

Porting COBOL and Powerhouse Applications to AIX/Oracle: Issues and Answers

**By Charles H. Finley, Jr.
Transformix Computer Corporation**

Introduction

The MPE environment contains a substantial amount of unique functionality. Unfortunately, this uniqueness can make applications that must be migrated off of the HP 3000 that much more difficult to port. The SunGard Bi-Tech Transport Migration Toolkit (Transport) provides substitute functionality for porting HP 3000 applications to Transport supported target platforms (Linux, UNIX, Windows or the IBM iSeries). This is supplemented by Transformix Tools from Transformix Computer Corporation. These tools enable applications written in HP 3000 supported third-generation languages such as COBOL to be ported with minimal manual changes. This method of porting, however, implies that the MPEism's will remain in the ported programs. For, example, intrinsic calls to the MPE file system are retained intact and the Transport run-time environment provides the expected MPE-like interface to files.

Migrating applications written in 4gl's such as Cognos Powerhouse can usually be ported to the same target environments with relatively little difficulty. However, these migration solutions are typically designed to run "natively" on the target platforms. This means that the Powerhouse program will expect a target environment file system instead of an MPE-like interface to files.

Applications that use both COBOL and Powerhouse can present a special migration challenge. For example, if the COBOL programs expects file A to be an MPE-like file and the Powerhouse program expects to access the same file A as using native UNIX calls, this can present a problem. The problem is due to the reality that file A may not be a standard UNIX file in all cases. MPE files usually have file labels. If the file has a label, on MPE, Transport takes a standard UNIX file and adds one record at the beginning with "label" information. Although this file may be readable by the Powerhouse program, the first record could cause problems in processing unless the Powerhouse program logic is changed to ignore it. There are many other opportunities for conflict as well.

Migrating MPE applications programs to any target platform requires identifying both the implicit and explicit references to intrinsics. In Appendix A, there are tables containing lists of the intrinsics that are used in the Bank of Austria Custody application to perform various functions, including accessing files, accessing features of the command interpreter, accessing TurboIMAGE databases, and using VPlus forms. If an application

program contains references to these intrinsics, it's very likely that the program logic also assumes certain proprietary characteristics of the MPE operating system.

This white paper will focus specifically on identifying the proprietary characteristics of MPE that are in use at the Bank of Austria by studying the use of Intrinsics. It will describe the proposed Transport/Transformix solution and it will describe how both the COBOL and Powerhouse applications will function in the target AIX/Oracle environment.

Solution Components and Overview

Although MPE Intrinsics use at the Bank of Austria does not tell the whole migration story, the COBOL/Powerhouse integration reality makes it more complex than many other migration scenarios. For this reason we have chosen to present the recommended solution as a separate document from the primary Assessment. Figure 1 provides an overview of the MPE Intrinsics Solution Components used by IBM/AMS in the proposed Bank of Austria migration.

The primary areas of interest are:

- Transport TurboIMAGE Migration
- Files and Related Concepts Migration
- VPlus Migration
- Other Intrinsics Migration

	COBOL/ VPlus	Powerhouse Quick	Powerhouse Quiz	Powerhouse QTP	SPL/C
TURBOIMAGE – database access	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer
TURBOIMAGE- locks	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer
TURBOIMAGE- sub-items	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer
TURBOIMAGE- Transactions	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer	Transport TurboIMAGE Layer
Files Systems: MPE Flat Files	Transport MPEIO	UNIX File I/O	UNIX File I/O	UNIX File I/O	Transport MPEIO
MPE KSAM	Transport MPEIO,	CISAM	CISAM	CISAM	Transport MPEIO,

	CISAM				CISAM
MPE Message Files	Transport MPEIO	UNIX File I/O, Transformix VFS	UNIX File I/O, Transformix VFS	UNIX File I/O, Transformix VFS	Transport MPEIO
MPE Circular Files	Transport MPEIO	UNIX File I/O, Transformix VFS	UNIX File I/O, Transformix VFS	UNIX File I/O, Transformix VFS	Transport MPEIO
MPE File Equations	Transport Shell	Transport Managed Environment Variables	Transport Managed Environment Variables	Transport Managed Environment Variables	Transport Shell
MPE File Labels	Not used	Not used	Not used	Not used	Not used
COMMAND/SYSTEM	Transport Intrinsic	Unix SYSTEM CALL/Transport	Unix SYSTEM CALL/Transport	Unix SYSTEM CALL/Transport	Transport Intrinsic
Environment Variables	Transport Intrinsic	Unix /Transport	Unix Transport	Unix /Transport	Transport Intrinsic
VPlus	Transport Intrinsic	N/A	N/A	N/A	N/A
Other MPE-Intrinsic	Transport Intrinsic	N/A	N/A	N/A	Transport Intrinsic

Figure 1 MPE Intrinsic Solution Components

TurboIMAGE

According to the *TurboIMAGE/XL Database Management System Reference Manual* “TurboIMAGE/XL is a set of programs and procedures that you can use to define, create, access, and maintain a database. A database is a collection of logically-related files containing both data and structural information. Pointers within the database allow you to gain access to related data and to index data across files.” It is the primary data management mechanism used to build applications on the HP 3000.

TURBOIMAGE – database access

Database access using TurboIMAGE is done using the TurboIMAGE API. The solution uses The Transport TurboIMAGE Layer provides a Cognos recognized, AIX version of this API that works with Oracle.

TURBOIMAGE- locks

TurboIMAGE provides a locking is unique. It includes a concept called predicate locking. A predicate lock is a way of establishing a lock without locking a physical record. It also includes the ability to the entire database, a specific data set, or a specific set of entries within a data set with a particular data item value. This is different than a RDBMS locking mechanisms. The Transport Database Layer provides TurboIMAGE compatible locking.

TURBOIMAGE- sub-items

TurboIMAGE supports a data type called a sub-item-count is an integer from 1 to 255 that denotes the number of sub-items within an item. If omitted, the sub-item-count equals one by default. A data item whose sub-item count is 1 is a simple item. If the sub-item count is greater than 1, it is a compound item. We might also call it an array or a table of values. These compound items are not supported by RDBMS's. The Transport Database Layer provides TurboIMAGE sub-items for supported RDBMS's. This makes it possible to migrate COBOL and Powerhouse applications with few if any logic changes.

TURBOIMAGE-Transactions

TurboIMAGE supports a unique transaction handling mechanism. The Transport Database Layer provides TurboIMAGE transactions.

Files and Related Concepts

MPE Flat Files

MPE Flat Files have unique characteristics. The Transport File System Layer provides the needed capability to COBOL applications while at the same time allowing Powerhouse programs to access the same files through the AIX API.

MPE KSAM

MPE KSAM Files have unique characteristics. The Transport File System Layer provides the needed capability to COBOL applications while at the same time allowing Powerhouse programs to access the same files through the C-ISAM API.

MPE Message Files

MPE Message Files have unique characteristics. The Transport File System Layer provides the needed capability to COBOL applications while at the same time allowing Powerhouse programs to access the same files through the AIX API using a Virtual File System mechanism similar to NFS.

MPE Circular Files

MPE Circular Files have unique characteristics. The Transport File System Layer provides the needed capability to COBOL applications while at the same time allowing Powerhouse programs to access the same files through the AIX API using a Virtual File System mechanism similar to NFS.

MPE File Equations

MPE File Equations are a mechanism that many MPE programs rely on. The Transport File System Layer provides the needed capability to COBOL applications while at the same time allowing Powerhouse programs to access the same files through the AIX environment variables.

MPE File Labels

MPE File Equations are not used on the Custody files because of their incompatibility with Powerhouse. They are not needed.

COMMAND/SYSTEM

An MPE program can issue an MPE command while it is running using the Command Intrinsic call. The UNIX SYSTEM call is similar. COBOL programs will use Command while Powerhouse programs will use SYSTEM.

Environment Variables

Environment variables for the MPE-like COBOL programs and the AIX-like Powerhouse programs will work in much the same way. There is no need for any engineering to make this happen.

VPlus

COBOL programs will use the Transport provided VPlus Ininsics. Powerhouse programs have their own screen handler.

Other Ininsics

An MPE program can issue an MPE command while it is running using the Command Intrinsic call. The UNIX SYSTEM call is similar. COBOL programs will use Command while Powerhouse programs will use SYSTEM.

Transport TurboIMAGE Migration

Migrating MPE applications generally involves migrating IMAGE databases. The Transport TurboIMAGE Migration Toolset is designed to replace IMAGE with a Relational Database Management System. The migration of IMAGE data to relational database management systems (RDBMS) and/or support of IMAGE intrinsics on the target platform can present a major technical challenge. IMAGE is a network, or a limited-hierarchical, database management system. Information is stored on two levels and is accessed by traversing a path from the highest level, or hierarchy, of the network to the desired lower level. The two levels of hierarchy in IMAGE are the master and detail level. At the master level, master datasets keep information about a uniquely identifiable entry. At the detail level, detail datasets keep detailed information related to the unique entries in the top level. Detail datasets are also used to relate the entries in the master level to each other.

In the relational model data is organized into tables. A table is a two-dimensional structure of columns and rows. A row is similar to an entry in an IMAGE dataset and a column is similar to a data item or field. Access to this data is facilitated through Structured Query Language (SQL). Examples of relational database management systems include Oracle, Informix, DB2, and SQL Server.

Carrying out database I/O to these relational engines from a procedural language like COBOL can be accomplished through Call Interfaces. A program using the call interface can control the actual steps of SQL statement processing (parse, bind, execute, rebind, reexecute, and so on) to achieve optimum application performance.

The ideal call interface to effect a quick and efficient migration is one that allows the existing COBOL program to make IMAGE calls as it has always done and to translate these into the call interface supported by the target RDBMS. The call interface of the target RDBMS gives the most control and highest performance available and is the preferred mechanism when creating a call-compatible library of IMAGE intrinsics. This is done by creating a library of C routines which includes procedures that emulate the functionality of each IMAGE intrinsic and that translate the IMAGE request into a series of native RDBMS database calls. This library can be either an archive or shared library. The library is then linked into the target COBOL run-time module to support the INTRINSIC calls and provide a call-compatible interface. Properly designing a call-compatible interface library can be challenging. Challenges include support of a backward chained or serial read when the RDBMS does not support FETCH previous. It is also necessary to adapt and enhance the ROWID address to support directed reads. The benefit of an automated migration that uses an IMAGE call-compatible interface is that either no changes need to be made to program sources or, if any are required, they are few.

Transport TurboIMAGE Database Migration Tools

Actually migrating an IMAGE database is a multi-part process. First, the IMAGE schemas are analyzed programmatically to extract dataset names, data item names, compound items, data types, key items, search items, sort items, and paths. The analysis program produces SQL CREATE table scripts for each IMAGE master and detail set (automatic master sets as a physical table are not required, although their indexing function will be preserved). These SQL scripts can then be executed to produce empty tables ready to be populated with data.

The xschema processor makes various changes that are needed in order to map IMAGE data to the target RDBMS. For example, in Oracle, names can contain only alphanumeric characters from the database character set and the characters '_', '\$', and '#'. The dash character is therefore mapped to the underscore character. Compound items are a feature of IMAGE that do not have an equivalent in the target RDBMS. Multiple unique column names are created to support these items. Naming of schema objects has some restrictions in the target RDBMS.

Migration of IMAGE data involves exporting master and detail sets to sequential files using third-party utilities or custom export programs. These files are then transferred to the target UNIX machine, and an automated tool uses the native RDBMS loader utility to populate the empty relational tables with this data.

It is important to note that the preceding process does not result in any normalization of the data. Relational database systems are based largely on mathematical relational theory. Designing a relational database involves the process of normalization. The theory of normalization is a key element of relational database design. In simple terms, normalization is the process of decomposing and arranging the attributes in a schema, which results in a set of tables with very simple structures. The purpose of normalization is to make tables as simple as possible and to reduce the redundancy of information within a database. When done properly, the normalization process will result in well-designed tables, with reduced storage of redundant data and the increased ability to effectively enforce integrity constraints.

This normalization process, if attempted on an existing IMAGE database, obviously would involve major reengineering of the application and would thus be impractical. The inherent network structure and key/search item construct of IMAGE, by its very nature, produces some data redundancy. The lack of normalization can be perceived as a drawback, but it is perfectly acceptable to have migrated IMAGE data into a non-normalized relational form for an intermediate period of time. Once the migration is complete, there are no restrictions on normalizing at a managed pace appropriate for the application and future development.

One of the advantages of migrating data into a RDBMS immediately even while retaining the IMAGE API is that it allows the user to take advantage of new functionality sooner. The flexibility of SQL allows for the ad hoc creation of indices to improve performance, as well as data structures, such as views and synonyms, to mask or massage the data structure without actually changing the table structure itself. The application developer is able to write new applications with the productivity tool sets that are available in the target environment and can then use the power of distributed transaction processing, client-server configurations, and full application portability.

With an IMAGE wrapper library and corresponding utilities, the first step in the conversion from TurboIMAGE to an RDBMS is automatic. However, it must be emphasized that the first step is exactly that; a first step. The project is not done when the automated part of the conversion is complete. In most IMAGE wrapper conversions, some modifications to the code must be done to address specific performance or functionality issues that may appear after the initial migration is complete.

In order to work around the differences between TurboIMAGE and the target RDBMS, specialized programming tools and techniques may be used to exercise additional controls over the RDBMS. For example, wrapper libraries and tools often add additional hidden fields to an RDBMS when it is built from a TurboIMAGE database. These products do this to support the TurboIMAGE simulation. Because of this, there may be problems with accessing the database through the wrappers and through conventional SQL access at the same time..

In TurboIMAGE, data is stored chronologically, at least by default. This means that when you read data from a TurboIMAGE database, by default you'll access the records in the same order in which they were added to the data- base — first in — first out.

If a TurboIMAGE application depends upon this behavior, then the application must be enhanced before it can be used with an RDBMS. Since an RDBMS does not “serve up” data in chronological sequence, (at least not by default) the wrapper library must do it on behalf of the RDBMS. In this way, the application program can continue to operate as if it were still operating against a TurboIMAGE database.

A common way of doing this is to add a sequence number to the database in order to support the chronological ordering of rows. Each time data is written to the database, the sequence number is incremented by one and written to the same row as the data. If the data is to be retrieved from the database in chronological sequence, the wrapper library will retrieve it and sort it using the sequence numbers before presenting it to the TurboIMAGE application. In this way, the chronological sequence can be preserved.

Transport TurboIMAGE Intrinsic Layer

The Transport TurboIMAGE Intrinsic Layer is the run-time component of the TurboIMAGE Migration Toolkit. This technology makes it possible to modify a TurboIMAGE application to use an RDBMS, and yet still leave all the references to the TurboIMAGE intrinsic interface intact.

When the application calls a TurboIMAGE intrinsic, a software library that translates the TurboIMAGE intrinsic into a corresponding SQL statement intercepts the call. Because the RDBMS is effectively being “wrapped” in a layer of TurboIMAGE software, this software library is sometimes referred to as an “IMAGE Wrapper.”

The Transport TurboIMAGE Layer allows application programs to continue calling TurboIMAGE intrinsics. The migration can be completed much more quickly, because the application programs do not have to be changed to support the RDBMS. All the complexities of accessing the relational database are performed by the wrapper code. Field names are mapped, data types are converted, and locking is automatically handled.

Complexities

The problem arises when another application tries to retrieve the data using conventional SQL. If this second application is not expecting to see the sequence number along with the data, it can result in problems. Users accessing the database without the wrappers must also maintain these hidden fields. Most relational databases have stored procedures and triggers that can help with this.

COBOL Support

Cobol programs that were written to use TurboIMAGE are supported on AIX with a few minor changes. These changes are all made with Transport tools. The programs are compiled with the Microfocus or Acucorp COBOL compiler then linked with the Transport library shared library. It contains the TurboIMAGE emulation routines. At run-time the program behaves in a manner substantially the same as it does on MPE and the HP 3000.

Powerhouse Support – Quick, Quiz, QTP

PowerHouse supports Eloquence from Marxmeier on HP-UX and Windows NT/2000/XP. PowerHouse accesses Eloquence through the Eloquence TurboIMAGE Compatibility Extension and supports those features of Eloquence that make it look like IMAGE on MPE/iX. Transport “pretends” to be Eloquence on AIX and allows run-time powerhouse code to run almost without changes on AIX. Prior to running on AIX the programs and dictionary must be compiled for AIX.

Here is a quote from Bob Deskin, 4GL Product Manager of Cognos:

“... Transformix ... tools work in a similar way to Eloquence in that they provide a library that intercepts IMAGE intrinsic calls and translates them into something else. In the case of Eloquence, the something else is Eloquence calls. In the case of the other vendors, the calls are translated into relational database calls. The library routines handle all of the relational considerations (transactions, commits, etc.) so to the application, the data source looks identical to IMAGE. These tools also provide, as does Eloquence, utilities to convert the IMAGE database to the target data source.

Transformix ... ran their own tests by substituting their libraries for the Eloquence ones. ... Transformix has run tests on HP-UX, Windows, and AIX.

AIX you ask? That wasn't in the Eloquence release list. That's right, but when contacted, we worked with these vendors to see whether we could provide a solution for our customers. We created a special AIX build that included the same IMAGE intrinsic calls that the HP-UX Eloquence version does. And I don't see any reason why we can't do the same on Sun Solaris if needed. The AIX changes will be part of the maintenance release due late this year so that they will be rolled into the production version.

In addition, we continue to work with Transformix to ensure that these IMAGE emulators continue to work with future releases by incorporating tests with their libraries into our own test suites.

That said, we will support third party IMAGE emulation in the same manner as we support ODBC on Windows. We support the standard and we test against SQL Server and SQL Anywhere, but we don't support specific drivers. Likewise, we support native IMAGE and Eloquence. If the IMAGE emulator shows an issue that is duplicated in our

standard, then we will endeavor to fix it. If the issue is unique to the emulator, we will initially look to the emulator. Of course we will work with the emulator vendor to resolve the problem as much as possible.”

File Systems and Related Concepts

There are a number of features of the MPE file system, which are significantly different from the AIX file system. These include:

- MPE’s unique file *naming convention*
- MPE’s three level *directory structure*
- MPE’s support for specialized file *types*
- MPE’s support for specialized file *domains*

File system functionality

The file system is the part of the MPE/iX operating system that manages the transfer of information between application programs running on the HP e3000 and peripheral devices such as disks. It handles input/output operations such as passing of data to and from user processes, compilers, and data management subsystems.

MPE files are record based. This means that the data in each file has a structure, and the operating system is aware of this structure. Application programs are responsible for maintaining this structure, as follows.

1. Application programs writing data arrange it into data elements called fields. Each field is a string of bytes. The fields are grouped together into ***records***. Data is written to the output file one record at a time.
2. An application program reading the data reads it one record at a time. As each record is read, the application program processes its fields.
3. A ***file*** is a group of logically related records, which may be kept on any storage medium or sent to any input/output peripheral device. MPE applications are ***device independent***, a feature, which cannot be taken for granted on other platforms.

Each MPE file is divided into records. The length of each record is user- definable; it may be fixed or variable. That is, each record may have the same length, or the records in a file may vary in length. The file system is aware of this file structure and ensures that data is read and written only in record-sized blocks. If your program tries to write a single byte to an 80-byte record, the file system will write the entire 80 bytes, padding the single data byte with blanks or binary zeroes.

The files on AIX are byte streams. AIX files make no provision for records, as we know them, from MPE/iX. Each file is handled by the operating system as a continuous stream of bytes. By default, when a UNIX application writes a string of bytes to a file, the string is delineated by a “new line” character. Since there is no concept of record length, when a program writes a single byte to the file, the byte is followed by the “new line” character. No padding is done, as would be the case on MPE/iX. This may result in a

more efficient use of storage space, but it can be a headache for programmers trying to port code from MPE/iX to one of the AIXs.

With either Microfocus or Acucobol standard COBOL statements to do disk I/O, the compiler will handle most of these issues. Transport provides a replacement IO module to ensure this. The compilers on the AIX systems understand the use of byte stream files and handle the data appropriately. This makes it comparatively easy to port software that uses standard COBOL I/O statements such as READ and WRITE. However, when porting the *data* from the HP 3000 to your target platform, it is necessary to convert the files to the byte-stream format. There are utilities available to do this. Similarly, if your program contains explicit references to MPE intrinsics, and you choose to use an MPE emulator, the record structures used on MPE will be retained in byte stream files.

On MPE, when an MPE file is created, it must be declared to be either an ASCII file or a BINARY file. If the file is declared to be ASCII, short records will be padded with ASCII blanks (hex “20”s). If it’s declared to be BINARY, short records will be padded with binary zeroes. When a file is produced by through the same type Transport libraries, padding is done. Moreover, on ASCII files we can optionally either use the new line character or not use it.

File sizes

MPE artificially restricts the size of files; this is different from the behavior of the file systems of the AIX. In fact, if you create a file on a typical UNIX system, there is, (by default) no limit to the amount of data that can be written to that file.

On MPE, when you create a file, you must specify a *filesize*. If you fail to specify one, a default value will be assumed. This filesize represents the maximum number of records that can be written to that file. If you create a file on an MPE system, you cannot write more records to the file than the limit specified in the *filesize*.

On the AIXs, limits can be imposed on the amount of data that can be written to a file (or files), but the limits are not imposed at the file level. For example, AIX systems permit the system administrator to impose quotas at the directory level, limiting the number of bytes that may be contained by *all* the files in a given directory.

File names

One of the most primitive features of the MPE file system is the file naming convention and directory structure. These features date back to the original design of MPE in the 1970s. The 16-bit “classic” models of the HP 3000 imposed the following restrictions on files and the file system.

- All MPE file names were restricted to no more than 8 characters.
- File names were case insensitive. On MPE the file names “myfile” and “MYFILE” are equivalent.
- All files were contained in a two level directory structure. UNIX and Windows have free-form directory structures in which users may contain directories (or folders), which in turn may contain other directories, and so on. By contrast, on

MPE the entire file system was divided into structures called ACCOUNTS. The accounts, in turn were divided into structures called GROUPS. All files had to reside in these groups. Groups could not contain other groups; the entire structure was restricted to two levels.

- Like file names, group names and account names were restricted to 8 case insensitive characters.
- A fully qualified MPE file name was represented by the file name followed by the group name followed by the account name all separated by periods. For example FILE.GROUP.ACCOUNT represents a file named FILE residing in a group called GROUP, which in turn resides in an account named ACCOUNT. Fully qualified file names are analogous to UNIX path names, but they must conform to this three-level syntax.

In the mid 1990s, HP enhanced the MPE operating system by adding POSIX extensions. Support for UNIX style directory structures became part of MPE/iX. MPE/iX3 calls this feature the Hierarchical File System or HFS.

- Files and directories were allowed to have names longer than 8 characters.
- As on UNIX, file and directory names became case sensitive — so that the names “myfile” and “MYFILE” could be used to represent two different files in the same directory.
- Path names could be used to refer to files in much the same way as they were used on UNIX, with file names and directory names separated by slashes. For example, a file with the fully qualified MPE filename FILE.GROUP.ACCOUNT could now also be referred to by its POSIX pathname, which would be /ACCOUNT/GROUP/FILE.
- “MPE”’s directory structure could now be more than 2 levels deep.
- In the past, only MPE Accounts could be attached to the top of the directory tree (i.e. the “root”). Now files and directories could also be attached to the root.
- In the past, MPE Accounts could only contain MPE groups. Now files and directories could be attached directly to the accounts.
- In the past, MPE Groups could only contain files. Now Groups could also contain directories and files.
- As on the target systems, directories could contain files as well as other directories, which could in turn contain more directories. This allows for a directory structure that can go far deeper than the 2 levels permitted by the “classic” MPE file system.

These enhancements were added to MPE in a way that maintained backward compatibility. In this way, existing MPE applications that were written around pre-POSIX file naming conventions could continue to run without the need for any modifications. The vast majorities of MPE applications were originally designed prior to the POSIX enhancements, and were not modified to take advantage of them. For this reason, when porting these applications, MPE’s file naming conventions must be taken into account.

For example, if a program running under MPE creates a file called MYFILE, it can refer to it in a number of different ways, including:

- MYFILE,

- “myfile”,
- MYFILE.MYGROUP,
- MYFILE.GROUP.ACCOUNT, (assuming that the user running the program is logged onto GROUP and ACCOUNT)

However, if you port this program to UNIX without Transport, there will be problems. The creating program will still be able to refer to the file it created as MYFILE. But an attempt to reference it in any of the other ways shown above will not in most cases.

- UNIX file names are case sensitive, so an attempt to reference the file as “myfile” will fail.
- UNIX will assume the MYFILE.MYGROUP and MYFILE.GROUP.ACCOUNT are the names of files in the current working directory. It won’t resolve the group and account names, because groups and accounts don’t exist on UNIX.

If you need to port an application that uses these older MPE file-naming conventions, (as most of them do) you have two choices:

- You can either convert the application to use the HFS-like naming conventions of the target platform, or
- You can use Transport, which allows for the use of the old names.

Transport using both formats MPE style file names to HFS style names. For example, suppose you run a program on a UNIX system that creates a file called MYFILE.

- The program’s current working directory will be /ACCOUNT/GROUP. Therefore the file will be created with the path /ACCOUNT/GROUP/MYFILE
- Transport will ignore the case of referenced filenames. Therefore, references to MYFILE or “myfile” will both be resolved to the same file.
- Although accounts and groups don’t exist on UNIX, the emulator will resolve references to MYFILE.MYGROUP or MYFILE.MYGROUP.ACCOUNT to the correct file.

If your target system is being used for purposes other than emulating an HP e3000, you may find it undesirable to attach the MPE accounting structure to the root. Another option is to place the two level MPE structure somewhere lower down in the directory tree of your target platform. Emulator packages allow you to define the location of the MPE root using an environment variable. For example, you might define XPORTDIR=“/directory1/directory2/”. In this case, the MPE fully qualified file name: FILE.GROUP.ACCT will reference the UNIX path /directory1/directory2/ACCT/GROUP/FILE.

Transport can greatly reduce the amount of work needed to migrate MPE/iX programs. In fact, for a COBOL only application, the entire application can be migrated without changing the file references. However, migrated Powerhouse applications do not use the Transport MPE-like file system. The emulation routines maintain information about the files for you automatically, such as record size and ASCII/binary. Non-emulated applications are unaware of the information and if not careful can corrupt the files.

When the application consists of both Powerhouse and COBOL application it is best to change all references to files to use an HFS structure.

MPE file types

Another feature of the MPE/iX file system is its support for specialized file types. Special file types provide features that may not be accessible using standard COBOL statements, and therefore may require the use of MPE intrinsics. When a file is created, the file's type is specified. The following are some of the file types supported on MPE.

Flat files

By far the most common type of file on MPE systems is the flat (standard) file. When you create a file on an HP e3000, if you do not specify a file type, it will default to STD (standard file). A standard file consists of a group of records beginning with record 0 and ending with record $n-1$ (where n is the maximum specified in the filesize option). Standard files may be read or written using either sequentially or using direct access. An example will suffice to explain how it works.

- Suppose a program opens a standard file for sequential output access. The records that it writes will be stored in this output file in the order in which they were written.
- If a program opens a standard file for sequential input, the records will be read in the order they're stored, (i.e. the order in which they were written).
- Programs may also open standard files using direct access. This permits a program to read or write a record based on its relative record number. So, for example, a program can read the n th record in a file without having to first read the $n-1$ records preceding it.

Message files

Message files (type MSG) are used for interprocess communication (IPC) under MPE. IPC is a facility of the file system that permits processes to communicate with one another in an easy and efficient manner. Message file functionality requires the use of the FREAD and FWRITE intrinsics. Message files act as first-in-first-out, queues of records. An example will explain how they are used.

Consider two processes, "A" and "B". Process "A" needs to send information to process "B", which is running at the same time as "A".

- Suppose Process "A" writes a record to a message file using the FWRITE intrinsic.
- Other processes may now read the record left by process "A". Process "B" can do this using the FREAD intrinsic.
- Message file reads are destructive. This means that when process "B" reads the record written by "A", that record will be deleted from the message file. This prevents records in message files from being read more than once.

Message files functionality is provided by Transport for AIX. COBOL programs accessing them through the Transport provided intrinsics will experience the same behavior as on MPE/iX.

On the other hand, Powerhouse applications on AIX are only able to access UNIX-like files through the AIX file API. The solution is to use the Transformix Tools VFS which is explained below.

Circular files

Circular files are wrap-around structures that behave just like standard files until they are full.

When a standard file is created on MPE, a filesize is imposed, limiting the number of records that the file can contain. If the filesize is "n", any attempt to write "n+1" records to the file will result in an error.

Similarly, when a circular file is created on MPE, there is a limit ("n") on how many records the file can contain. As records are written to the circular file, they are appended to the end of the file. But writing record number n+1 to a circular file does not cause an error. Instead, it causes the block at the beginning of the file to be deleted and all other blocks to be logically shifted toward the head of the file.

Circular files have been used on MPE for logging purposes. An application program can write to a circular file forever, and the file will always contain a log of the last n records. However, both readers and writers may not simultaneously access circular files. When all writers have closed the file, a reading process may open it. A reader takes records from the circular file one at a time, starting at the beginning of the file.

By default, circular file functionality is not part of any of the AIXs. Circular files functionality is provided by Transport for AIX. COBOL programs accessing them through the Transport provided intrinsics will experience the same behavior as on MPE/iX.

On the other hand, Powerhouse applications on AIX are only able to access UNIX-like files through the AIX file API. The solution is to use the Transformix Tools VFS, which is explained below.

Transformix Tools VFS

A Virtual File System (VFS) is an interface providing a clearly defined link between the operating system kernel and a different File System. The VFS supplies the applications with the system calls for file management (like "open", "read", "write" etc.), maintains internal data structures (the administrative data for maintaining the integrity of the File System), and passes tasks onto the appropriate actual File System.

The Transformix Tools VFS provides a way for COGNOS Powerhouse applications to access MPE-like specialized files through the standard UNIX file API. The VFS server exports the MPE-like files and provides the Powerhouse system access to them through what seems like the native AIX file system. After these files are mounted, they become

standard UNIX files to the Powerhouse application. Support is provided for MPE Message files and MPE circular files.

Spooled devices

By default, a printer is considered to be a non-sharable device by MPE, which means only one user, could use it at a time. To overcome this problem, HP e3000 printer devices are *spooled*. *Spool* is an abbreviation for *simultaneous peripheral operation online*. A spooler allows numerous user processes requiring a printer to run simultaneously. Their printed output is re-routed to disk files called *spoolfiles*. The spooler then prints the contents of these spoolfiles one at a time.

The spooling process for a printer is activated by the *SPOOL* console command. Once the spooler is activated for a printer all output that is sent to that device is automatically spooled. MPE's spooler provides a rich set of functionality for managing spoolfiles:

- Spoolfiles are queued up in the order in which they are received by the spooler. Users can assign output priorities to their spoolfiles, which can affect their position in the queue.
- Users can also assign attributes to their spoolfiles, including the number of copies that will be printed, or whether or not special forms be mounted on the printer when it's their spoolfile's turn to print.
- Users can view the contents of spoolfiles before they print, or alter the spoolfile's attributes.
- The MPE spooler supports "Page Level Recovery". This means that should the spooler be interrupted by an error, (such as a system "crash"), it can pick up at the point at which it left off when the system is restarted.

On AIX and Linux the spooler process does not provide such a rich set of functionality. The spooler is not associated with a particular device file (printer) when it is started. The spooler process is invoked directly and it then manages the output. An example will help explain some of the differences between MPE's approach to spooling and that of the AIXs.

Suppose an HP e3000 has a spooled printer configured with the device class name "LP". Assuming that this device has been spooled (i.e., that the *SPOOL* console command was issued), this user could issue the following file equation: **:FILE PRINT;DEV=LP**

From this point forward, any program that opens a file named PRINT will be opening a spoolfile. Any records written to the file named PRINT will be sent to the spooler. The user has the options of specifying spoolfile attributes (e.g. number of copies, output priority), by specifying appropriate parameters on the *:FILE* command.

Let's contrast this MPE spooler example with the AIX spooler. On AIX if a process opens */dev/lp* the process would "own" the printer until it closed it. To get your output spooled you would have to send it to the spooler process, instead of to the printer.

For example you might issue the shell command: **cat file > lp**. *lp* is a UNIX function that sends output to a spooler daemon that then sends the output to a printer. In addition to *lp*, there is the *pdpr* command that uses the HP Distributed Print Services (HPDPS).

Another way to send output to a spooler is to open a pipe using *popen()* in your program to the spooler process. This allows output to be sent directly to the spooler.

The bundled spoolers available on the recommended target machines vary in capabilities and most do not have the robust feature set of the MPE spooler (e.g. page level recovery). Third party spoolers are available that offer functionality beyond the basic capabilities. Additionally some of the emulator solutions include spooling capabilities.

Transport provides an MPE-like spooler called *mpespooler*. COBOL programs can print using this. Powerhouse programs simply use native AIX spooling.

File domains

MPE file can exist in three different domains, New, Temporary and Permanent.

- A new file exists only for the duration of the process that created it. It cannot be opened, accessed or seen by another process. When a process terminates on MPE, any files that it created in the new domain will be automatically deleted.
- A temporary file is one that exists only for the duration of the Job or Session that created it. It can only be opened, accessed or seen by other processes in the creating Job or Session. When a job or session logs off the system, any temporary files that they created are deleted automatically.
- A permanent file exists as a file in the system file domain. Its existence is not limited to the duration of its creating Job or Session, and depending on security restrictions; it may be accessed by jobs or sessions other than the one that created it. Permanent files are visible to any and all users on the system. There can only be one permanent file on the system with a given fully qualified file name.

It is common practice on MPE systems to create temporary work files in the temporary domain. The fact that the temporary domain is only visible to the creating job or session means that multiple instances of a job can run concurrently without fear of interfering with one another.

For example, suppose a user streams a batch job called J1. A program being run by J1 creates a scratch file called WORK, and places it in the temporary domain. If the user streams the job again, it will create a second job - we'll call it J2. J2 runs the same program as J1, so it too will create a scratch file called WORK. Because each job has its own temporary domain, the two WORK files are completely separate from one another. The files in J1's temporary domain are invisible to J2 and vice-versa.

Temporary files are automatically removed from HP e3000 systems when the creating Job or Session is completed, even if it is an abnormal termination.

On the AIXs, there is only one file domain: permanent. This means that programmers must make certain there are no naming conflicts in the file names that are used. Some programmers achieve this by creating unique file names (using the Process ID or PIN as part of the file name or putting in a time stamp in the name).

There is a convention, often used on UNIX systems, of putting temporary files into a directory called temp or /tmp. It is commonly agreed that files in the temp directory don't need to be retained. System administrators periodically purge any files that are left in the temp directory. Transport simulates MPE's temporary domain by generating unique names and placing them in the temp directory, or purging them automatically.

FILE equations

FILE equations are a unique feature of the MPE file system. FILE equations are a way to declare file attributes outside of a program. These attributes are used at the time the file is opened. They may be used to override programmatic or the system's default file specifications. FILE equations are Job or Session specific.

There are three basic uses of FILE equations:

- 1) **Change the attributes of a file that is opened in a program.** For example suppose a program has been written to create a scratch file that can hold 2000 records. When the program terminates, this file is deleted automatically. As your company grows, the program is called upon to process more data until one day, you find that 2000 records is no longer large enough. The program needs a larger scratch file. Even though the limit is hard coded in the program, it can be overridden using a FILE equation. Issue a FILE command associating the name of the scratch file with a limit of 4000 records. When the program creates the file, (by opening it), the attributes specified in the :FILE equation override the attributes specified in the program. For example, some of these attributes include:
 - o The capacity of the file (which determines the maximum amount of data that the file can contain)
 - o The structure of the file (which determines the size of the records that make up the file)
 - o The file domain, (which determines how long the file remains on the system)
 - o The file type
- 2) **Point a file to a different file type or device.** For example suppose a program normally writes a report called REPORT to the system line printer. One day, you have a need to route the report to your terminal. This can easily be achieved by issuing a FILE command before you run the program that associates REPORT with \$STDLIST (the filename associated with each user's terminal). When the program opens the REPORT, the attributes specified in the FILE equation will override those in the program. Each record that is written to the REPORT file will be directed to \$STDLIST instead, and be displayed on the user's terminal.
- 3) **Change the name of a file.** For example a program creates an output file called OUTPUT but with a FILE equation you can redirect the records written to this file to a different file: ORD0001D.WORK.ACCT.

There are more things you can do with FILE equations but they are variations of these three basic uses. What is common to all its uses is the ability to change aspects of a file without having to change the compiled code, all FILE equations are issued in JCL, command files, UDCs and the Command Interpreter (CI, the MPE shell).

Transport implements MPE File Equations for COBOL and other compliable programs. However, Powerhouse programs for UNIX are not designed to take advantage of this. Since FILE equations do not exist on any of the AIXs so the functionality has to be implemented in different ways. Let's go through the three basic uses:

- 1) **Change the attributes of a file that is opened in a program.** AIX does not offer the rich set of file attributes offered by MPE. Basically, file structures are very simple. There's no record structure, and only one type of file. This means that there are no attributes to change. Therefore, this functionality is not needed on the AIX.
- 2) **Point a file to a different file type or device.** On the AIXs, changing device types needs to be anticipated and programmed for. The device being used by the program can be defined in a configuration file or be set up in an environment variable.
- 3) **Change the name of a file.** Changing the name of a file can also be handled by using a configuration file or an environment variable. Often what a name change is doing is putting a file in a specific group and account or directory. Using a path variable can perform this same functionality. For example, suppose a program creates a work file called ord0001d. For this application all work files are located in /appl/work/. A path environment variable called WPATH can be set to "/appl/work/". The program can then be changed to get the WPATH variable and concatenate it with the file name to establish the full file name (/appl/work/ord0001d). Another way to redirect a file name is with symbolic links with the ln command. FILE equation functionality is also available in many of the emulation environments.

When Powerhouse is used on the same system as COBOL, Transport handles the details of providing the correct information to the modified Powerhouse program via environment variables.

KSAM

KSAM is the Keyed Sequential Access Method database system that comes with the MPE/iX operating system. The KSAM access method makes it possible to access files sequentially, or directly, or using Keyed Access. Sequential access and direct access is similar to the functionality associated with standard files so we won't explore that any further here. However, KSAM files can also be accessed using keys. Keyed access requires that a field in each record be identified in advance as a key field. When data is written to a KSAM file, the KSAM access method not only writes the data, it also writes the key field to an index, which can then be used to retrieve the data record. This allows applications to access records from the middle of a file without having to search the file sequentially, or keep track of relative record numbers.

KSAM has evolved over the years so that there 3 different versions of the KSAM access method available on the HP e3000.

- There is CM KSAM, the original compatibility mode KSAM. This dates back to the "classic" HP 3000 architecture. When MPE was re-written for the PA- RISC

architecture, CM KSAM was brought over from the earlier 16-bit architecture in compatibility mode. Although it worked, it was subject to many of the limitations of the older 16-bit architecture. It is normally maintained using a utility program called KSAMUTIL, and accessed using the CKxxx intrinsics.

- KSAM XL is a native mode implementation of KSAM that was implemented on the PA-RISC models of the HP 3000 as the limitations of CM began to become problematical for customers who were using CM KSAM. Programmatically speaking, KSAM XL is compatible with CM KSAM, but it is more resilient against system problems and application aborts.
- KSAM 64 a newer native mode implementation of KSAM that supports larger files.

As KSAM evolved, HP maintained a high level of compatibility, so that applications could be migrated to new versions without having to be rewritten. All three kinds of KSAM files can be created using MPE's :BUILD command and accessed using the file system intrinsics (HPFOPEN, FREAD...). KSAM files can be accessed from COBOL using the standard COBOL file I/O verbs (READ, WRITE, DELETE...) if the file is declared to be "Indexed" with the SELECT clause in the INPUT-OUTPUT SECTION. MPE applications that make use of one of the KSAM access methods can be ported to any of the AIXs. However, no KSAM access method is bundled with AIX. Transport implements KSAM for COBOL programs. It uses C-ISAM from Informix (now owned by IBM). Since Powerhouse supports C-ISAM, COBOL programs can access the data using KSAM and the Powerhouse programs can access the same data using C-ISAM.

Most MPE applications make use of intrinsics — either to access features of the file system, or to access some other part of MPE. To migrate these programs, the code referencing the intrinsics either needs to be changed, or you need to emulate the intrinsic. Fortunately, Transport provides this emulation.