. Table 1 lists the operating system differences between UNIX and MS-DOS.

Table 1. Operating System Differences

	UNIX	MS-DOS
File name length	14 characters, including periods and extensions (e.g., filename.one.a)	12 characters with a maximum of 8 characters in the name and a 3-character extension, separated by a single period (e.g., filename.txt)
Path separators	Slashes (/) (e.g., usr/app/afile.a)	Back slashes (\) with a colon (:) after the drive specifier (e.g., c:\app\afile.a)
Paths	ALLY files are stored in a tree structure (e.g., the path to ALLY's errors.e file is {ally}/afiles/errors/errors.e)	ALLY files are stored in a single directory (e.g., the path to ALLY's errors.e file is {ally}\errors.e)
Case in file names	File names are case sensitive (e.g., EMPLOYEE and employee are different)	File names are not case sensitive (e.g., EMPLOYEE and employee are the same)
Number of open files	N/A	Allows a maximum of 20 open files per process

Because of the differences listed in Table 1, you must change the path name to any external files that your application uses. These path names include those to your application's:

Help message AFILE	Use the *Library AFILE Information—Help AFILE* form (menu path 5 3 1 from the Dialog's main menu) to change this path name.
Error message AFILE	Use the *Library AFILE Information—Error AFILE* form (menu path 5 3 2 from the Dialog's main menu) to change this path name.
Default printer	Use the *Global Information—Printer* form (menu path 5 2 1 from the Dialog's main menu) to change this path name.

When you change these path names, be sure they are compatible with your operating system. The Dialog does not validate these path names. If they are wrong, you will receive an error message when you run the application.

The ALLY environment variable names are the same for both operating systems. For example, the ALLY environment variable ({ally}) tells ALLY where to find its files. When you transport an application to a different access method, you may have to define new values for the environment variables that allow ALLY to find the access method's files. Refer to the access method's developer notes to find more information about environment variables.

Access Method Issues

You should read this section if you are transporting an ALLY application from one access method to another.

In this section, we describe some of the access-method differences that affect the way that you build and transport ALLY applications. We do not discuss specific differences, such as differences in data types and ranges within data types. For specific information, refer to each access method's documentation and developer notes.

You can transport ALLY applications among the access methods that ALLY supports. These access methods are:

- C-ISAM
- dBASE III
- FX (fixed sequential)
- ORACLE
- UNIFY

As a developer, you must consider the differences among the access methods when you are developing applications that you want to transport.

The access methods that ALLY supports provide a wide range of data management features. Some access methods, such as ORACLE and UNIFY, are relational. C-ISAM uses indexed sequential files. dBASE III is a file-oriented relational system. The ALLY implementation of dBASE III files supports a single index (sort key) per file. The ALLY FX access method is a simple file management system for prototyping applications. It has no sorting facility.

General Design Considerations

When you are designing an application that will run on different access methods, you must examine the features of each access method to determine if one access method is more restrictive than another. Then, you can design the application so that it can be transported easily to the more restrictive access method. In some cases, this requires that you not take full advantage of the features of a less restrictive access method.

If you design your application to take advantage of a less restrictive access method's features, you may have to modify the application extensively when you transport it to a more restrictive access method—or one that does not provide the same set of features. And, it is possible that you will not be able to provide the same user-visible functions because of the differences in access methods.

Table 2 provides a summary of the access method differences that are important to you as an application developer. In the next sections, we talk more about these differences.

Table 2. Access Method Differences

Characteristic	C-ISAM	dBASE III	FX	ORACLE	UNIFY
Support for sorting?	Yes–with sort keys implemented by View Definitions.	Yes–with an index defined at the Base DSD level.	No	Yes– with "Order By" SQL statement.	Yes– with sort keys implemented by View Definitions.
Fields per sort key.	8	1	N/A	N/A	8
Default file or table naming conventions.	Tables have a ".dat" extension; indexes have an ".idx" extension.	Files have a ".dbf" extension; indexes have an ".ndx" extension.	Files have an ".fx" extension.	N/A–you cannot create tables from the Dialog.	N/A–you cannot create tables from the Dialog.
Build DSDs manually?	Yes	Yes	Yes	Yes	No
Build DSDs from an existing table?	No	Yes	No	Yes	Yes
Automatically change Base DSDs with the Dialog's *Change Base DSD Characteristics–DSD Type* form?	Yes	Yes	Yes	Yes	No–you must build new Base DSDs.

F002-0586-00

Differences in Sorting Techniques

The most important question that you must answer when you are designing an application that will be transported among access methods is, "How do the access methods sort data?"

As shown in Table 2, ALLY applications that use C-ISAM and UNIFY sort data by implementing Base DSD sort keys from View Definitions. Applications that use dBASE III sort data by implementing an index defined in a Base DSD. An application that runs on ORACLE sorts data by implementing an ORDER BY clause in a select statement. The FX prototyping access method does not support sorting.

These different sorting techniques affect the way that you design and build applications that you are going to transport. Let's assume that you are building an ORACLE application that you will transport to one of the other access methods that ALLY supports. ORACLE allows you to sort data by specifying SQL statements when you build the application's Base DSDs. The other access methods do not support this technique.

You should consider building View Definitions into your ORACLE application so that you do not have to build the View Definitions when you transport the application. Instead, you need only add sort keys at the View Definition level after you have transported the application. This is generally much easier than building View Definitions and connecting forms/reports to them after you have transported an application.

Let's look more closely at some of the issues you must consider when you are transporting applications to and from an access method.

C-ISAM Considerations

C-ISAM is a file management system that uses indexed-sequential files. Access to C-ISAM files from ALLY applications is supported on computers that use the UNIX operating system. The ALLY execution system includes a runtime version of the C-ISAM file management system.

The ALLY interface to C-ISAM supports sorting by using View Definitions that implement sort keys. When you transport an application to C-ISAM from an access method that uses this same sorting technique (e.g., UNIFY), you need only change the Base DSD types. The existing View Definitions will implement the sort keys properly.

dBASE III Considerations

ALLY allows you to build applications that work with dBASE III files. You need not have dBASE III running on your system. With ALLY, access to dBASE III files is available on computers that use the MS-DOS operating system. The ALLY interface to dBASE III files supports a single index (sort key) for each file.

The ALLY interface to dBASE III supports sorting by implementing an index that you define for a Base DSD field. When you transport to dBASE III an application that sorts data, you must consider the following issues:

- The ALLY implementation of dBASE III files does not support descending sorts.
- If you are transporting your application from C-ISAM or UNIFY, you can define the index by using the same key that was used as the sort key in the host application. However, you must edit the Base DSD to specify the name of the index file.
- If you are transporting your application from ORACLE, you will probably have to create a dBASE III key, because ORACLE applications use the ORDER BY clause in select statements instead of keys.
- If the host application requires multiple sorts, you should select the most important sort key to use as a dBASE III file's index.

FX Considerations

The ALLY Fixed Sequential (FX) access method is available on any system that supports ALLY. However, it is the most restrictive of the access methods and should be used only to prototype and test applications.

The FX access method does not support sorting. Therefore, you will not be able to maintain the same user-visible interface if the application you transport to the FX access method provides sorted data in reports.

ORACLE Considerations

ORACLE is a fully relational access method that accepts SQL statements from an ALLY application with its Host Language Interface (HLI). ALLY allows you to sort data and generate queries by passing SQL statements to ORACLE, through the HLI. This feature is not supported by the other access methods that ALLY uses.

Sorting

When you build an ALLY application that uses the ORACLE access method, you sort data by using an ORDER BY clause in an SQL select statement. The select statement is contained in the application's Base DSD.

Let's assume that your application uses the ORDER BY clause in a select statement to sort data in an ORACLE application. Then suppose that you want to transport this application to a different access method. To maintain the application's sorting functions with a different access method, you must change the application's Base DSD types and add the appropriate sort keys. In some cases, you will have to build View Definitions to implement these sort keys.

You should consider connecting your forms/reports to View Definitions in your ORACLE application, even though the View Definitions are not used when you run the application with ORACLE. This will make moving the application to another access method easier, because the View Definitions and form/report connections to the View Definitions will already be in place.

If an application that you are transporting to ORACLE was designed for an access method that uses sort keys (e.g., C-ISAM, dBASE III, or UNIFY), you must implement the sort operations with an ORDER BY clause. You must create multiple ORACLE Base DSDs to support the situation in which your incoming application uses View Definitions to sort data from a single Base DSD in different ways. This is because each sorting operation requires a different select statement, and each ORACLE Base DSD supports only one select statement.

Interactive Queries

In an ALLY application that uses ORACLE, you can allow users to invoke the 'query by where' command from an ALLY form/report. By using this command, users can make ad-hoc queries. You may not want to use this feature in an application that you are going to transport to another access method. Access methods other than ORACLE support only the ALLY 'query by example' command, with equality-matching only.

Committing Data

The Dialog's default for the Base DSD option "Record commits not automatic" is on for ORACLE applications. In the other access methods that ALLY supports, the option's default is off. This can cause an application to behave differently when you transport it to ORACLE from another access method.

When this option is off, an application's access method is updated whenever a user moves the cursor from a changed record. When the option is on (the default for ORACLE), the access method is updated only when it receives a 'commit' command from the application. If a user makes several changes to a form/report, then invokes the 'abort action' command, none of the changes will be written to the access method.

You can change this option's setting from the *ORACLE Base DSD—Characteristics* form's *Options Inheritable by a Form/Report* subform (menu path 3 1 2 < > 1 from the Dialog's main menu).

UNIFY Considerations

UNIFY is a relational access method that stores data in tables. You can create UNIFY Base DSDs only by reading an existing UNIFY table.

When you transport an application to the UNIFY access method, you cannot use the Dialog's *Base DSD Characteristics—DSD Type* form to change from another access method's Base DSD type to the UNIFY Base DSD type. Instead, you must rebuild the application's Base DSDs and copy them over the existing Base DSDs. When you rebuild the Base DSDs, you must also create any associated sort keys and foreign key links.

ALLY Revision Differences

Applications built with the 2.+ version of ALLY are upward and downward compatible within the 2.+ product line.

You can upgrade and transport an application developed on a 1.2+ version of ALLY to run on a 2.+ version of ALLY. To do that, you must have both versions of ALLY running on either the host machine or the target machine. If you create a transportable 1.2+ AFILE on the host machine, you must reconstruct it with a 1.2+ ALLY system on the target machine before you can upgrade the old AFILE to a newer version. Or, you can upgrade the old AFILE on the host machine and then create a transportable 2.+ AFILE. When you upgrade a 1.2+ AFILE, you will not be able to execute it on a 1.2+ ALLY system.

The 1.2+ version of ALLY stored a copy of the ALLY Command Menus in an application's AFILEs. The 2.+ version of ALLY stores Command Menus in the ALLY system's error AFILE (errors.e). You must have the proper path name to the error AFILE to allow the application to find the ALLY Command Menus.

Error AFILEs can be chained so that one error AFILE points to another. Your application's error AFILE can be anywhere in a chain. See the *System Manager's Guide* (UP-13765) for more information about chaining error AFILEs.

Migrating an Application—An Example

In this example, we show you how to transport the application that is described in the *Tracking Employee Hours Application Storybook* (UP-12502). This application, which is provided with the ALLY release software, uses the ALLY fixed-sequential (FX) access method. This is the access method that allows you to prototype applications on any system that runs ALLY.

We assume that you are transporting the application AFILE and its data files from a UNIX system (the host system) to a Unisys PC/IT (or compatible) MS-DOS system with the dBASE III access method (the target system).

Getting Started

Before you transport this application, you must answer the questions that we provide in the "Before You Start" section.

Are you transporting the application to a different operating system?	Yes—from UNIX to MS-DOS. You will change the path names of the application's external help AFILE, error AFILE, and default printer.
Are you transporting the application to a different access method?	Yes—from FX to dBASE III. You will change the type of the application's Base DSDs.
What are the names of the application's AFILEs, Base DSDs, and macro-command files?	The application's AFILE is "HOURS.A." There are two Base DSDs, "HOURS" and "EMPLOYEE." There are no macro-command files.

Are you updating the application to run on a newer version of ALLY?	No. You will move the application from ALLY 2.0 on the UNIX system to ALLY 2.0 on the MS-DOS system.
Does the application use any external programs?	No.

After you have answered these questions, you are ready to transport the application. We suggest that you actually follow these steps to transport the application.

On the Host System

Here is a summary of the steps that you will use to prepare the application's AFILE and data files on the host system. You will invoke both utilities from the Dialog, though you could invoke them from the operating system if you wanted to.

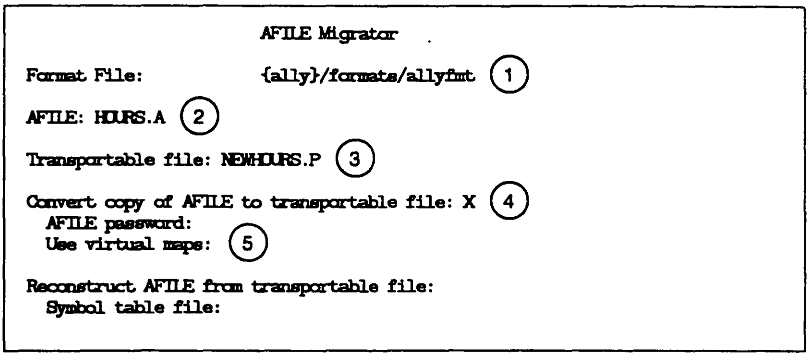
- Use the AFILE Migrator to construct a transportable version of the "HOURS.A" AFILE.
- Use the Data Migrator to construct transportable text files for the "HOURS" and "EMPLOYEE" data files.
- Move the transportable files to the target system, making sure that you do not overwrite any existing files.

Let's look at these steps in more detail.

Creating a Transportable AFILE

Menu path: **5 5 1 4** from the Dialog's main menu
Form name: *AFILE Migrator*

The first step is to construct a transportable version of the "HOURS.A" AFILE. To do that, invoke the Dialog on the "HOURS.A" AFILE and go to the *AFILE Migrator* form. Figure 5 shows you how the *AFILE Migrator* form will look after you have filled it out.



```
                         AFILE Migrator

Format File:             {ally}/formats/allyfmt

AFILE: HOURS.A

Transportable file: NEWHOURS.P

Convert copy of AFILE to transportable file: X
  AFILE password:
  Use virtual maps:

Reconstruct AFILE from transportable file:
  Symbol table file:
```

Figure 5. Invoking the AFILE Migrator

Let's look more closely at what you enter into these fields:

1) The name of the current Format File is filled in for you. Type <Return> to use this file, or change the name to use a different Format File.

2) Enter the name of the AFILE that you want to convert (HOURS.A).

3) Enter the name of the transportable file that you want to construct. The name that we use in this example is "NEWHOURS.P," where "P" stands for "portable AFILE." You could use any name that you want, as long as it is valid on the target operating system. We are using "NEWHOURS" so that we do not overwrite any "HOURS" files that might be stored on the target system.

4) Place an "X" in the "Convert copy of AFILE to transportable file" field.

5) Leave the "AFILE password" and "Use virtual maps" fields blank. The "HOURS.A" AFILE is not password protected, and you are not working on a limited-memory machine.

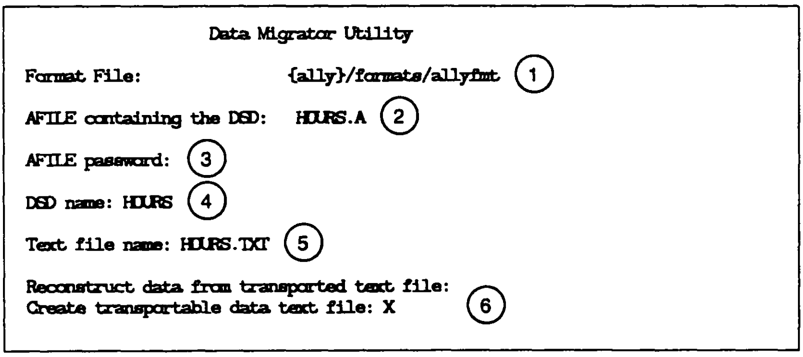
After you enter this information, invoke the 'exit action' command to construct the "NEWHOURS.P" file. The cursor moves to the lower-left corner of the form while the AFILE Migrator runs. After processing is complete, the cursor moves back to the menu titled *AFILE Utilities*.

Creating Transportable Data Files

Menu path: **5 5 2 3** from the Dialog's main menu
Form name: *Data Migrator Utility*

Now use the Data Migrator to construct transportable versions of the data files that are referenced by the application's "HOURS" and "EMPLOYEE" Base DSDs.

Figure 6 shows you how the *Data Migrator Utility* form will look after you have filled it out to migrate the "HOURS" data file.



```
                    Data Migrator Utility

Format File:                {ally}/formats/allyfmt  (1)

AFILE containing the DSD:  HOURS.A (2)

AFILE password: (3)

DSD name: HOURS (4)

Text file name: HOURS.TXT (5)

Reconstruct data from transported text file:
Create transportable data text file: X         (6)
```

Figure 6. Invoking the Data Migrator

Let's look more closely at what you enter into these fields:

1) The name of the current Format File is filled in for you. Type <Return> to use this file, or change the name to use a different Format File.
2) Enter the name of the AFILE that contains the DSD for the data you are migrating (HOURS.A).
3) Leave the password field blank because the HOURS.A AFILE is not password-protected.
4) Enter the name of the DSD for the data you are migrating (HOURS).

5) Enter the name of the text file that you want to construct ("HOURS.TXT," which is a valid file name on MS-DOS systems).

6) Leave the "Reconstruct data from transported text file" field blank, and put an "X" in the "Create transportable data text file" field.

After you enter this information, invoke the 'exit action' command to construct the "HOURS.TXT" file. The cursor moves to the lower-left corner of the form while the Data Migrator runs. After processing is completed, the cursor moves back to the menu titled *Auxiliary Utilities*.

You must now repeat this step to construct a portable copy of the data that is referenced by the "EMPLOYEE" Base DSD. Return to the *Data Migrator Utility* form and enter "EMPLOYEE" for the DSD name and "EMPLOYEE.TXT" for the text file name.

Moving the Transportable Files

The final step on the host system is to move the transportable files to the target system. The transportable files contain ASCII text and require no special communication protocol.

Note that different operating systems store files in different ways. When you transport this application's files to the Unisys PC/IT, you will see that the file sizes are larger. This is because MS-DOS appends a carriage return at the end of each line in a text file.

On the Unisys PC/IT Target System

Here are the steps that you use to reconstruct the application's AFILE and data files on the target Unisys PC/IT. You cannot run the ALLY utilities from within the Dialog on the PC version of ALLY. Therefore, you will run the AFILE Migrator and Data Migrator from the operating system. The detailed steps follow this list.

- Place the transportable files into a directory that is different from the one that contains the ALLY software.
- Use the AFILE Migrator to reconstruct the "HOURS.A" AFILE and rename it "NEWHOURS.A." You must complete this step first, because some of the other steps require information from this AFILE.
- Invoke the Dialog to change the application's Base DSD types to dBASE III. You must do this step before you reconstruct the application's data files.
- While you are in the Dialog, change the path names of the application's help and error message AFILEs and default printer.
- Exit from the Dialog and use the Data Migrator to reconstruct the application's "HOURS" and "EMPLOYEE" data files.

The following section provides the details of these steps.

Reconstructing the AFILE

You will use the AFILE Migrator to reconstruct the application's "HOURS.A" AFILE and rename it "NEWHOURS.A." Use the following command to invoke the AFILE Migrator:

```
amigrate NEWHOURS.P NEWHOURS.A none r<Return>
```

Here is an explanation of this command line's arguments:

NEWHOURS.P	The name of the transportable AFILE that you constructed on the host system.
NEWHOURS.A	The name of the AFILE that you want to reconstruct. We are renaming the AFILE to avoid writing over the "HOURS.A" AFILE that is supplied with the ALLY software.
none	The argument that tells the AFILE Migrator that you do not want "NEWHOURS.A" to have an external symbol table. If you wanted to construct an external symbol table, you would enter its name here.
r	The argument that tells the AFILE Migrator that you want it to reconstruct an AFILE from a transported file.

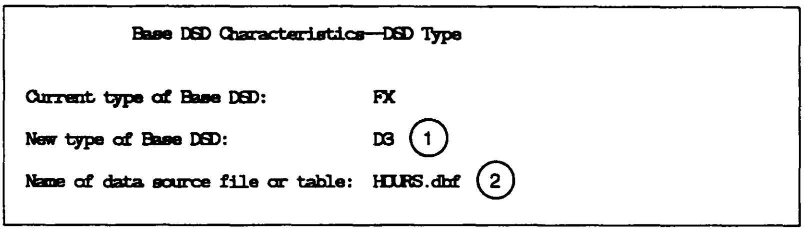
When you invoke the AFILE Migrator with this command line, it constructs "NEWHOURS.A." The size of "NEWHOURS.A" will be different from the "HOURS.A" file on the host UNIX system because of differences in the way that the two operating systems store AFILEs.

Changing the Base DSD Types

Menu path: **3 1 2** < > **6** from the Dialog's main menu
Form name: *Base DSD Characteristics—DSD Type*

You will use this Dialog form to change the types of the application's two Base DSDs to "D3" (dBASE III).

Figure 7 shows how the *Base DSD Characteristics—DSD Type* form must look after you have filled in the fields to change the type of the "HOURS" Base DSD. Note that the "Current type of Base DSD" field is already filled in for you.



```
          Base DSD Characteristics—DSD Type

Current type of Base DSD:            FX
New type of Base DSD:                D3 (1)
Name of data source file or table:   HOURS.dbf (2)
```

Figure 7. Changing a Base DSD Type

Let's look more closely at what you enter into these fields.

1) Enter the name of the new type of the Base DSD. In this case, you will enter "D3" to change the type to dBASE III.

2) Enter the name of the data file or table that contains the data that the Base DSD describes. You must enter "HOURS.dbf." The ".dbf" extension is a dBASE III naming convention that allows ALLY to recognize the file as a dBASE III data file.

When you have filled in the form, invoke the 'exit action' command to change the "HOURS" Base DSD type. The Dialog will return you to the *Base Data Source Definition Characteristics* menu.

You must now change the type of the "EMPLOYEE" Base DSD. Return to the *Base DSD Characteristics—DSD Type* form (menu path 3 1 2 <EMPLOYEE> 6) from the Dialog's main menu.

In the *Base DSD Characteristics—DSD Type* form, change the "EMPLOYEE" Base DSD's type to "D3" and name the file "EMPLOYEE.dbf."

Reconstructing the Data Files

You will use the Data Migrator to reconstruct the application's two data files. Invoke the Data Migrator with the following command line:

dmigrate NEWHOURS.A none HOURS HOURS.TXT r <Return>

Here is an explanation of this command line's arguments:

NEWHOURS.A	The name of the AFILE that contains the Base DSD that describes the file's data.
none	The AFILE's password, which is "none," because "NEWHOURS.A" is not password-protected. By default, an AFILE is not password-protected.
HOURS	The name of the Base DSD that references the external data file.
HOURS.TXT	The name of the text description file that the Data Migrator reads to reconstruct the data file.
r	The argument that tells the Data Migrator that you want to reconstruct data from the transported "HOURS.TXT" file and place it into the "HOURS.dbf" file.

When you invoke the Data Migrator with this command line, it constructs the "HOURS.dbf" dBASE III data file.

You must now repeat this step, substituting "EMPLOYEE" for the "HOURS" DSD name and "EMPLOYEE.TXT" for the "HOURS.TXT" text file name. This will construct the "EMPLOYEE.dbf" dBASE III data file.

Changing Path Names of External Files

Because you are transporting this application from a UNIX system to an MS-DOS system, you must change the path names that ALLY uses to find the different parts of the application. You must make these changes because of the different syntax that the operating systems use in their path names. Also, ALLY is installed in a single directory on the PC, instead of in a hierarchy of directories, as on a UNIX system. (We describe these differences in Table 1 of the "Operating System Differences" section.)

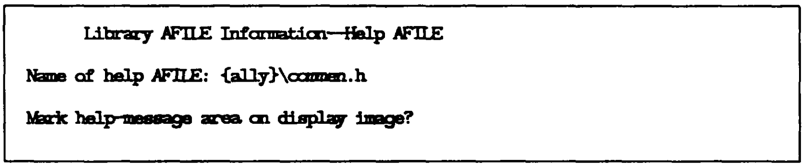
The path names that you will change are to the application's:

- help messages
- error messages
- default printer

Changing the Path Name of the Help Message File

Menu path: **5 3 1** from the Dialog's main menu
Form name: *Library AFILE Information—Help AFILE*

Figure 8 shows how this form must look after you change the path name of the application's help AFILE from the UNIX syntax to the MS-DOS syntax.



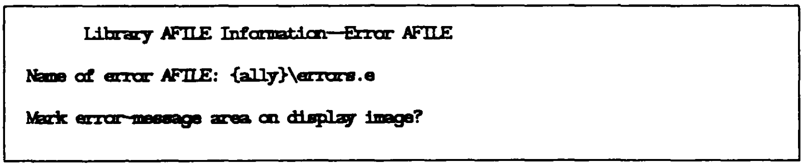
```
      Library AFILE Information—Help AFILE

 Name of help AFILE: {ally}\common.h

 Mark help-message area on display image?
```

Figure 8. Library AFILE Information—Help AFILE

Changing the Path Name of the Error Message File

Menu path: **5 3 2** from the Dialog's main menu
Form name: *Library AFILE Information—Error AFILE*

Figure 9 shows how this form must look after you change the path name of the application's error AFILE from the UNIX syntax to the MS-DOS syntax.



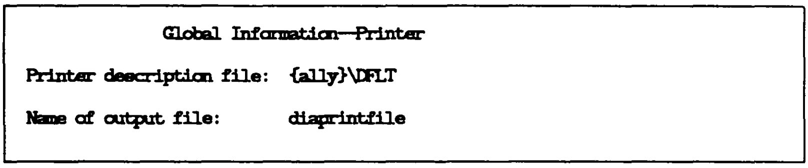
```
      Library AFILE Information—Error AFILE

 Name of error AFILE: {ally}\errors.e

 Mark error-message area on display image?
```

Figure 9. Library AFILE Information—Error AFILE

Changing the Path Name of the Default Printer

Menu path: **5 2 1** from the Dialog's main menu
Form name: *Global Information—Printer*

Figure 10 shows how the *Global Information—Printer* form must look after you change the path name of the application's default printer.



```
              Global Information—Printer

Printer description file:  {ally}\DFLT

Name of output file:       diaprintfile
```

Figure 10. Global Information—Printer

Notice that the output file name (diaprintfile) is longer than the eight characters allowed by MS-DOS. You need not change this name because the operating system will truncate it to "diaprint."

Testing the Application

After you complete these steps, you should test the application to be sure that the AFILE and data files were transported and reconstructed successfully. To run the application on the Unisys PC/IT, go to the directory that contains the application and type:

ally NEWHOURS.A

The "tracking employee hours" application's main menu will appear on your display. The sample of this application, released with the ALLY software, has in it several records. Display the "Weekly Hours Report" (choice 2 from the main menu). Then type <Return> twice to display all of the sample application's records. The report looks like the sample in Figure 11.

```
                                                        01/11/88  Page     1
                                  Weekly Hours Report

Week Ending   Name              Id       Regular Hours  Vacation  Sick  Holiday
01/03/88      Sarah Chalmers    00001          40          0        0      0
01/10/88      Sarah Chalmers    00001          32          8        0      0
01/03/88      Jonathan Meyer    00002          32          0        8      0
01/10/88      Jonathan Meyer    00002          40          0        0      0
01/03/88      Peter Wagner      00005          40          0        0      0
01/10/88      Peter Wagner      00005          40          0        0      0
```

Figure 11. Weekly Hours Report

You can also display the "Total Hours Report" (choice 3 from the application's main menu) and type <Return> to display its records. Figure 12 shows that report with its sample records.

```
                                                        01/11/88  Page     1
                                  Total Hours Report

Name                  Id
Sarah Chalmers        00001

Week Ending   Regular Hours  Vacation  Sick   Holiday
01/03/88         40             0        0       0
01/10/88         72             8        0       0

Jonathan Meyer        00002

Week Ending   Regular Hours  Vacation  Sick   Holiday
01/03/88         32             0        8       0
01/10/88         72             0        8       0

Peter Wagner          00005

Week Ending   Regular Hours  Vacation  Sick   Holiday
01/03/88         40             0        0       0
01/10/88         80             0        0       0
```

Figure 12. Total Hours Report

Summary

In these developer notes we have described the differences among operating systems and access methods that affect how you transport ALLY applications. We also described some of the decisions that you should make when you design an ALLY application that you will transport among different access methods.

End of Application Migration Developer Notes

Index

End of Index