IBM

International Technical Support Centers

CONVERTING SYSTEM/36 ENVIRONMENT APPLICATIONS TO NATIVE AS/400

GG24-3304-01

Converting System/36 Environment Applications to Native AS/400

Document Number GG24-3304-01

September 1990

International Technical Support Center
Rochester, Minnesota

Take Note

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page iii.

Second Edition (September 1990)

This edition applies to Release 2.0 of the OS/400 licensed program and related licensed programs, and the IBM AS/400 Programmer Tools PRPQ 5799-DAG Release 2, and to all subsequent releases until otherwise indicated in new editions or technical bulletins.

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

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

IBM Corporation, International Technical Support Center
Department 977, Building 003-1
Rochester, MN 55901 USA

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

Special Notices

This publication is intended to help the customer to convert AS/400 applications that run in the System/36 Environment so that they do not require the System/36 Environment. It primarily contains discussions of the characteristics of System/36 Environment applications and their native counterparts, and details many of the steps required to accomplish conversion.

The information in this document is not intended as the specification of the interfaces that are provided by the AS/400 system for use by customers in writing programs to request or receive its services. See the Publications section of the IBM Programming Announcements for the AS/400 system.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

AS/400, OS/400, RPG/400, COBOL/400, System/370, OS/2, Systems Application Architecture, and SAA are trademarks of the International Business Machines Corporation.

Abstract

This document is intended for the AS/400 user, planner, or manager who plans to convert a System/36 Environment application to native AS/400. It is assumed that the reader is familiar with the application being converted, with RPG or COBOL, OCL, and SFGR. It is also assumed that the reader is familiar with the System/36 Environment, AS/400 database concepts, the design and creation of physical and logical files, CL, and, to a lesser degree, work management.

This document describes the steps needed to convert applications running in the System/36 Environment to native AS/400. It also discusses the tools and approaches that can be used in the conversion process.

ASYS (210 pages)

Acknowledgments

This document has been revised by:

Errol Baird
IBM New Zealand

Klaus Pretsch
IBM Germany

The advisor for this project was:

Mike Anderson
International Technical Support Center, Rochester

This publication is the result of a residency conducted at the International Technical Support Center, Rochester.

Preface

This document is intended to give the reader an understanding of the steps needed to convert System/36 Environment applications to native AS/400. It discusses the choices to be made when converting, and includes performance considerations. It also discusses tools and techniques that can simplify conversion and lead to a more satisfying result.

This document is intended for the AS/400 user, planner, or manager who plans to convert a System/36 Environment application to native AS/400.

This document begins with a discussion of the choices one must make when considering conversion, such as whether to convert and the approaches that can be taken. Then it includes a list of actions that can improve the performance of applications *without* requiring conversion. It then discusses initial steps that will help make the conversion process manageable. Next is a discussion of the conversion of program-described files into externally described database files using the Programmer Tools PRPQ. Then the document discusses conversion for display files, menus, and printer files, followed by decimal data error handling, special considerations for RPG and COBOL, general high-level language and utility considerations, OCL to CL conversion, national language considerations, and Systems Application Architecture (SAA).

Related Publications

Following is a list of publications that contain additional information about the topics covered in this document.

ITSO Publications

System/36 to AS/400 Application Migration, GG24-3250

System/36 to AS/400 System Migration, GG24-3249

Writing SAA Applications for AS/400, GG24-3438

AS/400 Manuals

IBM AS/400 Information Directory, GC21-9678

IBM System/36 and System/38 Application Design Considerations, G580-0912

IBM System/36 to IBM System/38 Conversion Aid, SC09-1067

Control Language Reference, SBOF-0481

Programming: Control Language Programmer's Guide, SC21-8077

Programming: Database Guide, SS21-9659

Programming: Data Description Specifications Reference, SC21-9620

Programming: Data Management Guide, SC21-9658

Programming: Work Management Guide, SC21-8078

Programming: System Reference for the System/36 Environment, SC21-9663

Programming: Concepts and Programmer's Guide for the System/36 Environment, SC21-9663

Migrating from System/36 Planning Guide, GC21-9623

System/36 to AS/400 Migration Aid User's Guide and Reference, SC09-1166

Languages: COBOL/400 Reference, SC09-1240

Languages: COBOL/400 User's Guide, SC09-1158

Languages: System/36-Compatible COBOL User's Guide and Reference, SC09-1160

Languages: System/38-Compatible COBOL User's Guide and Reference, SC09-1159

Languages: RPG/400 Reference, SC09-1089

Languages: RPG/400 User's Guide, SC09-1161

Languages: System/36-Compatible RPG II User's Guide and Reference, SC09-1162

System Operations: Operator's Guide, SC21-8082

Systems Application Architecture (SAA) Publications

SAA: An Overview, GC26-4341

SAA Common Programming Interface: Application Generator Reference, SC26-4355

SAA Common Programming Interface: C Reference, SC26-4353

SAA Common Programming Interface: Communications Reference, SC26-4399

SAA Common Programming Interface: FORTRAN Reference, SC26-4357

SAA Common Programming Interface: COBOL Reference, SC26-4354

SAA Common Programming Interface: Database Reference, SC26-4348

SAA Common Programming Interface: Dialog Reference, SC26-4356

SAA Common Programming Interface: Presentation Reference, SC26-4359

SAA Common Programming Interface: Procedures Language Reference, SC26-4358

SAA Common Programming Interface: Query Reference, SC26-4349

SAA Common User Access Basic Interface Design Guide, SC26-4583

SAA Writing Applications: A Design Guide, SC26-4362

Other Publications

American National Standard Programming Language COBOL, ANSI X3.23 - 1985

Contents

Figures

1.0 Introduction

This chapter discusses possible ways of running System/36 applications on the AS/400[1] system. It discusses (very generally) migration, restructuring, conversion, and redesigning. It might not be reasonable, in some cases, to fully convert an application. In other cases, fully redesigning might be the best way.

1.1 Migration, Restructuring, Conversion, and Redesigning

There are four main approaches to running System/36 applications on the AS/400. These are migration, restructuring, conversion, and redesigning:

- Migration involves using the migration aid programs to move System/36 applications into the System/36 Environment on the AS/400.
- Restructuring involves changing some characteristics of an application while keeping it as a System/36 Environment application.
- Conversion involves selectively changing parts of the application to replace some of the System/36 functions with native AS/400 functions.
- Redesigning involves completely reworking the application to use all AS/400 functions to the fullest extent.

You must consider many factors when you are deciding whether to leave an application running in the System/36 Environment, or to convert or redesign it. These factors could include expected lifetime of the application, stability of the application, cost of maintenance, growth in business volumes, training needs, system capacity and performance targets, and availability of new or simplified functions.

1.2 Migration

The migration process is straightforward, does not take much time, and protects the user's investment in existing applications. Migration allows the user to move away from the System/36 hardware while continuing to maintain applications with existing skills. In addition, migrated applications can use some of the advanced functions of the AS/400, such as journaling and debugging.

On the other hand, migrated applications do not have access to many of the AS/400 functions which, in the long run, can reduce the cost of application maintenance and enhancement. The performance of applications that are simply migrated might not be as good as it could be if the applications were restructured or redesigned to take advantage of the AS/400 architecture and functions. Also, if new applications are written in native AS/400, the costs and problems of maintaining two sets of programming skills might be unattractive over the long term.

1 AS/400 is a trademark of the International Business Machines Corporation.

1.3 Restructuring

While migration moves your System/36 application to AS/400 with relative ease, there are some disadvantages of running the application unchanged in the System/36 Environment. The performance of some applications might not be satisfactory. This is because many of the characteristics of System/36 applications, such as multiple requester terminal (MRT) and read under format (RUF) take advantage of the architecture of System/36 and the design of its operating system. Some of the characteristics were *necessary* to ensure adequate performance on System/36. AS/400 has a very different architecture, so some System/36 characteristics are not only unnecessary, they can even degrade performance. Because of this, restructuring the application while keeping it in the System/36 Environment can be helpful, and some relatively simple structural changes can improve performance significantly.

On the other hand, this approach is more expensive than migration. It does not tap the additional function of the AS/400, nor does it improve your ability to maintain and enhance your applications.

1.4 Conversion

If your application needs access to functions that are not available in the System/36 Environment, or if you expect significant maintenance or enhancements, conversion might be the best solution. Only those parts of the application needing to move away from the System/36 Environment need be touched. Thus conversion could allow you to meet your objectives without the expense and time required to fully redesign the application.

Partly converted applications might also form a satisfactory basis for further enhancements through use of more AS/400 functions, or for development of new applications based on existing data files.

On the other hand, a bit-by-bit approach towards moving to the AS/400 native functions, while still keeping the underlying design of the System/36 application, may involve as much effort as redesigning without providing the corresponding advantages. Even extensive conversion may fail to remove restrictions inherent in the original design.

Also remember that conversion alone will not necessarily improve the performance of your application.

1.5 Redesigning

Redesigning can produce significant benefits in several areas. (These benefits also apply, generally to a lesser extent, to the conversion process.)

Consider redesigning for applications that have high visibility, are heavily used, need optimum response times, or lend themselves to significant functional enhancement.

Redesigned applications might be able to use some of the code already written and redesigning could use fewer resources than conversion.

Here are some of the benefits of redesigning applications:

- Improved throughput and response times.
- Simplified program logic through the use of system-supported relational data handling methods (for example, improved selective record processing, simpler handling of multiple record files through record-name operations, and better data sequencing functions).
- More productive design and development of new applications
- Replacement of batch applications by (simpler) interactive ones
- Access to new functions in the languages you already use
- Availability of new languages and utilities such as Structured Query Language (SQL) and Query.
- Reduction of the need to maintain System/36 skills
- Compliance with IBM Systems Application Architecture (SAA)[2] standards
- Future growth as IBM delivers new AS/400 functions
- Better and faster tools for recovery and restart.

In general, redesigned applications provide the best basis for design and implementation of both modifications and extensions.

On the other hand, fully redesigning may involve more work than is justifiable for a stable application having few planned modifications or enhancements and which has no critical performance requirements.

1.6 Full Conversion Versus Redesigning

The degree to which you rework each System/36 application to run on the AS/400 is up to you. Many different combinations are possible.

For example, you could:

- Use the migration aid, making only those changes needed to recompile and run an application on AS/400, then leave the application alone.
- Keep the applications in the System/36 Environment, but make some structural changes to improve performance of critical programs as discussed in 2.0, "Restructuring for Better Performance" on page 5.
- Convert some or all parts of each application:
  - Convert database files to externally described files, but leave screen files as program-described. This would allow new applications to be developed using the existing data.
  - Convert some Operation Control Language (OCL) statements and procedures to Control Language (CL) to gain access to desirable CL functions.

[2] Systems Application Architecture and SAA are trademarks of the International Business Machines Corporation.

- Convert programs and files to native AS/400 but run the programs from System/36 Environment OCL procedures. This could provide performance benefits without OCL conversion in some cases.
- Fully convert procedures, programs, and files for your application.

- Redesign key applications or parts of applications. For example, you might wish to redesign the interactive part of an application but leave the batch part unchanged as far as possible.

1.7 Recommendations

Remember that a decision to convert or redesign your application should be made with long-term goals in mind. Performance improvements alone are not always sufficient reason to convert. In fact, significant performance gains can be made merely by restructuring your application or making some operational changes. With this in mind, 2.0, "Restructuring for Better Performance" on page 5 describes ways you can improve performance without converting your applications to native AS/400. The rest of this document discusses conversion and redesigning.

As a general rule, the more thoroughly the conversion or redesigning is done, the more the benefits will be realized in the long term. Although there might seem to be lots of steps in the conversion process, many of the steps are needed to handle worst-case situations. Any particular application is not likely to need all the steps.

We suggest that you try a pilot conversion of a small application so that you understand the steps. Then you can estimate the conversion effort and balance it against the benefits to be gained in your computer system.

We strongly recommend that the database should be described externally in all cases. The process is straightforward and will give the application access to the relational database.

Program conversion for batch programs involves few changes. Program conversion for interactive programs requires extra changes to make display file processing consistent with AS/400 methods. In many cases the program changes are made as part of the file changes. Conversion of procedures and OCL can be more complex. The use of the programs discussed in this document can accurately convert many of the System/36 statements. But because of the new functions available on AS/400 and because of the different approaches of OCL and CL, it might be better to understand the intent of the OCL procedures and completely rewrite them.

Display files also need careful consideration. Unless a display file is shared among several programs, there may be little advantage in making it external. In addition, because we already had a kind of "external file" for displays on System/36 (the S and D specifications), more work might be needed to reconcile the field names.

2.0 Restructuring for Better Performance

This chapter summarizes and briefly describes ways of significantly improving the performance of System/36 Environment applications without converting them to native AS/400 applications. Additional information is available through your IBM representative or business partner. The information contained in this chapter is covered in more detail in the *IBM AS/400 S/36 Environment Performance Tuning Guide*, HONE number 155NC. The Tuning Guide is worth reading because it will contain the latest detailed performance information.

Application design on a particular machine is influenced by the functions available in that machine's architecture. Applications that are designed to run well on one machine might use functions that are supported differently on another machine, and might thus perform poorly on the other machine.

For example, System/36 users, partly because of the 64K limit, often use multi-step programs alternated with OCL, which passes data using the local data area (LDA) or read under format (RUF). Low memory sizes in the early life of the System/36 led to extensive use of multiple requester terminal (MRT) programs. Relatively small disk capacity and simple file-management systems required user control over disk allocation and the use of sorts and work files. These features will be particularly present if the application had originated from earlier System/3X environments.

On the other hand the AS/400, while removing many of the System/36 design constraints, operates in an environment where file creates, file opens, and job initiations take longer to complete, and where save/restore methods can be more complex.

This suggests that the causes of difference in performance are likely to be found in the above areas.

2.1 Relative Performance

Measurements of interactive transactions in an unstressed environment indicate that a migrated System/36 interactive application can handle 50 to 85 percent of the throughput of a similar native application given that each application is optimized to the System/36 or AS/400 environment respectively.

In a stressed environment with a shortage of disk space, the interactive performance can be significantly worse.

The following sections describe application changes that might significantly improve the performance of applications running in the System/36 Environment. They might permit you to use a smaller AS/400 configuration than would be required if the changes were not made.

In the material that follows, those changes which solve the most commonly-occurring performance problems are listed first. Next are changes that solve problems that occur less frequently, but have more dramatic effect when they do. Last are assorted other recommendations. Of course, your results might vary, depending on the structure of your applications.

2.2 Recommendations

You can make all these changes without converting your application from the System/36 Environment. Some are merely operational changes. Some are discussed in more detail in the following paragraphs.

- Make MRT programs never-ending, or specify a long MRT delay time.
- Reduce file create and delete activity.
- Use shared database file opens.
- Increase the DBLOCK parameter value for sequentially accessed files.
- Use packed decimal data, particularly in compute-intensive programs.
- Reduce unnecessary use of EVOKE and JOBQ.
- Avoid unnecessary nesting of operator commands.
- Use alternative index or logical files instead of sorts under certain circumstances.
- Remove unused and unnecessary OCL.
- Increase storage pool size.
- Use of 27x132 display support carefully.
- Eliminate or simplify read under format.
- Avoid batch-type work in an interactive pool.
- Turn off procedure logging.
- Delete unused spool files.
- Clean up history logs.
- Remove RPG F-specifications that are used only by the DEBUG operation code.
- Limit sign-on and sign-off activity.
- Change MRT security.

2.2.1 Making MRT Programs Never-Ending, Specifying Long MRT Delay Time.

First, note that MRTs perform well if they are used correctly.

AS/400 is relatively slow when initiating MRTs because starting a new job for the MRT requires a great amount of system resources. So, avoid ending MRTs when the last user exits. You can do one of the following to facilitate this:

- Compile individual MRT programs as never-ending programs (MRT-NEP). This can be done via a parameter or keyword on either the RPGC procedure or the CRTS36RPG CL command. You can also use the EDTS36PGMA command to change the attribute for a program that already exists. Finally, you can specify NEP-Y on the ATTR OCL statement.
- Change the MRT delay time to at least 300 seconds (5 minutes) or up to an hour or more for those MRT programs that you do not want defined as never-ending. This delay does not affect system performance. The delay value determines how long a non-NEP MRT remains active after the last user has exited it. To enable MRT delay, use the CHGS36 command to modify the time-out value to a non-zero value. Once MRT delay has been enabled, you

can activate and deactivate it on a procedure basis using the EDTS36PRCA command.

A MRT-NEP, or a MRT with a long delay will generally perform better than the same program as an SRT. The MRT saves memory (It uses only one PAG) and file opens. Converting MRTs to SRTs is recommended only if the application is redesigned for the AS/400 native environment or if queuing of multiple users is causing response time problems.

2.2.2 Reducing File Create and Delete Activity

Creating and deleting scratch or work files is common in System/36 applications. This activity uses many more AS/400 system resources and can be key to program load performance. Reducing work file creation and deletion where possible will help improve performance.

Consider creating file members that remain permanently on disk and removing data from the members with the CLRPFM command (This approach replaces the use of DELETE, BLDFILE or // FILE RETAIN-S or -J OCL). The system will handle space allocation automatically.

2.2.3 Using Shared Database File Opens Where Possible

File opens are key to program load performance and take longer on AS/400. If the application closes and re-opens files many times then performance can be improved by making the first file open a "full" open and the subsequent opens of the same file "shared" opens that use fewer system resources.

The System/36 Environment automatically shares open data paths if the file is used in the next job step, and if all the open options and access methods are the same as in the previous job step or if the JOB-YES keyword is used.

The example below shows how the System/36 Environment analyzes the job stream FILE statements and tries to use automated shared file opens to decrease the system overhead. Note that the key to the analysis is not the program file name (in this case 'FILE1') but the LABEL name.

```
// FILE NAME-FILE3,LABEL-JOBFILE,JOB-YES
// LOAD PROGA
// FILE NAME-FILE1,LABEL-MASTFILE
// FILE NAME-FILE1,LABEL-TRANFILE
// RUN
        Program opens FILE1     <---- Full open of MASTFILE
        Program opens FILE2     <---- Full open of TRANFILE
        Program opens FILE3     <---- Full open of JOBFILE
        Program closes FILE1    <---- File is held open
        Program closes FILE2    <---- File is held open
        Program closes FILE3    <---- File is held open
// LOAD PROGB
// FILE NAME-FILE1,LABEL-MASTFILE
// RUN
                                <---- TRANFILE is closed
        Program opens FILE1     <---- Shared open of MASTFILE
        Program closes FILE1    <---- File is held open
// LOAD PROGC
// FILE NAME-FILE1,LABEL-JOBFILE
// RUN
                                <---- MASTFILE is closed
        Program opens FILE1     <---- Shared open of JOBFILE
        Program closes FILE1    <---- File is held open
// RETURN
                                <---- JOBFILE is closed
```

To implement shared opens, consider doing the following:

- Add the JOB-YES keyword where applicable. However, be aware of the effect of any DISP keywords associated with the file.
- Change programs using the same files to always open the files the same way. For example if the first program opens the file for input and the next program opens it for update, change the first program so it also opens the file for update.
- Add a CL program to pre-open the files, thereby forcing shared opens for the entire session. Create your CL program with the statements

  ```
  OVRDBF FILE(file-name-on-disk) SHARE(*YES)
  OPNDBF FILE(QS36F/file-name-on-disk) OPTION(*ALL)
  ```

  Call the CL program as an initial program for each user or as part of the application startup.

  Note: The OPTION(*ALL) parameter overrides any blocking and only allows handling of single records from disk for each I/O operation. Depending on the application, this may not be the best thing to do.

2.2.4 Increasing DBLOCK Parameter Value for Sequentially Accessed Files

You can improve performance for sequentially accessed files (input or output) by increasing the DBLOCK value or by adding an OVRDBF command with a SEQONLY(*YES) parameter.

The recommended DBLOCK value is

```
For B10 - B45:  DBLOCK approximately =  (16384) / (record-length +1)

For B50 - B70:  DBLOCK approximately =  (24576) / (record-length +1)
```

To make the change, do the following:

```
// LOAD PROGA
// FILE NAME-FILE1,LABEL-BLOCKUP,DBLOCK-nn
// RUN
          *** OR ***
   OVRDBF FILE(BLOCKUP) SEQONLY(*YES nn)
```

Note: The System/36 always provides the latest version of each record. Making the blocking larger on the AS/400, where the file is being shared between programs, may not always provide the latest version, since there could be a number of changed records in the block waiting to be written to disk.

2.2.5 Using Correct Data Types

System/36 batch programs are usually designed to handle large volumes of data and hence have a lot of calculation steps, so this section applies mainly to batch jobs. The System/36 uses zoned decimal fields in its instruction steps, while the AS/400 uses packed decimal fields. Thus there are considerable performance gains to be had by changing programs that have many arithmetic instructions.

In these cases, you should consider:

- Changing to packed decimal data in your files wherever possible. This saves many unnecessary instructions for conversion to packed decimal fields when running the program. The more incompatible the data is with packed decimal, the greater is the overhead. For example, binary fields take even more overhead than zoned decimal fields.
- Replacing even-length numeric fields with odd-length fields. This saves many unnecessary instructions that deal with the high-order 4 bits when the even-length field is used in the packed decimal instruction set (which always uses odd-length).
- Removing result field truncation (such as a result field too small or shortened or decimal places dropped) wherever possible, as this has a high system instruction overhead.
- Analyzing your files for invalid decimal data (that is, non-decimal data, such as blanks, in decimal fields). While the System/36 Environment allows invalid decimal data (the same as for the System/36) many additional instructions must be performed to resolve the data. Also, if AS/400 RPG III programs are subsequently used to process that data; program failure through data exception will occur.

2.2.6 Reducing Unnecessary Use of EVOKE and JOBQ

The use of EVOKE or JOBQ to do tasks in an application system (which can be common in systems generated with an application generator) can have a dramatic impact on performance.

If your application uses EVOKE or JOBQ to do trivial tasks (tasks that take less than 10 to 20 seconds), perform them inline.

If you do not want to perform the function inline (perhaps because the response time is too great for the interactive user), use a data queue. Change the evoked program into a continually running batch job that waits to receive requests via a data queue.

2.2.7 Avoiding Unnecessary Nesting of Operator Commands

This is an operational change. Simply tell your operators that they should not run a command while the display from a previous command is still active. They should exit the previous display by pressing Enter or F3 before running the next command.

Stacking commands by entering successive commands on the command input line of the display of the previous command significantly increases the operator's requirement for main memory (the PAG) and can impact other users by increasing paging requirements.

2.2.8 Sort Performance

A general guideline is to replace sorts with logical views (using CRTLF command) if:

- The file has a low update activity where less than 30% of the record keys are changed between sorts.
- A small percentage of the total records are selected by the sort, and the selection criteria does not vary.
- Elapsed time for the batch job is critical, and there is extra interactive capacity (to maintain the logical view).

Sort performance is sensitive to pool size, so you should:

- Increase *BASE size if possible when running large sorts in batch.
- Collapse the interactive pool into *BASE for batch processing when the interactive load is light.
- Use 500K per sort minimum, or better, use 2 to 3MB for optimum performance.

2.2.9 Careful Use of 27x132 Display Support

Mapping the 27x132 display size to 24x80 can have a very dramatic effect.

Avoid writing screen formats to 24X80 displays if they are defined for 27x132 displays. Much more processing time is required to map the data stream defined for a large screen to a display with a smaller screen. This should be done only if a large majority of the displays used by the applications have the larger screen.

If this is a must, then create a new display file suitable for the 24x80 screens along with a modified program for those screens.

2.2.10 Eliminating Read Under Format

Read under format (RUF) occurs when one program (or PROMPT OCL) writes a display and another reads it. RUF in the System/36 Environment performs well when the same file is used in both steps of a single requester terminal (SRT) program.

Other cases incur significant overhead because of the need to transfer data between display files in different jobs (in the case of MRTs) or between different file open data paths in the case of:

- Different file, SRT to MRT or MRT to SRT (worst case)
- Same file, SRT to MRT or MRT to SRT
- Different file, SRT step to SRT step.

If few transactions are made between occurrences of RUF, consider removing or simplifying the RUF by:

- Making the same program that writes the display, read it (combine programs and files or change logic).
- Combine the display files used in the two steps, when in an SRT.

2.2.11 Limiting Sign-on and Sign-off Activity

You could also make another operational change. Simply tell your operators that they should remain signed on the system between transactions. If security is a concern, it might be necessary to add a routine to the application that requests and verifies a password.

2.2.12 Changing MRT Security

System/36 Environment support for MRT programs ensures that each user who attaches to MRTs is authorized to the programs and all the files they are using. This is expensive. Your security requirements might be satisfied merely by checking that each user is authorized to the programs. If this is true you can specify, using the CHGS36 CL command, that only the first user (or the owner) of each MRT needs to be authorized to all the files. Subsequent users need to be authorized only to the program.

2.2.13 Work Management Considerations

You should be familiar with the work management practices given in the *AS/400 Work Management Guide*, particularly those sections that describe balancing the activity levels and pool sizes in your system. Set your system pools and and activity levels within those guidelines once your system has reached a steady work load condition.

Other good work management practices, such as:

- avoiding batch work in interactive pools
- avoiding interactive compiles (do these in batch)
- avoiding large interactive queries
- avoiding interactive sorts (unless these are very small)
- deleting unused spool files
- cleaning up history logs
- ensuring adequate *BASE pool size

can improve the overall system performance, whether you use the System/36 Environment or AS/400 native mode.

You should control the use and definition of logical files and the maintenance of the access paths, because large numbers of logical views with use of immediate maintenance will degrade performance.

2.2.14 Using Utilities

Replace the use of system procedures, such as BLDFILE, COPYDATA, and DELETE, with direct OCL (for example, // LOAD $FBLD, // FILE.., // RUN, ...). This will generally result in better utility load performance since there are fewer OCL statements to interpret.

While this is not a significant item, it could be beneficial in a system that uses many System/36 utilities.

3.0 Getting Started with the Conversion

This chapter discusses some steps you can take to simplify the conversion process, regardless of whether you are converting your entire application or just some of it.

Note: Many of the directions in later chapters assume you have followed the steps described in this chapter.

IBM provides a product that helps with many of the conversion steps. This product is called the IBM AS/400 Programmer Tools PRPQ 5799-DAG. The functions of this tool, which are discussed in more detail later, will help you to move your application from the System/36 Environment to AS/400 "native mode". The product will generally be referred as PTK.

3.1 Attend AS/400 Education

Whether you choose to migrate, restructure, convert or redesign, you need a strong working knowledge of the AS/400 and its functions. We recommend that you use AS/400 education, starting with the AS/400 Online Education. Contact your IBM representative or business partner for course details, and work with them to develop appropriate schedules for your computer system.

3.2 Starting Point for Conversion

We assume that you have installed OS/400[3] Release 2.0, with all necessary PTFs. We also assume that the System/36 applications to be converted have already been migrated from the System/36 to the AS/400 System/36 Environment, and that you have decided to convert rather than redesign your applications. You should be familiar with:

- *Migrating from System/36 Planning Guide*
- *System/36 to AS/400 Migration Aid User's Guide and Reference*
- *System/36 to AS/400 Application Migration* (ITSC publication).

If you have not already migrated your System/36 applications to the AS/400 System/36 Environment, we recommend that you do so before starting to convert. The migration process helps ensure that all applications meet consistent standards, and converts System/36 Screen Format Generator Routine (SFGR) S- and D-specifications to AS/400 data description specifications (DDS). Also, some of the tools that help in the conversion operate on System/36 Environment files and programs.

[3] OS/400 is a trademark of International Business Machines Corporation.

3.3 Conversion Steps

A complete conversion could involve the following steps:

1. Choose the programs, files and procedures to be converted.
2. Analyze the database files.
3. Analyze fields within data files and create DDS.
4. Change the DDS to take care of special situations.
5. Create external physical and logical files.
6. Create native screen definitions.
7. Create native menu definitions.
8. Create external printer file definitions.
9. Copy data into the external database files.
10. Detect and remove sources of decimal data errors.
11. Change high-level language (HLL) programs to use external database files.
12. Change HLL programs to use external display files.
13. Change System/36 OCL to AS/400 CL programs.
14. Test programs.
15. Build a field reference file.

This is a fairly natural sequence of steps for converting program-described files to AS/400 external files which, if followed, will cut down the amount of work to be done. Some of these steps can be overlapped. You can start OCL conversion early since is not dependent on most of the other activities, even though you will not be able to finish the CL until file names have been resolved. You can start data transfer and cleanup as soon as external files have been created.

Screen files are best left until the database files have been resolved, since screen files may use database fields, and it is only at this point that you know the database field names to put into the screen files. The same is true for printer files.

3.4 AS/400 Programmer Tools PRPQ 5799-DAG

The PTK has functions and utilities to help a user convert an application to AS/400 native mode in a short time. The PTK assists by:

1. Generating DDS from your application
2. Verifying decimal data error within data files
3. Using journal commands to analyze changed data and identify programs causing decimal data error
4. Converting System/36 Environment OCL to AS/400 CL
5. Converting RPG II to RPG III
6. Converting System/36 Environment screen format source to AS/400 DDS source code

7. Converting System/36 Environment menu objects to AS/400 menu objects
8. Retrieving DDS from existing files on the AS/400
9. Creating programs on remote systems
10. Tracking the application development process

PTK documentation is available online.

3.5 Choosing Programs and Files to be Converted

You do not have to convert all your System/36 applications at one time. You do not even have to convert all of a single application. However:

1. If you plan to convert a program, try to convert all files used by that program. This avoids rework of the program later.
2. If you plan to convert a file, try to convert all programs using that file. This helps to ensure that the resulting AS/400 file definitions are accurate.

3.6 Analyzing the Database

The AS/400 database operates on files that contain a single record type, and that contain no repeating groups of fields. Also, the AS/400 requires that records containing related information (for example, about a customer order) have a common field (the order number), so that the order information can be collected for presentation to the user program. Files that are structured in this way are referred to as normalized files.

The more multiple record files, repeating groups, and missing common fields there are in your application, the less normalized the files are, and the more work that will be needed to convert them to the AS/400 database.

We recommend that before starting the conversion, you take the time to understand the idea of normalization, and reduce your files to at least second (and preferably third) normal form.

Refer to *IBM System/36 and System/38 Application Design Considerations* for a discussion of normalization.

Although in most cases we expect that there will be no changes in the logic of the programs, it might happen that changes are needed. It is worth making the changes during the conversion process, so that you can fully use the AS/400 relational database functions.

When you migrated from System/36 to AS/400, your files were copied into a "files library" on AS/400. The default files library is called QS36F, although you might have specified a different library name when you migrated. Throughout this document, references to your System/36 Environment files assume that they are in QS36F; this is reflected in the examples. If you specified a different files library name when you migrated, use that library name instead of QS36F.

3.7 Moving Selected System/36 Source to New Library

During the migration process, you set up one or more AS/400 libraries to hold the migrated applications. Perhaps you used just one library to hold all applications, or perhaps there were several libraries - one for payroll, one for order processing, and so on.

However, in each library some physical files were already created by the the migration aid:

- High-level language source was put into QS36SRC.
- Copy members were put into QS36SRC.
- OCL and procedures were put into QS36PRC.
- Display format (SFGR) source members were put into QS36SRC, and DDS members generated from that SFGR source were put into QS36DDSSRC.

In addition, each library contains object programs, and display files (created from the DDS). These are used to run your System/36 Environment applications.

At this stage, move copies of the source and procedures to be converted into a separate library so that you can convert them without disturbing the System/36 Environment.

Create a new conversion library. Your library should contain the following source files QS36SRC, QS36PRC, and QS36DDSSRC at the beginning of the conversion. If you are working in the System/36 Environment and you use the BLDLIBR Procedure, the source files QS36SRC and QS36PRC are created for you. Do the following:

- Create a source file QS36SRC containing all RPG (or COBOL) source programs and /COPY modules to be converted. Use the CRTDUPOBJ command if all your source files are in a single migration library, and all of that library is to be converted. Otherwise use the Programming Development Manager (PDM) copy option to copy those source programs to be converted. (The source programs to be converted will come from the QS36SRC file in the migration library or libraries.)
- Create a source file QS36PRC. Copy into this file all of the procedures to be converted.
- Create a source file QS36DDSSRC. Copy into this file all of the display file source formats. These were created by the Migration Aid.

The next source files are all optional, because the PTK creates them if they are not in your conversion library.

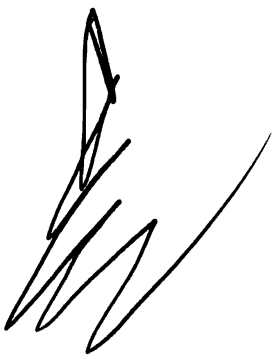
- Create a QRPGSRC source file to hold RPG programs while they are being prepared for compilation on AS/400.
- Create a QLBLSRC source file to hold COBOL programs while they are being prepared for compilation on AS/400.
- Create a QCLSRC file to hold the procedures (OCL) that have been converted to CL programs.
- Create a QDDSSRC file to hold all of the DDS for your files, formats and menus.

3.7.1 Member Types

You will probably want to use PDM to work with the various source programs. For some options, PDM uses the member type of the source program to decide how to operate on the source member. For example, when you use PDM option 14 (Compile), the PDM looks at the type to decide which compiler to use.

Eventually you will want to change the types of the high-level language programs in your new library from RPG36, CBL36, or RPT36 to RPG, CBL, or RPT respectively, so that PDM will use only AS/400 options from then on. However, while using the IBM AS/400 Programmer Tools PRPQ, the member type must continue to be of the form "xxx36" or "xxx38" so that the PRPQ can correctly analyze the members.

You are now ready to start the conversion process. All further work will be done in the new library. You will not need the migration library again, unless you need to recopy the source for some reason. Remove the migration library from your library list.



4.0 Analyzing Files and Fields

In this chapter we describe how to convert System/36 Environment files into AS/400 externally described database files. We also describe some problems you could encounter and solutions to these problems.

Read this entire chapter before starting your conversion on AS/400.

We recommend that you use the PTK to create AS/400 DDS. It provides a high level of assistance and automates many of the key functions. With PTK, you can efficiently analyze and process your OCL and HLL source files to get file descriptions. The PTK produces working DDS significantly faster than can be done by hand. Of course you can also create the DDS without the PTK using normal AS/400 functions.

This chapter includes some PTK output as examples. Use these examples for general guidance only. Refer to the PTK documentation for detailed operational instructions and formats. You can print the documentation by selecting option 10 on the Programmer Tools Main Menu. To print the documentation you must be enrolled in the distribution directory. Use the WRKDIR command to be enrolled.

4.1 General File Considerations

One of the biggest problems of a conversion is defining the files and fields used in different programs. On System/36 you have only program described files. Therefore, different programs can have different descriptions of the same file. Also, System/36 programs can process alphabetic data as numeric. Neither of these is allowed with externally described database files on the AS/400.

Here are some other considerations:

- File naming must also be considered. On System/36 and in the System/36 Environment, you can name a disk file, for example, "Z.FILE". On the AS/400 you cannot use this file in a native COBOL program, because the "." (period) is treated by the COBOL compiler as "end of statement". If you code the following, the compile will fail:

  ```
  COPY DDS-ALL-FORMATS OF Z.FILE.
  ```

  In this case, consider changing names of the form "Z.FILE" to "ZPFILE", substituting a valid character for the period.

- On System/36 and System/36 Environment, you can name a disk file, for example, "#FILE". Again, on AS/400 you cannot use this file within a native COBOL program because COBOL doesn't support the "#" (pound sign). If you must use these kinds of names, use the OVRDBF command to access this file within a COBOL program.

- In System/36 Environment you can name a disk file, for example, "DATAFILE1". On the AS/400 you cannot use this file in a native RPG program because of the length of the file name. RPG/400 only supports 8 characters in a name.

- In a System/36 program you can have an internal name (on the file specification) that differs from the external name of the file (on the disk).

This is not allowed with externally described files; the names must be the same at compile time or the program will not include the correct description of the record formats from the previously created file.

Therefore, the name given to the file when it was created must be put into the F-specification of the RPG program. This is not a problem unless you have made it common practice to make external file names user- or workstation-dependent:

```
?WS?FNAME    or   FNAME?USER?
```

These names are not valid for an external file. Use a valid external name, for example "WSFILE", for such a file. Then use the same name, "WS FILE", as the internal name in the program. At run time you can link the name "WSFILE" with the name of the disk file with an override statement like

```
OVRDBF FILE(WSFILE) TOFILE(AS400LIB/W3FNAME) MBR(W3)
```

Refer to the discussion of ?WS? in Appendix E, "Table of System/36 Substitution Expressions" for ways to define a variable whose value is the workstation ID.

4.2 File Conversion Functions

The second release of the PTK has two function for getting DDS from System/36 Environment files:

1. Creating DDS for System/36 Environment files as they are currently known to the system (new PTK function).
2. Creating DDS for System/36 Environment files as they are known by programs and procedures.

How to work with the PTK is shown step-by-step in 4.3, "Creating DDS from System/36 Environment File Descriptions" on page 21 and 4.4, "Retrieving the Descriptions of Program Described Files" on page 24. These sections show most of the screens used by PTK.

4.2.1 Input to the First Function

For this function, PTK creates DDS based on the current descriptions of your files. As an alternative, you can use the DSPFD and DSPFFD commands on the AS/400 and then create the DDS from this information.

The resulting DDS can be compiled and the data can be copied into the externally described database files. This is a very simple way to create native AS/400 files, but it does not result in files that are useful to your native programs. While the files are externally described, the descriptions probably do not include any useful field names.

In spite of it's limitations, this function can be helpful if you have lost your DDS Source from any externally described files. You can use this function to recreate the DDS source.

There are some things that are not handled correctly. For example, the function always gives you a source file member type DSPF. (Simply change the source type of the resulting source file to PF or LF, whichever is appropriate.) The function does not handle logical files very well, because it recognizes only the

first record format name. But it does give all the names and the lengths of your fields.

Input to this function is the files for which you want DDS generated.

4.2.2 Input to the Second Function

For this function, PTK uses your HLL source statements along with procedures to create DDS. First, it retrieves the necessary information. Then, with your input, it analyzes the files and fields, and finally creates the DDS statements.

Each retrieval is identified by the PTK work file member name. The run member name is a run identification which you will have to remember afterward. Input to the retrieval step is from up to two files:

- A (mandatory) file containing RPG or COBOL source programs. This will be QS36SRC if you have followed the instructions in 3.0, "Getting Started with the Conversion" on page 13.
- A file containing the procedures that load and run the HLL programs. This file is optional, but its inclusion will help the later steps to match names on the file specifications with external names on the FILE OCL statements. This file will be QS36PRC in your conversion library.

The PTK will accept input from any file in any library. The default is to look for HLL source modules in QS36SRC and all OCL in QS36PRC. Since our source and procedures were moved to these files during 3.0, "Getting Started with the Conversion," all you need to do is fill in the appropriate library names.

4.2.3 Rerun Options

Note that all output from the retrieval step is placed in the QPTK library - not the library where the source files are.

Also note that once you have run a retrieval step and created the work files, you cannot run another retrieval step with the same "RUN MEMBER" name. This run member name is a run identification. However you can run a retrieval with a new run member name over the same set of files as a previous run and run a new retrieval over a new set of files.

4.3 Creating DDS from System/36 Environment File Descriptions

This section shows you how to use the PTK to get the S/36 file descriptions as they are known by the system. This step is optional because it does not effect the conversion at all, but it does give you some useful information.

```
MAIN                              AS/400 Main Menu
                                                          System:   RCHAS008
Select one of the following:

     1. User tasks
     2. Office tasks
     3. General system tasks
     4. Files, libraries, and folders
     5. Programming
     6. Communications
     7. Define or change the system
     8. Problem handling
     9. Display a menu
    10. User support and education

    90. Sign off


Selection or command
===> strptk

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=User support
F23=Set initial menu
```

Enter STRPTK. STRPTK sets QPTK as the product library and displays the Programmer Tools Main Menu.

```
 PTMENU          Programmer Tools Main Menu

Select one of the following:

     1. Generate DDS from user applications
     2. Verify decimal data
     3. Journal commands
     4. Analyze changed data
     5. Convert applications
     6. Create remote object
     7. Track application development process

    10. Print Programmer Tools documentation


Selection or command                          (C) COPYRIGHT IBM CORP. 1990
===> 5

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support      F16=System main menu
```

Select option 5 (Convert applications). This displays the Convert Applications menu, which is new in the second release of PTK.

```
 QCVMNU1                      Convert Applications

Select one of the following:

     1. Convert S/36 OCL to AS/400 CL
     2. Convert S/36 RPG II to AS/400 RPG III
     3. Convert S/36 screen formats to AS/400 DDS
     4. Convert S/36 menu files to AS/400 menus
     5. Create DDS from data file descriptions


Selection or command
===> 5

F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel
F13=User support        F16=System main menu
```

Select option 5 (Create DDS from data file description).

```
                          Create DDS (CRTDDS)

Type choices, press Enter.

File . . . . . .    Z.DRD30      Name, generic*, *ALL
  Library . . . . . .  CONLIB     Name, *LIBL, *CURLIB
Source File . . . . . .  QS36DDSSRC  Name
  Library . . . . . .  CONLIB     Name, *CURLIB
Submit to batch . . . . . .  *NO  *NO, *YES

                                                              Bottom
F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
F24=More keys
```

Type the file name and it's library for which you want DDS created. Also type the names of the source file and library where you want the resulting DDS stored. We recommend that you use the QS36DDSSRC source file, not the QDDSSRC. Do not mix System/36 descriptions with native descriptions.

After you press Enter, PTK collects all of the information about the file and creates the DDS for you. After this step, it returns to the menu QCVMNU1.

Next you can use the Programming Development Manager (PDM) or Source Entry Utility to examine the DDS source member that was created. The next screen shows a simple example.

```
Columns . . . .:   1  71              Browse      CONLIB/QS36DDSSRC
Find  . . . _____________________________________Z.STD15
FMT A* .....A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
       ********** Beginning of data *************************************
0001.00      A* Z.STD15    CONLIB     900307
0002.00      A          R Z@STD15
0003.00      A            F00001       128A  B
       ****************** End of data *****************************************


F3=Exit           F5=Refresh          F10=Top           F11=Bottom
F12=Cancel        F13=Change defaults                   F24=More keys
                            (C) COPYRIGHT IBM CORP. 1981, 1989.
```

This screen shows you a migrated System/36 file as the file is known in System/36 Environment. This file has no index, so it has only one field. This field represents the entire record length.

The next example is a little more complicated:

```
Columns . . . .:   1  71              Browse      CONLIB/QS36DDSSRC
Find  . . . _____________________________________Z.DRD30
FMT A* .....A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
       ********** Beginning of data *************************************
0001.00      A* Z.DRD30    CONLIB     900307
0002.00      A                                    UNIQUE
0003.00      A          R Z@DRD30
0004.00      A            F00001        1A  B
0005.00      A            K00001        8A  B
0006.00      A            F00002      247A  B
0007.00      A          K K00001
       ****************** End of data *****************************************


F3=Exit           F5=Refresh          F10=Top           F11=Bottom
F12=Cancel        F13=Change defaults                   F24=More keys
                            (C) COPYRIGHT IBM CORP. 1981, 1989.
```

This screen shows you a System/36 Environment file as the file is defined to the system. This file has an index, so it has three fields. These fields represents the entire record length. The DDS also indicates that the file has a UNIQUE Index and that K0001 is the key field.

Print this information to have it ready in the next conversion step.

4.4 Retrieving the Descriptions of Program Described Files

This section shows how to get descriptions of program-described files using the PTK.

You can run the commands either interactively or in batch, and there are considerations for both. If you run in batch mode, auto report members will be expanded to capture the interim source, and then analyzed. If you run interactively, the auto report members will not be expanded, but will be analyzed just like a normal RPG member. The program will then send a completion message to the requester indicating that auto report members were found but not expanded. Also, source members with subtypes not containing the

characters 36 or 38 will be analyzed, but their copy book statements will not be processed, because PTK cannot determine what syntax to use.

To run the retrieval step interactively, do the following:

1. Type STRPTK on the command line. The command sets QPTK as the product library and displays the PTK main menu.

```
 PTMENU          Programmer Tools Main Menu

Select one of the following:

     1. Generate DDS from user applications
     2. Verify decimal data
     3. Journal commands
     4. Analyze changed data
     5. Convert applications
     6. Create remote object
     7. Track application development process

    10. Print Programmer Tools documentation


Selection or command                          (C) COPYRIGHT IBM CORP. 1990
===> 1

F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel
F13=User support        F16=System main menu
```

2. Take option 1 (Generate DDS from user applications).

```
 PTMENU2         Generate DDS from User Applications

Select one of the following:

     1. Retrieve S/36 file description
     2. Analyze S/36 file description
     3. Analyze S/36 field description

    10. Remove Programmer Tools member


Selection or command
===> 1

F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel
F13=User support        F16=System main menu
```

3. Take option 1 (Retrieve System/36 file description).

```
                  Retrieve S/36 File Description (RTVS36FD)

Type choices, press Enter.

Run Member . . . . . .    KMP030101
Source member file . . .  QS36SRC      Name
  Library  . . . . . .      CONLIB     Name, *LIBL, *CURLIB
Procedure member file  . . QS36PRC     Name, *NONE
  Library  . . . . . .      CONLIB     Name, *LIBL, *CURLIB

                                                                  Bottom
F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
F24=More keys
```

4. Enter a meaningful "Run member" name. Remember this name; you will use it later.
5. Enter the names of the source file and the library containing your RPG, COBOL and Copy members.
6. Enter the names of the source file and the library containing your OCL procedures.

Note that you also can get to this screen by typing the command RTVS36FD and pressing function key F4. Or you can start the command directly by typing:

```
RTVS36FD KMP030101 CONLIB/QS36SRC CONLIB/QS36PRC
```

PTK will collect file-related information from your procedures and programs and match it together. If you only need the information about how your files are described within your program, then you only have to type in your source file that contains the programs.

During this step PTK creates some objects that will be used in the next conversion step.

To run the retrieval step in batch, do the following:

1. Type STRPTK. (The command sets QPTK as the product library.)
2. Type SBMJOB and press F4.
3. Type RTVS36FD and press F4.
4. Enter a meaningful "Run member" name. Remember this name; you will use it later.
5. Enter the names of the source file and the library containing your RPG, COBOL and Copy members.
6. Enter the names of the source file and the library containing your OCL procedures.

If you have very large application, you can get an error message.

```
CPA5305. Record not added. Member "run member" is full.
```

Note the file named in the second-level message text is QATKIIL2. This is a logical file related to the physical file QATKIFP. The message occurs because the file QATKIFP was defined with SIZE(10000 1000 3). That is, the initial size of the file is 10000 records, and it can be extended by 1000 records three times. To eliminate the problem use the following command.

```
CHGPF FILE(QPTK/QATKIFP) SIZE(*NOMAX)
```

4.4.1 Identifying the Files

The file identification step matches the file names as known in the programs (internal names) with the file names on disk (external names).

The internal names are collected from the HLL file specifications and the NAME parameter of the FILE OCL statement. The external names are found in the LABEL parameter of the FILE OCL statement, and in the System/36 Environment files library.

4.4.2 Matching Internal and External Names

The next screens show you how to use PTK to associate the internal and external file names.

After entering STRPTK to display the PTK Main Menu, select option 1 to display the menu shown below.

```
 PTMENU2        Generate DDS from User Applications

Select one of the following:

     1. Retrieve S/36 file description
     2. Analyze S/36 file description
     3. Analyze S/36 field description

    10. Remove Programmer Tools member


Selection or command
===> 2

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support      F16=System main menu
```

Select option 2 (Analyze S/36 file description) to display the next display.

Alternatively, you could type the ANZS36FD command on a command line and press F4.

```
                  Analyze S/36 File Description (ANZS36FD)

Type choices, press Enter.

Run Member . . . . . .  KMP030101

                                                                  Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Type in the run member name you want to analyze. If you cannot remember the run member name, press function key F4. PTK shows you a list of the created run members.

When you press Enter, the PTK performs two steps. First, the information gathered in the retrieval step is used by the PTK to match, where possible, the internal and external names.

Second, the result of the matching is displayed as in the following screen so that you can type the external names for those files which the system could not match, that is, for those files for which the system could not deduce any external name.

```
EXTINT                    Work with External/Internal Files

Run member . . KMP030301                    Position to . . . . . _
File library . QS36F

Type options, press Enter.
  2=File resolution    3=Format resolution

Opt External  Internal   Procedure  Program    Rcdln M  KS   KL  Status
 2  Z.DRD30   DRD30      STP965     STP965       256 1   2   16
    Z.DRD30   DRD30      TSP60      TSP60        256 1   2   16
 _  Z.STD15   STD15      STP965     STP965       128 1   2
 _  Z.STD25   STD25      STP965     STP965       128 1   3   20
 _  Z.STD3T8  STD30      TSP60      TSP60        256 1   2
 _  Z.STD30   STD30      STP965     STP965       256 1   2    6
 _  Z.STD50   STD50      STP965     STP965       256 1   2   14
    Z.STD50   STD50      TSP60      TSP60        256 1   2   14
 _  Z.STD55   STD55      STP965     STP965       128 1   2    6
    Z.STD55   STD55      TSP60      TSP60        128 1   2    6



F3=Exit          F5=Refresh          F7=Rolldown         F8=Rollup
F10=Top          F11=Bottom          F14=Sort(program)   F15=Set external
```

Enter an external file name for those files for which PTK could not deduce the external name. Do *not* type over a name in the external file name if the PTK has shown this name. If you do so, you can not run the next conversion step of the PTK successfully.

External names cannot be deduced when substitution OCL is used, either for the NAME and LABEL parameters of the FILE OCL statement, or for LOAD OCL statements. For example:

```
// FILE NAME-DRD30, LABEL-DRD30?WS?
```

This next example shows some of the situations that can occur when resolving file names.

```
EXTINT                    Work with External/Internal Files

Run member . . KMP03010 1                    Position to . . . . .   _
File library . QS36F

Type options, press Enter.
  2=File resolution    3=Format resolution


      Opt External  Internal  Procedure  Program  Rcdln M  KS  KL  Status
Note 1            INTF4     PROCA      PROG01    132  0
Note 2    ?       INTF5     PROCA      PROG01    132  0
Note 3    ACCOUNG ACCOUNG             TRANSACT   88  0   3  14
          ALTMAST ALTMAST   CBLBNK43   CBNK43   226  0  43  30  Resolved
Note 4    ALTONE  ALTONE    PROCJ      PROG10    132  0   2   2
          A.FIRST INTF1     PROCH      PROG08    132  0
          A.SECOND INTF2    PROCH      PROG08    132  0
          A.THIRD INTF3     PROCH      PROG08    132  0
Note 5    FIVE    INTF2     PROCN      PROG66    132  0
```

Figure 1. PTK File Resolution Screen

The entries without notes are good - the HLL, the external and internal names, and the record length (Rcdln) were found properly. The program field will never be blank. Entries will be shown only if they can be tied directly to an HLL source member.

Note 1 The external file name was not set by PTK. This means that either the analyze OCL phase was not done or that PTK did not find a FILE OCL statement containing the name of that file. In the example, the procedure name is listed, so the OCL analysis was done, but a FILE OCL statement could not be found with NAME-INTF4. One common reason that the PTK cannot find a FILE OCL statement is that the statement is not between the LOAD OCL and the RUN OCL statement. Procedures must be of the following form to ensure the the PTK will find the external file name.

```
/* PROC1
// LOAD PROG
// FILE NAME-FNAME, LABEL-FLABEL
// RUN
```

In some cases it might be worth the additional time to check your procedures for this format, because it will save time during the file name resolution step.

Note 2 The external file name was indicated by a "?" (question mark). This means that PTK found a FILE OCL statement, but the external file name could not be determined due to substitution expressions.

Note: Do not try to resolve an external file name containing a "?" (question mark). A file name containing a "?" is an invalid file name. When the resolution is attempted, the PTK will cancel. To solve the problem, change your procedure and begin again with Option 1 (Retrieve S/36 file description) of menu PTMENU2 (Generates DDS from User Applications).

```
Old OCL Statement = // FILE NAME-DRD30, LABEL-DRD30?WS?
New OCL Statement = // FILE NAME-DRD30, LABEL-DRD30WS
```

Note 3 The procedure name was not set by PTK. This means that either the analyze OCL phase was not done, or no procedure was found that loaded the HLL program. In the example, the external file name is shown. It was entered manually, since it could not be determined without a procedure.

Note 4 File ALTONE is an example of an alternate index. Later during field resolution (see -- Heading 'PTKRES' unknown --) you cannot resolve the field names because there are none for an alternate index, but the PTK will create the DDS for a logical file for it.

Note 5 PROG66 has had the internal name INTF2 matched against the disk label FIVE. File INTF2 is also listed within the file group A. But in fact different files were intended. Be especially careful when resolving file names if multiple programs use the same internal name for different files.

At this time you should consider naming conventions for both the (internal) names used by the programs and the (external) names in the LABEL parameters of the FILE OCL statements.

Additional considerations for external names include:

1. The names you choose for the external files should be the names you use when creating the files on disk. At this stage it should be a name of an existing file so PTK can pick up information from the external file. For group file naming recommendations, refer to 15.11, "Group Files" on page 150.

2. It is particularly important to type in the external names for those internal files whose external names are blank, like INTF4 alongside note 1 in Figure 1 on page 29. If you do not, the following steps will not bring the field descriptions for INTF4 in PROG01 forward for later comparison with the other, possibly different field descriptions held in other programs. This will make the task of selecting the correct fields for the DDS harder.

   In other words, if you suspect that there are different descriptions of the same file (field names, field length and attributes) in several programs, assign new external names to those files for each program. Then you can resolve those differences later on during the "Analyze S/36 Field Description" step (refer to 4.6, "Resolving Field Names" on page 34). One way to do this is to change your procedures, specifying different names for the LABEL parameter in the FILE OCL statements. (Refer to the note at the end of this list.)

3. A given external file may appear several times on the list, once for each program in which it appears. Files with external names are grouped together in alphabetical order at the bottom of the list.

4. As you enter each external name, the line with that name is placed at the bottom of the list in alphabetical order, while the files with no external names remain at the top of the list.

5. To attach an external name to a single internal name, type the external name in the left-hand column and press F5 to rebuild the screen. Note that just pressing the Enter key does not gather all occurrences of the same external file name together.

6. Do not try to resolve a file with an external name that is not valid, for example, one containing question marks.

Note: If PTK finds an external name for the file, do not change that name, because it is directly pointed to in the next conversion step.

Once the files you want to resolve have external file names, type in 2 for all files you want to resolve. PTK will display the following screen:

```
FILRSLV                  Work with File Resolution

External file        Z.DRD30

Enter/Update resolved definitions, press Enter.
                          Program       Resolved       System
                        Specification   Definition    Information

File type                     K             K             K
Record length               256           256           256
Multiple formats            Yes

                                                    Unique keys
Key start-1                   2             2             2
Key length-1                 16             8             8



F6=Accept          F12=Cancel
```

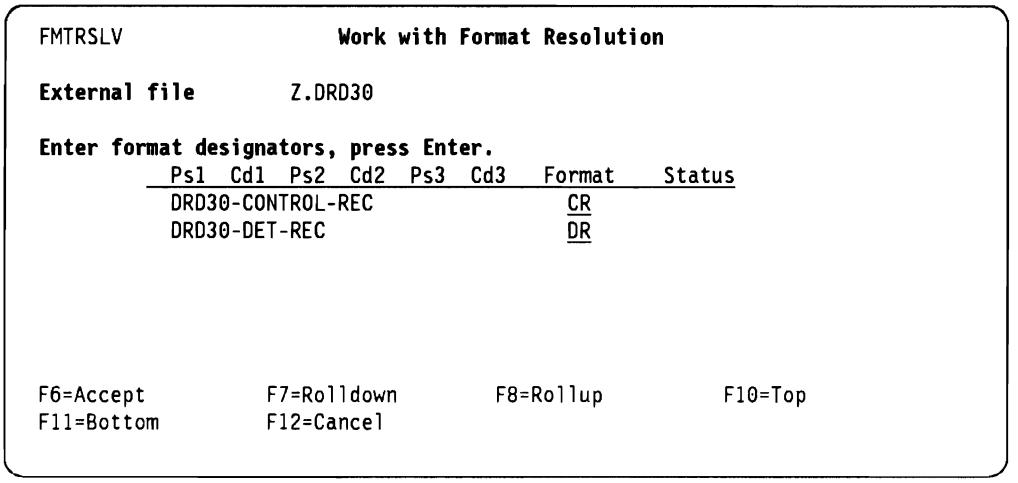
This screen is shown when you work with file resolution.

For this step, you might find it useful to refer to file descriptions that were produced in section 4.3, "Creating DDS from System/36 Environment File Descriptions" on page 21.

If a file is used in more than one program, it will have more than one internal description. The resolution step asks you to ensure that the record length and key position information for the file is correct.

The system compares the different descriptions and presents a "majority rule" under the heading Program Specification. If the file exists on the system, information is shown under System Information, while your choice is entered under Resolved Definition. You do not have to resolve all the files in one step. You can leave some and come back to them at a later, re-running this step.

Verify or correct all the information and press function key F6 to resolve the file. If multiple formats are defined for the file, PTK automatically displays the Work with Format Resolution screen. Otherwise the Work with External/Internal Files screen is displayed.



This screen is shown when you work with format resolution. After a multiple record file has been resolved, you need to resolve the formats in that file. This step simply involves choosing a 2-character identifier that will be added to the end of the file name to give a name to the DDS source that will be created for each physical file. (One physical file is created for each record type in the multiple record System/36 file.) Note that the external names have been restricted to eight characters to allow for this to happen. If you have multiple record formats in your file and are using RPG, you should not use a file name longer than six characters. This is because the file name in an RPG/400 program may be no longer than eight characters.

Sometimes only one format is shown on this screen, even when the previous screens have told you that the file is a multiple-record file. This can happen because the I-specifications for the file may contain a record ID (with no fields or indicators) that is included in the program to catch previously undefined record types, and thus avoid a run-time error.

Type in a two-character identifier to distinguish your record formats. Then press command key F6 to resolve the formats. When finished, PTK displays the Work with External/Internal Files screen.

```
EXTINT                    Work with External/Internal Files

Run member . . KMP03010 1                  Position to . . . . .  _
File library . QS36F

Type options, press Enter.
  2=File resolution    3=Format resolution

Opt External  Internal  Procedure  Program  Rcdln M  KS  KL  Status
 _  Z.DRD30   DRD30     STP965     STP965     256 1   2  16  Resolved
    Z.DRD30   DRD30     TSP60      TSP60      256 1   2  16
 _  Z.STD15   STD15     STP965     STP965     128 1   2
 _  Z.STD25   STD25     STP965     STP965     128 1   3  20
 _  Z.STD3T8  STD30     TSP60      TSP60      256 1   2
 _  Z.STD30   STD30     STP965     STP965     256 1   2   6
 _  Z.STD50   STD50     STP965     STP965     256 1   2  14
    Z.STD50   STD50     TSP60      TSP60      256 1   2  14
 _  Z.STD55   STD55     STP965     STP965     128 1   2   6
    Z.STD55   STD55     TSP60      TSP60      128 1   2   6



F3=Exit           F5=Refresh          F7=Rolldown         F8=Rollup
F10=Top           F11=Bottom          F14=Sort(program)   F15=Set external
```

You can see that file Z.DRD30 is now resolved.

Type in 2 for all other files you want to resolve.

4.5 Field naming considerations

Before you resolve field names, you need to be aware of the kinds of problems that can occur when different programs have different definitions of fields.

On S/36 you can have non-numeric data within a numeric field. This situation (decimal data error) is not allowed on AS/400. Therefore you must know whether a field is or is not numeric. Here is an example that shows how multiple file definitions can lead to decimal data errors. More information about finding and correcting decimal data errors can be found in section 10.0, "Decimal Data Errors" on page 75.

On S/36 you can have the following field definitions in program PROG1 and PROG2.

```
PROG1
       FD FILE1.
       01 FILE1-RECORD.
          05 FIELD-1         PIC 9(5).  (numeric field)  HEX(F0F0F0F0F0)
          05 FIELD-2         PIC X(5).  (alpha-numeric)  HEX(4040404040)

PROG2
       FD FILE1.
       01 FILE1-RECORD.
          05 FIELD-1         PIC 9(3).  HEX(F0F0F0)
          05 FIELD-2         PIC X(7).  HEX(40404040404040)
```

Both files have the same record length, but each file has different field lengths and types. So if PROG2 adds or updates a record, and PROG1 reads that record, PROG1 would find non-numeric characters in the last two characters of FIELD-1. To solve this problem you need to decide which field definition is correct.

In the next example, two programs have defined fields that have the same data type, but different lengths.

```
PROG1
       FD FILE1.
       01 FILE1-RECORD.
          05 FIELD-1          PIC X(5).
          05 FIELD-2          PIC X(5).

PROG2
       FD FILE1.
       01 FILE1-RECORD.
          05 FIELD-1          PIC X(3).
          05 FIELD-2          PIC X(7).
```

We recommend that the file description for the externally described physical file look like this:

```
       FD FILE1.
       01 FILE1-RECORD.
          05 FIELD-1          PIC X(3).
          05 FIELD-2A         PIC X(2).
          05 FIELD-2B         PIC X(5).
```

If necessary, you can create logical files that can be used by PROG1 and PROG2 to preserve their different views of the record. But, if the data types are different, you will have to decide which data type is correct.

Also you should review your field naming conventions. Follow these rules to help ensure consistent, meaningful field names:

- Use only supported characters.
- Limit field names to eight characters. RPG/400 is restricted to eight-character field names. For COBOL you can use the DDS Keyword ALIAS to define field names longer than eight characters.
- Use names that are self-explaining.
- Use the DDS Keyword TEXT to explain the field. PTK does not permit this, but it can be done in a later step.

4.6 Resolving Field Names

This is the next PTK step. Now the PTK has enough information to gather together all the internal field descriptions for each record in a file. During the field name resolution step you will identify the fields needed to generate data descriptions for each AS/400 file.

The following screens show you how to resolve field names using the PTK. Enter STRPTK to display the Programmer Tools Main Menu. Then select option 1 (Generate DDS from user applications) to display the following screen:

```
 PTMENU2          Generate the DDS from User Applications

 Select one of the following:

      1. Retrieve S/36 file description
      2. Analyze S/36 file description
      3. Analyze S/36 field description

     10. Remove Programmer Tools member



 Selection or command
 ===> 3
_
 F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel
 F13=User support        F16=System main menu
```

Select option 3 (Analyze S/36 field description). The following screen is displayed.

```
                  Analyze S/36 Field Description (ANZS36FFD)

 Type choices, press Enter.

 Run Member . . . . . .  KMP030301



                                                                  Bottom
 F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
 F24=More keys
```

Type in the run member name. If you can not remember the run member name, press function key F4. PTK shows you a list of run members.

The next screen allows you to select the file whose fields you want to resolve. You can also display this screen using the ANZS36FFD command. For example:

```
ANZS36FFD KMP030101
```

```
FILSEL                  Select External File          KMP07032

External file . . . _________

Enter name or select item from the list.

   S   File name   Status

   S   Z.DRD30

  F3=Exit        F7=Rolldown    F8=Rollup      F12=Cancel     F21=Select all
```

The "Analyze S/36 Field Description" display first shows a list of resolved file names. In our example, we only resolved one file in the previous step. Therefore there is only one file in the list.

Select the file you want to convert.

If multiple formats are defined for the file you selected, the next screen allows you to select one of the formats to work with.

```
FMTSEL                  Select Format Name            KMP030101

External file . . . Z.DRD30
Format name . . . . _________

Enter name or select item from the list.

   S   Fmt name   Status

       CR
   S   DR

  F3=Exit        F7=Rolldown    F8=Rollup      F12=Cancel     F21=Select all
```

PTK shows you all of the formats within this file. Select the format of the file you want to work with.

PTK now gathers together, from every program associated with the file, the field descriptions for the format you selected. The fields are sorted by start and end positions and presented on the main field resolution screen. You will use this screen to build the DDS for the format. In the best case, every program defines the same fields in the same positions. This could happen, for example, if /COPY modules had been used for all files. If there are overlapping fields, different

definitions for the same record positions, empty record positions, or alternative spellings of the same field, they are shown on the next screen.

```
FLDSEL                    Work with Fields                  KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    44
 Format name . . . . DR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   _   1    1   00 DRD30RECOR DRD30-RECORD-CODE                  2
 K _   2    4   00 DRD30CUSTA DRD30-CUST-AREA                   2
 K _   2    9   00 DRD30CUSTO DRD30-CUSTOMER      Overlap       2
 K _   5    9   00 DRD30CUSTO DRD30-CUST-OTHER    Subfield      2
   _  10   17   00 DRD30CHARG DRD30-CHARGE-TO                   2
   _  18   41      DRD30NAME  DRD30-NAME                       2
   _  42   42      DRD30ALPHA DRD30-ALPHA-CODE                 2
   _  43   52      DRD30ALPHA DRD30-ALPHA-SEARCH  Duplicate     2
   _  53   76      DRD30ADDR1 DRD30-ADDR-1                     2
   _  53  124      DRD30ADDR  DRD30-ADDR          Overlap       2
   _  77  100      DRD30ADDR2 DRD30-ADDR-2        Subfield      2
   _ 101  124      DRD30ADDR3 DRD30-ADDR-3        Subfield      2   +


  F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
  F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

Refer to Chapter 2 in the Programmer Tool Users Guide for a description of the fields and function keys available on this display.

First, fix the field names for the file. As you do this, remember the rules given in section 4.5, "Field naming considerations" on page 33. Change field names so that there are no error messages except Overlap and Subfield.

```
FLDSEL                    Work with Fields                  KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    44
 Format name . . . . DR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   1   1    1   00 RECOR      DRD30-RECORD-CODE                2
 K 1   2    4   00 CUSTA      DRD30-CUST-AREA                  2
       2    9   00 DRD30CUSTO DRD30-CUSTOMER                   2
 K 1   5    9   00 CUSTO      DRD30-CUST-OTHER                 2
   1  10   17   00 CHARG      DRD30-CHARGE-TO                  2
   1  18   41      NAME       DRD30-NAME                       2
   1  42   42      ALPHAC     DRD30-ALPHA-CODE                 2
   1  43   52      ALPHAS     DRD30-ALPHA-SEARCH               2
   1  53   76      ADDR1      DRD30-ADDR-1                     2
   _  53  124      ADDR       DRD30-ADDR          Overlap      2
   1  77  100      ADDR2      DRD30-ADDR-2        Subfield     2
   1 101  124      ADDR3      DRD30-ADDR-3        Subfield     2      +


  F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
  F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

After correcting all error conditions except "Subfield" and "Overlap", select those field names which give a true description of the record format.

For overlapping fields, we recommend that you always select the smallest, or lowest level fields. Generally, this means you should not select a field with the Overlap error message. You can define the other fields with a logical file later in the conversion. When you have finished selecting the fields, exit from the screen.

If no errors are found, and the file definition matches the external file (That is, the record length and key definitions match.), the next screen allows you to generate the DDS for the format.

```
FLDSEL                  Work with Fields              KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    44
 Format name . . . . DR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur


                        Data is approved for DDS
                      1 0. Do not generate DDS
                        1. Generate DDS source member


  F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
  F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

Note that only the DDS will be generated, not the file itself. You can find the DDS in a source physical file in the QPTK library with the "run member" name you have chosen for this run.

The next screen shows that the DDS has been created for format DR, and allows you to select another format.

```
FMTSEL                  Select Format Name            KMP07032

External file . . . Z.DRD30
Format name . . . . _________

Enter name or select item from the list.

   S   Fmt name   Status

   S   CR
   _   DR          Generated




  F3=Exit        F7=Rolldown    F8=Rollup      F12=Cancel     F21=Select all
```

Select record format CR.

```
FLDSEL                    Work with Fields                 KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    10
 Format name . . . . CR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   _   1    9      FILLER___ FILLER                              2
   _  10   12 P 00 RECOR____ DRD30-RECORDS                       2
   _  13   16 P 00 DATEU____ DRD30-DATE-UPDATED                  2
   _  17   22      PROGU____ DRD30-PROG-UPDATING                 2
   _  23  256      FILLER___ FILLER              Duplicate       2




   F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
   F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

Note that this format has no key field definition, yet PTK shows a key start and length at the top of the screen. This format is part of a multiple record format. It will result in a logical file over multiple (two in this case) physical files. All the physical files must have key definitions.

Again, correct the fields to remove errors other than Overlap and Subfield errors. The next screen shows the result.

```
FLDSEL                    Work with Fields                 KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    10
 Format name . . . . CR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   _   1    9      F1_______ FILLER                              2
   _  10   12 P 00 RECOR____ DRD30-RECORDS                       2
   _  13   16 P 00 DATEU____ DRD30-DATE-UPDATED                  2
   _  17   22      PROGU____ DRD30-PROG-UPDATING                 2
   _  23  256      F4_______ FILLER                              2




   F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
   F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

If you have a file with multiple formats where one of the formats has key fields and the other format does not, you cannot create DDS for this format. PTK displays the error message "Key Area not discrete", which means that you have not defined a key area in this in this format. You will get this message when you try to resolve a format with the wrong Index description.

PTK allows you to add a new field. Press function key F9 to add a new field.

```
FLDSEL                    Work with Fields              KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    10
 Format name . . . . CR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   _   1    9      F1         FILLER                            2
   _  10   12 P 00 RECOR      DRD30-RECORDS                     2
   _  13   16 P 00 DATEU      DRD30-DATE-UPDATED                2
   _  17   22      PROGU      DRD30-PROG-UPDATING               2
   _  23  256      F4         FILLER                            2




 Beg End T DP DDS fldnam
 2__ 9__ _ __ KEYFIL_ F12=Cancel
```

Fill in the correct starting and ending position and the name of the field, then press enter. PTK will allow you to enter more fields. Press F12 to finish.

```
FLDSEL                    Work with Fields              KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    10
 Format name . . . . CR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
       1   9      F1         FILLER                            2
   1   1   1                                    Unassigned
 K 1   2   9      KEYFIL     KEYFIL
   1  10  12 P 00 RECOR      DRD30-RECORDS                     2
   1  13  16 P 00 DATEU      DRD30-DATE-UPDATED                2
   1  17  22      PROGU      DRD30-PROG-UPDATING               2
   1  23 256      F4         FILLER                            2


   F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
   F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

PTK now shows that an unassigned field has been added. That field is added automatically to resolve the record length once the key field was added.

In our example, we select the unnamed field instead of field F1, so that we do not have overlapping fields. Enter a name for the unassigned field, select the other fields that represent the record, and press enter. The screen that follows shows the selected fields and their names.

```
FLDSEL                    Work with Fields              KMP07032

 External file . . . Z.DRD30          Key str/len . .    2 /   8   #
Rcds    10
 Format name . . . . CR               Record length . . .    256

 Select fields for DDS or enter new items.

 K S Beg  End  T DP DDS fldnam Internal field name Error msg  Occur
   1   1    1      F1_______ F1
       1    9      F1_______ FILLER                              2
 K 1   2    9      KEYFIL___ KEYFIL
   1  10   12 P 00 RECOR____ DRD30-RECORDS                       2
   1  13   16 P 00 DATEU____ DRD30-DATE-UPDATED                  2
   1  17   22      PROGU____ DRD30-PROG-UPDATING                 2
   1  23  256      F4_______ FILLER                              2


   F3=Exit        F5=Refresh     F7=Rolldown    F8=Rollup      F9=Add item
   F10=Top        F11=Bottom     F12=Cancel     F13=Fast
```

Now press function key F3 to leave this screen and create the DDS. The resulting screen confirms that the DDS was generated.

```
FMTSEL                   Select Format Name             KMP07032

 External file . . . Z.DRD30
 Format name . . . . _________

 Enter name or select item from the list.

    S   Fmt name    Status

    _   CR          Generated
    _   DR          Generated


   F3=Exit        F7=Rolldown    F8=Rollup      F12=Cancel     F21=Select all
```

PTK shows you that the DDS was generated for both formats.

Resolution can now be repeated for other files and formats. While this example showed resolution of only one file, you can resolve more than one file at a time.

Now copy the DDS source file from library QPTK to your conversion library, renaming your "run member" source file to QDDSSRC.

4.7 How to get the information without the PTK

If you want to convert your System/36 Environment files without the PTK, you can use Query functions to collect information about all your program source. You can also use the PDM search function to find the information. Or you can write a program that gives you the information.

The following example shows how to get your internal file names from a COBOL program with QUERY:

- Use the STRQRY to start AS/400 Query.
- Select the query option "Specify file selection".
- Select the source file (for example, QS36SRC).
- Select the source file member (for example, STP965).
- Select the query option "Select records".
- Type the following definition for COBOL:

```
AND/OR  Field          Test   Value (Field, Number, or 'Characters'
        SRCDTA_______  LIKE_  '%SELECT%'_____________________
 ____   _____________  _____  ____________________________________
```

- Press F5 to get the report.
- Type the following definition for RPG:

```
AND/OR  Field          Test   Value (Field, Number, or 'Characters'
        SRCDTA_______  LIKE_  '%DISK%'_______________________
 ____   _____________  _____  ____________________________________
```

- Then press F5 to get the report.

The report lists all the internal file names used in this program. You also can put the report into a Database file for later use, or you can print the report. Do this with every source member. If you search in the procedures you can use the "LABEL" as search word. This give you all the external file names.

However, you must collect the information from your procedures and source code and match them together.

4.7.1 Implications

Smaller fields, such as day, month, and year, can be concatenated together through a logical file definition to give larger fields such as DATE. Refer to the *Data Description Specifications Reference* manual for details. Note, however, that there are restrictions on the use of concatenated fields in COBOL/400 programs. For more detail refer to 12.0, "COBOL Considerations" on page 111.

5.0 Convert System/36 Environment Formats to Native Formats

Display format conversion is one of the easier parts of application conversion. You can get the DDS for the native display file in several ways:

- Using the System/36 to AS/400 migration aid
- Using the PTK (Selecting option 5 from the PTK Main Menu)
- Using Screen Design Aid (SDA)
- Creating display files with the CRTS36DSPF command.

You always will find the created DDS in either the QS36DDSSRC or QDDSSRC source file.

Producing DDS is not a problem on AS/400. But there are some differences between System/36 Environment and AS/400 native mode:

- System/36 Environment display files use display services which behave like those of System/36.
- AS/400 native display files use display services which behave like those of System/38.

You might see some differences when you are using a display format in both a System/36 Environment program and a native program. For detailed information about the differences refer to the Appendix F of the *Data Management Guide*.

The next problem is that you have different programming techniques between S/36 and AS/400. Normally nobody uses the "variable start line number" technique to display a list in AS/400 native mode. Instead, use a subfile. If you want to convert a program using such techniques you must redesign the application.

The main problem of screen conversion is that you can have two or more programs in the System/36 Environment using the same DSPF. These programs might have different DSPF field names, but when you are using externally defined display files, you can only have one field name defined. This means that you have to look very carefully through your programs and make the necessary changes in the programs and DDS of the display file.

5.1 Finding All Programs That Use the Same Display File

Programming Development Manager (PDM) provides you a search function. Using this function it is possible to find strings within source mem bers. The next screens show how to find out which of your programs use the same display files.

If you followed the recommendations in Chapter 3.0, "Getting Started with the Conversion" on page 13 you will find your program source in the QS36SRC source file in your conversion library (CONLIB in this example).

Use the PDM functions or the WRKOBJPDM command, for example:

WRKOBJPDM CONLIB QS36SRC

to get the following display:

```
                        Work with Objects Using PDM

Library . . . . .  CONLIB      Position to . . . . . .
_________
                               Position to type . . . . .  _

Type options, press Enter.
  2=Change       3=Copy         4=Delete       5=Display      7=Rename
  8=Display description         9=Save         10=Restore     11=Move ...

Opt  Object      Type        Attribute   Text
25   QS36SRC     *FILE       PF-SRC

                                                                    Bottom
Parameters or command
===> _
F3=Exit          F4=Prompt           F5=Refresh          F6=Create
F9=Retrieve      F10=Command entry   F23=More options    F24=More keys
This is a subsetted list.
+
```

Type option number 25 (Find string) in front of the source file and press Enter.

```
                              Find String

Type choices, press Enter.

  Find . . . . . .  STP965FM

    From column number . . . . . .  1        1 - *RCDLEN
    To column number . . . . . .    *RCDLEN  1 - *RCDLEN

    Kind of match  . . . . . .  2           1=Same case, 2=Ignore case

  Option . . . . . .  *NONE     *NONE, Valid option
    Prompt . . . . . .  N       Y=Yes, N=No

  Print list . . . . . .  Y     Y=Yes, N=No

  Find string in batch . . . . . .  N     Y=Yes, N=No

  Parameters . . . . . .  _____________________________________________

F3=Exit              F5=Refresh       F12=Cancel       F16=User options
F18=Change defaults
```

Type in the name of the display format for which you are searching, specify *NONE for the Option prompt and Y for the Print list prompt, and press Enter to start the search.

The list of members containing the string is printed and can be found in your job's output queue (output queue CONLIB in library CONLIB in this example). Use the WRKOUTQ command to display your output queue. The resulting display is shown next:

```
                          Work with Output Queue

Queue:   CONLIB         Library:   CONLIB         Status:   RLS

Type options, press Enter.
  2=Change   3=Hold   4=Delete   5=Display   6=Release   8=Attributes

Opt  File        User        User Data   Sts   Pages  Copies  Form Type   Pty
 5   QPUOPRTF    ITSCID13                RDY       1       1   *STD    5

                                                                      Bottom
Parameters for option 2 or command
===> ____________________________________________________________________
F3=Exit   F11=View 2   F12=Cancel   F22=Printers   F24=More keys
```

Type option 5 (Display) next to the entry containing the list of members and press Enter to display it.

```
                          Display Spooled File
 File  . . . . . :   QPUOPRTF                        Page/Line   1/2
 Control . . . . .   _____                           Columns
1 - 78
 Find  . . . . . .   ____________________________
 *...+....1....+....2....+....3....+....4....+....5....+....6....+....7....
+...
 Q5728PW1   R02 M00 891006            Programming Development Manager - Membe
 File  . . . . . :   QS36SRC
   Library . . . . . :   CONLIB
 Member  . . . . . :   *ALL
 Type  . . . . . :   *ALL
 Find  . . . . . :   STP965FM
 From column . . . . . :   1
 To column . . . . . :   *RCDLEN
 Kind of match . . . . :   2       1=Same case, 2=Ignore case
 Number of matches . . :   1
                        Creation  Last Changed                Deleted
  Member      Type      Date      Date      Time      Records  Records  Text
  ----------  ----------  --------  --------  --------  -------  -------  -----
  STP965      CBL36     03/06/90  03/12/90  11:31:08  02677    00000    Mainp
                                     * * * * *  E N D  O F  L I S T I N G

                                                                      Bottom
 F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys
```

Within this list you find all of the programs that use your defined display file. In our example, we only have one program using the DSPF. This is only one of the ways to get the information. But it is a fast way to find the display file name in a potentially large number of source programs.

5.2 Changes Required to Use Externally Described Display Files

At this stage we have DDS for the display files and we know which programs are using these files. But the programs still contain the internal description of the display file. These descriptions may still contain different names for the fields or different field attributes.

The next step is to check the internal display file definitions against the external file definition. To get the external description use the DSPFFD command or look into the DDS source. To get the internal definition you must look through each program listed in the search list obtained in the previous step. You can do this by browsing through each program. If you find differences between the internal and the external display fields you, should note them. You should also note whether there are differences between the programs. If there are differences, you must change either the programs or the display file DDS.

But don't change the programs yet. Conversion of programs is discussed later in separate sections for COBOL and RPG:

- 12.0, "COBOL Considerations" on page 111 for COBOL programs
- 11.0, "RPG Considerations" on page 79 for RPG programs

In most cases you will have to change your programs because they use different field names within the programs. The worst case is when you have different field descriptions. For example, one program might use a field as a numeric field, and another program might use the field as an alpha-numeric field. In such a case you must decide how to use the field in both programs. Perhaps you will decide that a field should be an alpha-numeric field within the program. Later in the conversion procedure you must remember this, because you cannot use alpha-numeric data within a calculation, or move it easily to a numeric field.

Remember, changes to your programs are discussed later. So, at this point in the conversion, limit your changes to those that can be made to the DDS. For example, if all programs are using a field as an alpha-numeric field, but the DDS defines the field as numeric field, you should change the DDS so that the field is defined as an alpha-numeric field.

6.0 Converting Menus

This chapter first discusses the building blocks of both the System/36 Environment and the native menus. It then describe how to create native menus with and without using PTK.

6.1 Menu Considerations

When a System/36 menu is migrated, the result is two members with type MNU36 in the source file QS36SRC in the specified library:

- One member, whose name includes the suffix "##", contains the menu commands.
- One member, whose name includes the suffix "DT", contains the $SFGR definitions of the menu layout.

When you create the menu (CRTS36MNU command), a new source member with type DSPF is created on source file QS36DDSSRC in your library. Two new objects are also created in your library:

- One object, with type MSGF, is created by $MGBLD. The name of the object includes the suffix "##".
- One object, with type *FILE and attribute *DSPF, is created by $BLDMNU.

To create a native menu, you must have the following source members:

- One source member with type MNUDDS. This source member contains the menu layout.
- One source member with type MNUCMD. This source member contains the menu commands. The name of this member must contain the suffix "QQ".

After creating the native menu (CRTMNU command), the following objects are created in your library:

- One object type *MSGF (The name of the file contains the suffix "QQ").
- One object type *FILE, attribute DSPF
- One object type *MENU, attribute DSPF.

6.2 Creating a Native Menu with PTK

PTK converts System/36 Environment menus to native menus. This function is useful if you need to convert the menus very quickly, and you know that your menus call only user-written procedures (not System/36 Environment functions, such as LISTLIBR). If the menus call the System/36 Environment function, you must look very carefully at the converted menu commands. If you find functions that are not converted correctly, you must convert the menus that contain them manually.

One shortcoming of PTK is that it does not create DDS when converting System/36 Environment menus to native menus. This means that if you want to modify converted menus, you must change the System/36 Environment source

members. Then, after the modification, you must convert the menus again. The easiest way to modify the menu is to use the System/36 Environment Screen Design Aid, but again, you can only change the old menu layout and command statements. You can not modify the new created CL menu command object. Therefore you must rerun the conversion after every modification. If you have menus that will be updated often, you should create them using the native screen design aid functions.

In spite of this shortcoming, PTK offers a quick, easy way to convert your menus. After the creation of your Menus, you can use the native command.

```
GO MENU
```

PTK needs the System/36 Environment menu objects, so you must create them or copy the objects from your original library. In our conversion examples we created the necessary objects again. The necessary menu objects are:

- *MSGF (created by $MGBLD)
- *FILE attribute DSPF (created by $BMENU).

PTK creates new objects for native menus out of the objects, and also creates a new object of type *MENU with attribute *DSPF, and a list of changed commands.

The following screens show how to create a native Menu with the PTK:

First, create the necessary menu objects in your conversion library. Type CRTS36MNU and press F4 to display the following screen:

```
                    Create S/36 Menu (CRTS36MNU)

Type choices, press Enter.

Command text source member## . .   kmpcbl##      Name (## required)
Option text source member  . . .   *NONE         Name (menuDT), *NONE
Command text source file . . . .   QS36SRC       Name
  Library  . . . . . . . . . . .     *CURLIB     Name, *CURLIB
Option text source file  . . . .   QS36SRC       Name
  Library  . . . . . . . . . . .     *CMDLIB     Name, *CMDLIB, *CURLIB
Menu library (LOADLIB) . . . . .   *CMDLIB       Name, *CMDLIB, *CURLIB
Replace menu . . . . . . . . . .   *YES          *NO, *YES
Free form menu . . . . . . . . .   *NO           *NO, *YES
Keep option text msg file  . . .   *YES          *NO, *YES


                                                                    Bottom
F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display        F24=More keys
```

Type in the text source member name and press Enter to create the menu.

Next, use the STRPTK to display the PTK main menu:

```
 PTMENU          Programmer Tools Main Menu

Select one of the following:

     1. Generate DDS from user applications
     2. Verify decimal data
     3. Journal commands
     4. Analyze changed data
     5. Convert applications
     6. Create remote object
     7. Track application development process

    10. Print Programmer Tools documentation



Selection or command                          (C) COPYRIGHT IBM CORP. 1990
===> 5
_
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support      F16=System main menu
Type option number or command.
```

Select option 5 (Convert applications) to display the next screen.

```
 QCVMNU1                      Convert Applications

Select one of the following:

     1. Convert S/36 OCL to AS/400 CL
     2. Convert S/36 RPG II to AS/400 RPG III
     3. Convert S/36 screen formats to AS/400 DDS
     4. Convert S/36 menu files to AS/400 menus
     5. Create DDS from data file descriptions



Selection or command
===> 4
_
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support      F16=System main menu
```

Select option 4 (Convert S/36 menu files to AS/400 menus).

```
QCVMNU5              Convert S/36 Menu Files to AS/400 Menus

Select one of the following:

     1. Convert individual menu
     2. Create menu conversion list
     3. Work with menu conversion list
     4. Print menu conversion list
     5. Convert menus from conversion list

    10. Delete menu conversion list



Selection or command
===> 1

F3=Exit    F4=Prompt    F9=Retrieve    F12=Cancel
F13=User support        F16=System main menu
```

Select option 1 (Convert individual menu) to get the following screen.

This screen is also displayed if you use F4 after typing the CVTMNU command. You can start the conversion directly as a batch job by using:

```
CVTMNU KMPCBL
```

```
                          Convert Menu (CVTMNU)

Type choices, press Enter.

Menu . . . . . .  kmpcbl      Name
  Library . . . . .  *CURLIB    Name, *CURLIB
To Library . . . . . .  *CURLIB    Name, *CURLIB
Prefix . . . . . .  x         A through Z
Submit to batch . . . . . .  *YES       *NO, *YES



                                                                  Bottom
F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F13=How to use this display
F24=More keys
```

Type in the name of the menu you want to convert.

Also type a prefix if you wish. The prefix is then added to the called program name. Use a prefix if you know that your procedures and your RPG or COBOL programs have the same names. On the System/36 and in the System/36 Environment, procedures and programs may have the same name. But application conversion normally includes converting procedures to CL programs. The CL programs may not have the same names as the RPG or COBOL programs. PTK allows you to add a prefix to the name of the procedures that are called from the menu, and you can add the same prefix to the CL program name later when you convert OCL to CL.

When you press enter, PTK creates a conversion list as well as the other objects. The conversion list can be found in your output queue and displayed using the WRKOUTQ command as described on 44.

```
                              Display Spooled File
 File . . . . . :   KMPCBL                          Page/Line   1/1
 Control . . . . .  _____                           Columns
1 - 78
 Find . . . . . .   _____________________________
 *...+....1....+....2....+....3....+....4....+....5....+....6....+....7....
+...
  CONVERSION AID VERSION02                         MENU MESSAGE FILE DISPLAY
  MESSAGE FILE NAME . . . . . :   KMPCBL##         LIBRARY . . . . . :
  MSGID              COMMAND
  USR0001            CALL       PGM(*LIBL/XSTP965)
  USR0002            CALL       PGM(*LIBL/XTSP60)


                                                                      Bottom
 F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys
```

The conversion list shows you what PTK has done with your System/36 Environment menu commands. For example, PTK converted the commands in the original menu call procedures STP965 and TSP60 to "CALL" commands.

As you can see, the prefix was added to the name of the called program. This is an easy way to match the names of the CL programs that will be created from your procedures in the OCL to CL conversion step.

6.3 Creating Native Menus Without PTK

The easiest way to create a native AS/400 menu is to use AS/400 native SDA. With SDA you can create the necessary objects. The next screens show you the old System/36 Environment menu and the new native menu. The DDS for the new menus is also shown.

```
COMMAND                          MENU: KMPCBL              W6

Select one of the following:

 1. Procedure STP965                13.
 2. Procedure TSP60                 14.
 3. Call program XYZ                15.
 4.                                 16.
 5.                                 17.
 6.                                 18.
 7.                                 19.
 8.                                 20.
 9.                                 21.
10.                                 22.
11.                                 23.
12.                                 24.



Ready for option number or command
===> _
_
```

This menu is shown by the MENU KMPCBL command in the System/36 Environment. It is the menu we want to convert.

The first step in our menu conversion is collecting all the available information. We can use the Print key to print the screen layout, and use LISTLIBR or PDM to print the menu command source member (the source member with the suffix "##").

The next step is to rebuild the screen layout with the native SDA menu functions. Then you must examine your menu commands and change them to usable CL commands. For a detailed description of how to change OCL to CL refer to 15.6, "Operation Control Language Statements" on page 139.

The next screen shows the native version of the same menu. This menu was shown with the GO KMPNAT command.

```
KMPNAT                          KMPNAT Menu

Select one of the following:

     1. Call program STP965
     2. Call program TSP60
     3. Call program XYZ
     4.
     5.
     6.
     7.
     8.
     9.
    10.



Selection or command
===> _
_
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=User support      F16=System main menu
```

SDA creates two source file members for each menu. They are:

- A DDS source member type MNUDDS
- A DDS source member type MNUCMD.

You can use PDM to display the source members.

The next figure shows a listing of the MNUDDS source file.

```
FMT A* .....A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
       ********** Beginning of data *************************************
     A* Free Form Menu: KMPNAT
     A*  90/03/12  15:35:46    ITSCID13    REL-R02M00  5728-PW1
     A                                      DSPSIZ(24 80 *DS3
     A                                             27 132 *DS4)
     A                                      CHGINPDFT
     A                                      INDARA
     A                                      PRINT(*LIBL/QSYSPRT)
     A          R KMPNAT
     A                                      DSPMOD(*DS3)
     A                                      LOCK
     A                                      SLNO(01)
     A                                      CLRL(*ALL)
     A                                      ALWROL
     A                                      CF03
     A                                      HELP
     A                                      DSPMOD(*DS3)
     A                                      LOCK
     A                                      SLNO(01)
     A                                      CLRL(*ALL)
     A                                      ALWROL
     A                                      CF03
     A                                      HELP
     A                                      HOME
     A                                      HLPRTN
     A                                    1  2'KMPNAT'
     A                                       COLOR(BLU)
     A                                    1 33'KMPNAT Menu'
     A                                       DSPATR(HI)
     A                                    3  2'Select one of the following'
     A                                       COLOR(BLU)
     A                                    5  7'1.'
     A                                       HLPRTN
     A                                    1  2'KMPNAT'
     A                                       COLOR(BLU)
     A                                    1 33'KMPNAT Menu'
     A                                       DSPATR(HI)
     A                                    3  2'Select one of the following'
     A                                       COLOR(BLU)
     A                                    5  7'1.'
     A                                    6  7'2.'
     A                                    7  7'3.'
     A                                    8  7'4.'
     A                                    9  7'5.'
     A                                   10  7'6.'
     A                                   11  7'7.'
     A                                   12  7'8.'
     A                                   13  7'9.'
     A                                   14  6'10.'
```

Figure 2 (Part 1 of 2). DDS for Native Menu

```
A* CMDPROMPT  Do not delete this DDS spec.
A                                019  2'Selection or command'
A                                  5 10'Call'
A                                  5 15'program'
A                                  5 23'STP965'
A                                  6 10'Call'
A                                  6 15'program'
A                                  6 23'TSP60'
A                                  7 10'Call'
A                                  7 15'program'
A                                  7 23'XYZ'
  ****************** End of data ****************************************
```

Figure 2 (Part 2 of 2). DDS for Native Menu

To modify the screen layout you can change the DDS directly within the DDS source members or you can use SDA functions. We recommend using SDA, because it is very easy. A discussion of creating and updating an AS/400 menu without SDA is at the end of this chapter.

Next is a display of the MNUCMD source member.

```
 Columns . . . .:   1  71              Browse      CONLIB/QDDSSRC
 Find  . . . _____________________________________KMPNATQQ
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
        ********** Beginning of data ************************************
0000.01 KMPNATQQ,1
0000.10 0001 CALL PGM(CONLIB/STP965)
0000.11 0002 call pgm(conlib/tsp60)
0000.12 0003 call xyz
        ****************** End of data ****************************************



 F3=Exit           F5=Refresh          F10=Top           F11=Bottom
 F12=Cancel        F13=Change defaults                   F24=More keys
                                        (C) COPYRIGHT IBM CORP. 1981,
1989.
```

In most cases you will have to convert the commands that they call to CL programs. You must create the CL programs from your existing System/36 Environment procedures. Remember, you cannot have two programs with the same name in the same library. So if you have procedures with the same name as your programs, you should add a prefix to your called programs (CL) within your menu. For example, add a P = procedure:

```
System/36 Environment menu command : PROG01 (// LOAD PROG01).
Native menu command: CALL PROG01P (CALL PROG01).
```

If you want to create menus, but you do not have SDA, you must execute the following steps:

1. Describe the menu and menu help information using DDS.
2. Create the described DSPF using the CRTDSPF command.
3. Create a message file using the CRTMSGF command.

4. Add messages to the message file using the ADDMSGD command.
5. Create the menu using the CRTMNU command.

For a detailed description of this approach, refer to the *CL Programmer's Guide SC21-8077 Chapter 10.*

7.0 Converting System/36 Environment Printer Files

7.1 PRTF Considerations

To convert printer file definitions, you should know how to define a printer file on AS/400. A printer file on AS/400 must have a source type of PRTF and is described with the DDS.

Conversion to externally described printer files must be done manually. PTK has no option to help you with this procedure.

We recommend that you do this conversion, only if you have several programs using the same printer file. The reasons for printer file conversion are easy maintenance and the availability of IPDS functions within the DDS definition. There are other programming techniques you can use, but you must redesign your programs. For example, create a printer program that creates your printouts. This can be a batch run program using a DATAQUEUE that is started at the beginning of your work. When you send data into the DATAQUEUE, the program starts running. After finishing, the print program looks again in the DATAQUEUE. If there is data in the queue, the program starts again. If there is no data in the queue, the status of the job becomes DEQW (The job is waiting on a dequeue operation). In this status, the program is not using the CPU, but the program is still active. This means that all your printer files are open and the job has not been initiated the next time.

To create your externally described printer files, search your programs to find the necessary information about the printer file layout and the printer formats.

The work you must do is approximately the same as creating a display file when you have different internal display file descriptions.

8.0 Building the Field Reference File

A field reference file (FRF) can be considered a dictionary that contains all of the field descriptions of your database files. You may choose to have several FRFs, one for each independent application. We recommend that you have only one FRF, however, because the independent applications might grow together in the future and use the same fields. The FRF contains only one record format, and is created like any other physical file on the system. You can save disk space if you use the MBR(*NONE) parameter when you create the file. You do not need a member within this file because you never put data into a field reference file. All fields, together with their attributes, reside in one easily maintained file and any other file will just refer to the FRF when being created. Each field in an externally described file is now known to all programs by the same name.

As you can see in the examples on the next page, the existence of an FRF reduces the difficulty of coding DDS and the incidence of coding errors dramatically. We could have created the FRF during the process described in 9.0, "Modifying DDS and Creating Database Files" on page 63. However, this might require additional DDS changes due to changes in the HLL programs.

8.1.1 Creation Steps

There are several ways to create an FRF. The one you select depends on how many files you have and how many unique records and fields are contained in them. Here is one way to create the FRF:

1. Create a source physical file (called QDDSSRC, for example) to contain the FRF. Library QGPL already contains a file by that name that you could use as well.
2. Create a member with a name of your choice.
3. Add a dummy record name on top of the source.
4. Copy all the fields of your DDS to the FRF source. Consider Figure 3 on page 60 and Figure 4 on page 60. At this time you could add some descriptive text and any additional keywords you might need (if not already done during 9.1, "Adding Documentation" on page 63).
5. Carefully read through the FRF to see whether you have several field names for the same piece of information. You do not necessarily have to change them because in some cases (especially for RPG) you may have different field names for the same field. A field called DATE may be written SDATE when used in a display file or when read from another database file.

   The REFFLD keyword can be used when the same attributes apply to different field names. Using the previous example, you could name the field DATE in the FRF and using the REFFLD keyword on the display file, name the field SDATE.
6. Now compile the FRF as a physical file. See Figure 5 on page 60 for a (shortened) sample of the compilation list. From now on, any new database or display file you create should use the FRF to obtain its field definitions.
7. Change your original DDS (after backup) to refer to the FRF. See Figure 6 on page 61.

- Insert a line with REF(FRF) keyword. This specifies the file that is referred to.
- In the field lines, remove everything except the name itself.
- Insert the character R in column 29 to indicate that this field is a reference field (defined in file FRF).

This helps ensure that the field attributes are used consistently in all files and reduces the effort in coding DDS statements. For further information about FRFs refer to the *Database Guide* and the *Data Description Specifications Reference*.

8.1.2 Programming Examples

These are only a few coding examples; they are not related to the previous examples showing the use of PTK. These examples are included to show you how easy it is to refer to an FRF.

```
A          R CR
A            CUSTNO         8          TEXT('CUSTOMER NO')
A            NAME          17
A            CODE           4  0
A            AMOUNT         5  2
```

Figure 3. DDS Source Not Using Field Reference File

```
A          R DUMMY
A            CUSTNO         8          TEXT('CUSTOMER NO')
A            NAME          17
A            CODE           4  0
A            AMOUNT         5  2
A            ...
A            ...
```

Figure 4. DDS Source Field Reference File

```
           R DUMMY
             CUSTNO         8A  B      TEXT('CUSTOMER NO')
                                       COLHDG('CUST#')
             NAME          17A  B      COLHDG('NAME')
             CODE           4P 0B      COLHDG('CODE')
             AMOUNT         5P 2B      COLHDG('AMOUNT')
             ...
             ...
             ...
```

Figure 5. Compiled Field Reference File

```
A                                       REF(TETLIB/FRF)
A           R MASTREC
A             CUSTNO     R
A             NAME       R
A             CODE       R
A             AMOUNT     R
```

Figure 6. DDS Source After Changing to Field Reference File

9.0 Modifying DDS and Creating Database Files

This chapter helps you change the DDS created by the PTK to cater to certain conditions that the PTK does not handle. The following list shows the items considered here:

- Adding documentation
- Shortening record lengths
- Changing record names
- Adding keys
- Checking data type
- Alternate index files
- Creating files.

9.1 Adding Documentation

The PTK does not add any documentation to the generated DDS. Take this opportunity to describe the purpose of the file and to specify any relevant information using the DDS keyword TEXT. Use the DDS Keywords TEXT, COLHDG, and EDTCDE to describe the field.

9.2 Shortening Record Lengths

In a multiple record file on System/36, the record length of the file is set to that of the longest record format. On the disk, the longest format uses all of the record, but the shorter records have some wasted or unused space, usually at the end of the record.

When you used the PTK to generate the DDS for one of the shorter records, you had to add a filler or unused field to take up the unused positions. The PTK then produced the DDS for the record, setting the record length to that of the longest format in your multiple record file.

On the AS/400, each record format has its own physical file. You can save space on the disk by reducing the record lengths of the shorter files. Simply edit the DDS and remove the line (or lines) describing the unused, filler fields.

Do not do this if you want to run your programs using internally described files.

9.3 Changing Record Names

The PTK automatically assigns a record name in the generated DDS. For a single-format file, the record name will be the name of the file with SF added. For a multiple-record file the name will be the file name with the two additional characters (for example AA, BB, or CC) that are specified on the "Analyze File Description" screen like the one below.

```
FMTRSLV            Format Resolution

External File        TWO
Enter format designators.

      Ps1  Cd1  Ps2  Cd2  Ps3  Cd3   Format    Status

      0001  CD  0000      0000         AA    Generated
      0001  CH  0000      0000         BB
      0001  CT  0000      0000         CC
```

Figure 7. Modifying DDS - PTK Analyze File Description

You may wish to change these record names to conform to your own naming conventions.

9.4 Adding Keys

This step is needed for multiple record format files where the PTK generates the DDS with key fields of *NONE, instead of using data fields as key fields.

Let us take an example where we have three record formats representing the order header, order detail, and total records in an order's file. This file might be read sequentially and matched against a sorted transaction file for update.

```
IORDERS  AA  01   1 CH
I                                      1   1 ID
I                                      2   6 CUST
I                                      7  10 ORDER
I                                     11  16 DATE
IORDERS  BB  02   1 CD
I                                      1   1 ID
I                                      2   6 ITEM
I                                      7   90QTY
I                                     11  16 DATE
IORDERS  CC  03   1 CT
I                                      1   1 ID
I                                      2   6 TOTAL
```

Figure 8. Modifying DDS - A Multiple Record File

Note: There is no common field in the record types.

The PTK splits this file into three physical files. The DDS for the detail record, for example, will contain only the fields ID, ITEM, QTY, and DATE. The DDS for the detail record does *not* contain the order field. Therefore, when we come to recombine the three physical files (through the logical file DDS), there is not enough information present to let the AS/400 know which details belong to which order.

This missing information was previously held in the System/36 file because the detail records immediately followed the header record to which they belonged (by means of the sequence).

On AS/400 you must add an ORDER field to the detail record layout in the physical file and an ORDER field to the total record layout for the same reason. A sample DDS for the original detail record is shown below, followed by the changed DDS.

```
            A*  DETAIL RECORDS
            A          R BB
            A            ID             1
            A            ITEM           4
            A            QTY            3S 0

                                manually added ORDER field:

            A*  DETAIL RECORDS
            A          R BB
            A            ID             1
----->      A            ORDER          5
            A            ITEM           4
            A            QTY            3S 0
```

Figure 9. Modifying DDS - Adding a Key to a Physical File

In addition, you must adjust the DDS for the logical file to add the required key fields:

DDS generated by the PTK:

```
          A*  ORDER HEADER
          A           R AA                        PFILE(TWOAA)
          A           K *NONE
          A*  LINE ITEMS
          A           R BB                        PFILE(TWOBB)
          A           K *NONE
          A           K ITEM
          A*  ORDER TRAILER
          A           R CC                        PFILE(TWOCC)
          A           K *NONE
```

manually changed key field

```
          A*  ORDER HEADER
          A           R AA                        PFILE(TWOAA)
----->    A           K ORDER
          A*  LINE ITEMS
          A           R BB                        PFILE(TWOBB)
----->    A           K ORDER
          A           K ITEM
          A*  ORDER TRAILER
          A           R CC                        PFILE(TWOCC)
          A           K ORDER
```

Figure 10. Modifying DDS - Adding a Key to a Logical File

9.5 Checking Data Type

The DDS generated for numeric fields may include a data type of "S":

```
          A           QTY           3S 0
```

Unless you specifically want such a field treated as a zoned decimal field, replace the "S" with a "P" to treat the field as a packed decimal field. You will also need to change the data in the file. This is recommended for performance reasons, but *only* if you plan to convert *all* programs using this file to external definitions. Otherwise, the buffer positions in the programs will be off.

9.6 Alternate Index Files

Alternate indexes become logical files on AS/400 during migration. On System/36, the index might have been made up of several fields or positions in the record. For instance, one index may have used positions 3 to 15 as the key field, another may have used positions 10 to 20. On the System/36, there was no need for the key fields to correspond to field names.

On AS/400, the PTK tries to use field names when generating the DDS for an alternate index. Each alternate index generates a logical file (see Figure 11 on page 67).

The PTK generates DDS for each alternate index at the time when the file on which the index is based is resolved.

Because the positions specified for the alternate index key may not correspond with the positions of the fields in the based-on file, the PTK can, if necessary, substring the fields in the based-on file (see Figure 11).

```
      File contains  FIELD A    in positions  1   5
                     FIELD B                 16  20

If the alternate index is over positions 2 to 5 of the file,
the generated DDS will look like:

                     R RECORD
                       A
                       #KEY1       I      SST(A 2 4)
                       B
                     K #KEY1
```

Figure 11. Modifying DDS - An Alternate Index

The alternate key may also overlap two or more fields.

If the DDS for the alternate indexes contains substrings, you may wish to alter the DDS of the based-on physical file to give each key field an individual name, thus removing the need for substrings. The presence of substrings indicates an unusual and perhaps incorrect database definition of the key fields.

9.7 Creating Files

Now we are ready to create physical and logical files using the proper AS/400 commands. If your application uses external (within the procedure) file names with substitutions (perhaps USER or WS-ID dependent), choose a "common" name to create a file containing just the format. Your file specification in your programs should be changed to that name before compilation (the file must exist during compilation). Before running the program, use the OVRDBF command to link to the "real" file.

9.8 Format Selection

This section is intended to provide an understanding of the purpose of format selection and show how format selection can be done.

The topic is introduced at this stage because one of the methods of doing format selection (use of a format selector program) could be useful in two subsequent

steps: first, when copying data into the AS/400 database files, and second, at run time when your programs are using the files.

9.8.1 Why Format Selection?

If your System/36 program processed a file with multiple record formats, we have discussed how the PTK converted that file to multiple physical files, which were then joined together by a logical file containing multiple records. Your program then reads the logical file.

Note that if your System/36 program added records to the multiple record file, it did so by referring to the name of the file in the output specification. For example:

```
OORDERS  DADD    01
```

However, if an AS/400 HLL program tries to add to a multiple record logical file by referring to the name of the file on the output instruction, it will fail.

The reason is that when asked to add a record to a file named ORDERS, the AS/400 HLL program does not know which underlying physical file to put the record into. Extra information must be supplied to the AS/400 data management to help it select the correct format name.

Format selection information needs to be provided only under certain conditions. If these conditions do not apply, then you do not have to worry about format selection.

Format selection is only needed if:

1. You are using the file name instead of the record name.
2. The file is a logical file with more than one record format.
3. You are adding records to a file.

Note that if the program is doing updates only (no adds), you do not need format selection because the program has already read the record and knows which physical file it came from.

The name of the record format (rather than the file name) can be supplied to AS/400 data management in two ways:

1. Selection by Record Names

One way is to replace file names by record names in the HLL source. Thus:

```
OORDERS  DADD    01
```

could become

```
ODETAIL  DADD    01
```

or

```
OHEADER  DADD    01
```

depending on the intent of the program.

2. Selection by Format Selector Program

The format selector program allows you to continue using the file name when adding a record. The format selector program can be written in CL or in RPG.

A COBOL format selector program is also provided. It normally requires more modification than the RPG format selector. In general, however, the format being written is obvious in a COBOL program. The record name is used in the WRITE statement. For COBOL, we suggest that you modify the program to include the appropriate format name on the WRITE statement.

The format selector program is attached to the logical file. When your program writes a record to the logical file, the format selector program examines the record (after it goes outside your HLL program) and, based on the content of the record, supplies the appropriate record name to the logical file.

9.8.2 How is the Format Selector Written?

You do not have to write a format selector. The PTK generates one written in RPG for each multiple record logical file. The selector is written when the DDS for all the physical files supporting the logical file have been created.

Figure 12 shows the multiple record description, and Figure 13 on page 70 shows the format selector program that is attached to the logical file. The logic of the program uses the RPG record identifying indicators to generate the appropriate record format name. It is up to you to review the program and make any needed corrections.

```
FORDERS  IP  F      17           DISK
I* HEADER RECORD
IORDERS  AA  01   1 CH
I                                       1   1 ID
I                                       2   6 CUST
I                                       7  11 ORDER#
I                                      12  17 DATE
I* DETAIL RECORD
I        BB  02   1 CD
I                                       1   1 ID
I                                       2   6 ORDER#
I                                       7  10 ITEM
I                                      11  130QTY
I* TOTAL RECORD
I        CC  02   1 CT
I                                       1   1 ID
I                                       2   6 ORDER#
I                                       7  112TOTAL
```

Figure 12. Format Selection - A Multiple Record Format

The format selector is tied to the logical file during the CRTLF command.

```
     ****************************************************************
     * This Format Selector is ONLY a skeleton example of the program
     * that is required. Please review the format selection criteria
     * and make the necessary adjustments.
     * NOTE : Default format selection code must be added below.
     ****************************************************************
    E                    @        0032  1
     ****************************************************************
    C           *ENTRY    PLIST
    C                     PARM           @
    C                     PARM           FORMAT 10
     ****************************************************************
     * Format Selection Criteria retrieved from ANZS36FD Format Resolve
     ****************************************************************
    C*
    C* FORMAT - AA
    C*
    C           @,0001    IFEQ 'H'
    C                     MOVEL'HEADER'  FORMAT
    C                     GOTO EOP
    C                     END
    C*
    C* FORMAT - BB
    C*
    C           @,0001    IFEQ 'D'
    C                     MOVEL'DETAIL'  FORMAT
    C                     GOTO EOP
    C                     END
    C*
    C* FORMAT - CC
    C*
    C           @,0001    IFEQ 'T'
    C                     MOVEL'TOTAL'   FORMAT
    C                     GOTO EOP
    C                     END
    C*
    C           EOP       TAG
     *
     * ==> Add default format processing (ex. MOVEL '??' FORMAT) <==
     *
    C                     MOVE '1'       *INLR
```

Figure 13. Format Selector Program

9.8.3 Recommendations

If you need to provide format selection in any of your programs, we suggest that you use the first method (replacement of file names in the program by the underlying record format names) rather than the second method (format selector). It is more visible to the maintaining programmer.

Although the use of record names may involve (in some cases) changes in the program logic to set up the correct output indicators on the O-specifications, we believe that there are several benefits to be gained by using record names:

- Ability to add extra formats to the logical file without the need to recompile the format selector
- No need to understand, create, and maintain format selector programs
- Possible performance advantages of operating directly on the underlying physical files

It should be stressed that the generated format selector is a "best guess" at the selection criteria to determine a format. The source must be reviewed and a default format added where the source indicates it should be.

9.9 Copying Data into the Files

At this stage, the DDS has already been compiled and we have one or more empty AS/400 files. The data must be copied from the existing System/36 files to the AS/400 files before our programs can be run.

There are two main cases, which are discussed in this section:

- System/36 file with a single record format
- System/36 file with multiple formats.

9.9.1 System/36 File with a Single Record Format

In this case the PTK has generated a single physical file. Copy the data into the physical file using the CPYF command with the format:

```
CPYF FROMFILE(QS36F/S36FILE) TOFILE(NEWLIB/AS400FILE) MBROPT(*REPLACE)
     FMTOPT(*NOCHK)
```

Of course, if the record format of the physical file differs from that of the System/36 file you must write a program to copy across the correct fields or create a physical file that describes the layout of your System/36 Environment file. For example:

The old file has the following field order:

```
FIELD01
FIELD02
FIELD03
FIELD04
```

The new file has the following field order:

```
FIELD01
FIELD04
FIELD03
```

Now you can solve the problem in one of two ways. The first way is to write a program that reads the System/36 Environment file and moves the fields into the correct sequence. Then add the record to the native file. Use this technique if the field names of the files are different.

The other way is to create a physical file that matches the System/36 Environment file. To do this, copy the DDS of your new file and build a physical file with the old field sequence. In most cases you only have to move the field description into the correct sequence.

Now copy your System/36 Environment file into the created file. First:

```
CPYF FROMFILE(file-name) TOFILE(file-name) MBROPT(*REPLACE)
     FMTOPT(*NOCHK)
```

FROMFILE This is your System/36 Environment file.

TOFILE This is your native file with the correct field layout of your System/36 Environment file.

Because both files use the same field names, but the fields are not in the correct positions, you have to use the following CPYF command. Next:

```
CPYF FROMFILE(file-name) MBROPT(*ADD) TOFILE(file-name)
     FMTOPT(*MAP *DROP)
```

FROMFILE This is your native file with the correct field layout of your System/36 Environment file.

TOFILE This is your native file with the new field layout.

The *MAP keyword allows you to copy fields with the same name from one file to another even if the fields have different starting positions.

The *DROP keyword allows you to drop fields when they are not defined in the receiving file.

9.9.2 System/36 File with Multiple Formats

This case is a little more detailed. The PTK has built a logical file over multiple physical files.

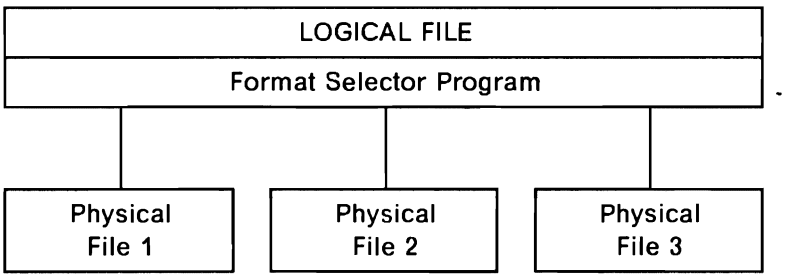


Figure 14. Format Selection - Diagram of Logical File and Three Physical Files

Again, there are several techniques for copying multiple format files:

1. Copy using the format selector created by PTK

This is a quick method because the format selector is already written for you, and you can use a general-purpose RPG II or RPG/400 program to do the copying.

If you plan to use this method, be sure that the logical file specifies its format selector. The selector is given the same name as the logical file with two # characters added. Thus, a logical file PKFILE would have the selector PKFILE##. You will need to recreate the logical file (with the FMTSLR keyword) if you have changed the logical file DDS in any way.

You will then need a program to read the System/36 file and write to the logical file with its associated selector. A general-purpose program could be like the one shown in the following figure:

```
FFILEIN  IF  F     256           DISK
FFILEOUT O   F     256           DISK
IFILEIN  AA
I                                      1 256 DATA
C                   READ FILEIN                   99
C           *IN99   DOWEQ'0'
C                   EXCPT@WRITE
C                   READ FILEIN                   99
C                   END
C                   MOVE '1'       *INLR
OFILEOUT E              @WRITE
O                       DATA     256
```

Figure 15. Format Selection - a General Build Program

The record lengths for input and output must be adjusted to the length of the System/36 input file. The field lengths must be adjusted accordingly. The program must then be compiled with a suitable name (such as BUILD).

When the program is run you will need override statements to attach the correct files to the program:

```
OVRDBF FROMFILE(FILEIN) TOFILE(QS36F/MASTER)
OVRDBF FROMFILE(FILEOUT) TOFILE(PRODLIB/MASTER)
CALL BUILD
```

You could add to the logic of the build program to handle the case where additional fields need to be written to some records. The build program also could rearrange fields in records. This is because the format selector does not care what the record layouts of the input and output records are, provided the characters used for record identification are correct.

The build program could thus be changed to add an order number field to the item detail records, as required in the earlier example.

2. Copy individual physical files

Another method of copying a multiple format file is to create a program to read the multiple format files and write records to one or more of the physical files. This method gives more freedom than the format selector, because there is no need to preserve the record identification codes.

3. Copy individual physical files with the CPYF command

Normally on System/36 you have one field within the file that indicates your record format. You can use this field to identify the record within the CPYF command. But before you can use the command, you have to build a physical file that looks like your System/36 Environment file, but has the indicating record defined. Define all fields as character fields:

```
F0001 = (record format identification field)
F0002 = (filler field to match the record length)
```

Now create the physical file. Copy the data from your System/36 Environment file into the newly created physical file using the CPYF command:

```
CPYF FROMFILE(file-name) TOFILE(file-name) MBROPT(*ADD) FMTOPT(*NOCHK)
```

FROMFILE This is your System/36 Environment file.

TOFILE This is your native file with the correct field layout for the indicating field of your System/36 Environment file.

Now you have a native file that contains the same records in the same sequence as the System/36 Environment file. The next step is to copy the correct record into your newly created files.

Because the format indication field is know known by the system, you can copy the records to the corresponding files. You have to run the following CPYF command for each format you have in your file:

```
CPYF FROMFILE(file-name) TOFILE(file-name) MBROPT(*ADD)
INCCHAR(F0001 1 *EQ '2') FMTOPT(*NOCHK)
```

FROMFILE Native file with the correct field layout for the indicating field of your System/36 Environment file

TOFILE Native file, which contains the selected record layout

F0001 Field name of the indication field

1 Start position of the indicating field

*EQ Indicates the relationship; in our case it's equal.

'1' Specify the character string compared with the specified indicating field.

10.0 Decimal Data Errors

A decimal data error occurs when an AS/400 program processes an instruction that requires at least one of the fields involved to contain a number, but where all of the fields contain values that are not numbers.

This is not a problem on the System/36, because programs in RPG II and ANS COBOL74 can process non-numeric fields as numeric. Because the System/36 operated on zoned fields, it was able to treat a field like X'414243' as numeric "123" when doing arithmetic and still get a correct answer.

AS/400, however, operates internally on numeric fields with packed data. It requires numeric fields to contain valid numeric values. For example, to be valid, a zoned decimal field must have the high-order four bits as X'F' except in the rightmost byte, where a valid sign is required. The number "123" is represented as X'F1F2F3'. In fact, on AS/400 all numeric fields (if stored in zoned form) are converted to packed format before arithmetic calculations are done on them. A packed decimal numeric field must also have valid numeric digits and signs.

Thus, a field containing data that could be processed acceptably on System/36 or the System/36 Environment might not be acceptable in native AS/400.

This problem could be quite severe, because an AS/400 program will cancel if it processes an instruction that finds non-numeric data where it expects numeric. We thus need to be able to detect and correct the causes of these errors.

10.1 Some Rules for Non-Decimal Data

Programs compiled with the System/36-compatible RPG II or ANS COBOL74 compilers will run the same way as they did on the System/36. In other words, they will accept non-numeric fields and process them without decimal data errors.

Decimal data errors can occur in RPG/400 programs in two ways, either when the program is doing arithmetic, or at logical I/O time. When the program is moving data from a file input buffer to a field, if the field is specified as numeric but the data is not numeric, an error will occur. The result is similar for output fields.

Both RPG II or RPG/400 programs with program-described fields can write invalid decimal data. The program involved can define the same field twice--once as numeric and once as non-numeric. This can be done either on input specifications or with a data structure.

Thus, System/36 Environment RPG II programs and RPG/400 programs can place invalid data in database files.

COBOL programs can place invalid data in database files as well. Unlike RPG, which is a field-oriented language, COBOL is a record-oriented language. This makes it possible to put alphabetic data in a decimal field and then write the field out to a database file. This could happen if you have mapped your record

layout in the working-storage section, and a field in the working-storage section has a different attribute (decimal/alphabetic) in the file section. The wrong data may then be written out using a WRITE FROM.

In COBOL you will not get a "Decimal Data Error" message when reading or writing alphabetic data to or from decimal fields in a file. That message occurs only when performing arithmetic operations on that data. On the other hand, the compiler itself catches some of those operations by not allowing arithmetic operations on alphabetic fields.

To make sure that you operate on properly defined fields with proper contents in the file we recommend use of the PTK.

10.2 Finding and Correcting Errors in Files

The PTK can help find fields containing data that should (according to the external file description) be numeric, but which is not. The PTK is simple to run and avoids the need to write user programs to do checking.

10.2.1 Single Format File

The PTK will check a file with a single record format and tell you if there are decimal data errors.

There are two inputs required by the PTK:

- A single format file. This could be a System/36 (program-described) file in QS36F, or an externally described AS/400 file.
- An AS/400 file that contains the record format description for the single format file above. Of course, if the single format file was an AS/400 externally described file, it would carry its own format description.

There are three outputs produced by PTK:

- A report indicating the records with decimal data errors
- A copy of the analyzed file with data corrected (optional)
- A completion message telling whether decimal errors were detected.

The PTK takes the record format from the AS/400 file and reads the data records. It checks to see that each field in the record contains only the type of data specified in the record format.

If a field specified as numeric contains non-numeric data, the PTK will change it in one of two ways (you choose which one): either set the whole field to zero (*ZERO), or set the zone portion of each invalid byte in the field to X'F' (*S36). For example:

```
If your field contains:        A B J 8 9
                              X'C1C2D1F8F9'

    PTK corrects it to:

      using option *S36         1 2 1 8 9
                              X'F1F2F1F8F9'

      using option *ZERO        0 0 0 0 0'
                              X'F0F0F0F0F0'
```

The PTK will also correct errors in packed numeric fields.

10.2.2 Multiple Format File

The PTK currently cannot analyze a file with multiple record formats, like the ORDERS file with HEADER DETAIL and TOTAL records. This file must be converted into a logical file and three physical files. If you have already copied the data into these physical files, just run the PTK over each (single record format) physical file as described in the previous section.

10.2.3 Compiler Options for Decimal Data Errors

The CRTS36RPG command has an option, FIXDECDTA, to fix decimal data. The CRTS36CBL command has the same option. The default is *YES. This option "corrects" any invalid data by forcing the zone portion (high-order four bits) of each byte to X'F' as described in 10.2.1, "Single Format File" on page 76. Also, the low-order four bits are forced to zero if they are not in the range, zero through nine. Results may be unpredictable, the same as for the System/36.

We recommend that the *NO option be used. FIXDECDTA(*NO) will also allow your programs to run faster, since no correction of the data will be attempted. Of course, if FIXDECDTA(*NO) is used and there is invalid numeric data, then your program will stop when a decimal data error occurs.

The CRTRPGPGM command has an option, IGNDECERR, to ignore decimal data error. (There is no option on the CRTCBLPGM command to do the same.) The default is *NO. If you change this to *YES, you will not get decimal data errors when running the program. But your program may give unpredictable results, because any instruction that detects the error may only partly complete before processing the next instruction.

11.0 RPG Considerations

11.1 RPG and Database Files

The next sections present a step-by-step guide for the process of changing RPG programs to use externally described database files. This process is required whether or not you have used PTK for the first stage of the RPG II-to-RPG III conversion, because you will still have to eliminate the program-described input and, optionally, the output for the new externally defined files.

The ideas given should cover most of the situations you will find in practice, and might help solve problems not specifically mentioned.

At this stage we have generated DDS for the AS/400 external database files. But the programs still contain program-described files, which might differ in record layout from one program to another.

Several changes must be made to the source code of each program to change a file from a program-described to an externally described file. These are discussed below with the aid of some sample code. The following sections describe the changes to be made.

11.1.1 Auto Report Changes

Read this section if you are using auto report for data files (DISK on the file specifications). Also read Chapter 8 of the *RPG/400 Reference* manual.

The intent of this step is to remove the /COPY modules that contain file definitions. They are no longer needed on AS/400, since the description of an external file is held in the file itself.

There might still be some /COPY modules that you want to keep for frequently used calculation subroutines or for files that will not be converted because they contain arrays. These /COPY modules can still be used on AS/400.

The suggested method of handling auto report uses a preparatory step which, in effect, converts the auto report program to an ordinary RPG program. You then follow the steps for converting database files in an RPG program. This method will work in all cases and gives complete diagnostic messages.

Handling COPY Statements Before Conversion

All the auto report programs and /COPY modules are stored as members in QS36SRC by the System/36-to-AS/400 migration aid.

- Scan each auto report source program and change every /COPY statement to AS/400 format.

  For example,

  ```
  C/COPY library,module
  ```

  becomes

```
C/COPY AS400LIB/QS36SRC,module.
```

- Change each /COPY statement that you want to keep (for example, the calculations subroutines) to a comment statement. Also, add some characters, such as "KEEP", so you know it is a COPY statement that you want to keep.

  For example,

  ```
  C/COPY AS400LIB/QS36SRC,module
  ```

  becomes

  ```
  C*COPY AS400LIB/QS36SRC,module       KEEP
  ```

- Create a new source program that includes all the /COPY modules that have not been commented out (that is, the modules for which the source is to be placed into the program).

  To create the new program, run two commands of the form shown below (the first creates an empty member to receive the output):

  ```
  ADDPFM FILE(AS400LIB/QRPGSRC) MBR(xxxxxx)

  CRTRPTPGM PGM(AS400LIB/xxxxxx) SRCFILE(AS400LIB/QS36SRC)
  RPTOPT(*NOCOMPILE) OUTFILE(AS400LIB/QRPGSRC) OUTMBR(xxxxxx)
  ```

  where xxxxxx is the name of the source program.

This command merges the RPG lines and /COPY modules from QS36SRC and places the output in QRPGSRC, in a member with the same name as the original program. The command can be run repeatedly using PDM, or by writing a simple CL program that accepts the member name of the member to be processed.

- If there are any /COPY modules that still need to be kept, copy them over to the QRPGSRC file.
- Re-scan the new source program (now in QRPGSRC) and change the commented *COPY lines that are still needed back to /COPY.

  For example,

  ```
  C*COPY AS400LIB/QS36SRC,module       KEEP
  ```

  becomes

  ```
  C/COPY AS400LIB/QRPGSRC,module
  ```

- You might also remove the commented *COPY statements generated by the compilation, as they are no longer required.
- The pre-compile step that merged the program and /COPY modules removes the U specification. The U specification is only needed if you want to specify date or asterisk suppression. All other options may be supplied at compile time. Insert the U specification if required.

- If you still have /COPY statements in the program, on exit to the PDM screen you will need to change the member type to RPT to continue to use auto report.

You now have a source program which, as far as database files are concerned, is just like a program that never used /COPY modules.

11.1.2 Resolving the Use of Names

RPG II on the System/36 allows multiple uses of any name within a program. For instance, the same name could be used for a file, a field, and a subroutine, as long as the purpose was different. RPG III **does not allow such usage**.

You will need to resolve name usage within your application system and have some disciplined approach and naming scheme before you start the conversion.

Neither the Migration Aid nor the Programmer Tools PRPQ analyzes this name conflict. It does not show up until you compile the program with the RPG/400 compiler.

11.1.3 RPG Changes

The following program and DDS are the starting point for the examples in the following conversion steps. Read them before proceeding.

Here is a sample program with a program-described file. To illustrate a number of points, an indexed disk file is used requiring random key access. We will assume that the key is the customer number, which is made of two parts--a store number and a serial number.

```
FMASTER  UP  F     256  8AI     1 DISK
.
.
IMASTER  AA  01
I                                      1   2 STORE
I                                      3   80SERIAL
I                                      1   8 CUST#
I                                     26  290CODE
I                                     30  342AMOUNT
.
.
C                    MOVELISTORE    KEY     8
C                    MOVE ISERIL    KEY
C             KEY    CHAINMASTER                    99
C  N99              MOVE CUST#     NEWCUS  8
C   99              MOVE KEY       NEWCUS
.
.
OMASTER  D        01
O                        NEWCUS     8
O                        CODE      29
O                        AMOUNT    34
```

Figure 16. RPG and Database - A Program-Described File

Next is sample DDS used for the 'MASTER' file. It was generated by PTK, probably by making field name choices from conflicting and overlapping file and field definitions from many programs. In this example, there is a composite key with two database fields, which is different from the way that the file key is referenced in the RPG II F-specification.

You will also note that there is no overlapping redefinition in the DDS external record definition, and that the CODE and AMOUNT fields are *packed decimal*, not zoned decimal, as in the original definition above.

```
A          R MASTREC
A            LOCATN         2
A            NUMBER         6S 0
A            NAME          17
A            CODE           4P 0
A            AMOUNT         5P 2
A          K LOCATN
A          K NUMBER
```

Figure 17. RPG and Database - DDS for File MASTER

In this example there will have to be a number of program changes to convert to AS/400 native with an externally described file. The PTK conversion will do the minimum changes required for *program described files only*. You must make any further changes and reconciliation of input and output fields for externally described files. These changes are described in the next sections.

Minimum Changes for Program-Described Files

You should make some changes to the file specifications for program-described as well as externally described files:

- Be sure the file name on the F-specification is still the same name as the externally defined file you created. This will ensure that you will not need an OVRDBF statement (similar to // FILE in System/36) at run time, because the internal and external names will be the same. (You can still have OVRDBF to point to a different file or member.)
- Replace C or D in column 16 (File Designation) with F to indicate that the file is *fully procedural.* You still use CHAIN, READ(E/P), and SETxx operations in the program, and the program logic remains the same.
- Blank out Block Length.

Now you can compile the program. Because the file is still program-described (F in column 19), the external definition as defined in DDS is overridden by the program's definition in the F- and I-specifications.

The data fields in each record must, of course, match in length and type with the fields as defined in the F- and I-specifications. If they do not, the program might produce decimal data errors when attempting to process alphabetic data as numeric, or might produce incorrect results when working with fields whose internal and external lengths are different.

The external file as defined by the DDS can be either a physical or logical file. Either is acceptable, as long as the record layout and sequence is as expected by the RPG program.

In other words, if the DDS for the file does not exactly match the old record layout in the F- and I-specifications, the program will compile without error, but might not run correctly. In this case, you need to change the program source to match the altered record layout. The program can still be changed and recompiled with a program-described file. For the example above, you must change the I- and O-specifications for the CODE and AMOUNT fields to reflect the new length and type (packed) of those fields. Or the file can be made fully external, as described in the next section.

11.1.4 Changes for Externally Described Files

You need to implement the changes described in this section if you intend to use the externally defined field and record format names in the RPG program. Changes made to the DDS can then be incorporated directly into the program by recompiling it. Some of these changes are shown in Figure 19 on page 87.

11.1.5 Changing File Specifications

- Be sure the file name on the F-specification is still the same as for the externally defined file you created. Your program will not compile if the file cannot be found. This will also ensure that you will not need an OVRDBF statement (similar to // FILE in System/36) at run time, because the internal and external names are the same. (You can still have OVRDBF to point to a different file or member.)
- Replace C or D in column 16 (File Designation) with F to indicate that the file is fully procedural. You still use CHAIN, READ(E/P), and SETxx operations in the program, and the program logic remains the same.
- Replace F in column 19 (File Format) with E to indicate that the file is an externally described file.
- Blank out the following fields: Block Length, Record Length, , and Length of Key Field, as these must not be coded for externally described files.
- Code K in column 31 (Record Address Type) if a key is used to access the file. See the *RPG/400 Reference* manual for further information about record address type.
- Blank out the following fields: File Organization and Key Field Start Location, , as these must not be coded for externally described files.

The F-specification in the example program is changed:

```
FMASTER  UP  F     256  8AI     1 DISK
```

Becomes

```
FMASTER  UF  E           K        DISK
```

If you compiled your program now, you would get compiler messages that tell you what further changes must be made to process the externally described file. To save time, these changes are discussed below. Make them before compiling.

Changing Input Specifications

With an external file, input specifications must refer to the record name, not the file name. Change the file name coded on the I-specifications to the record name. In the following example, the file name is changed to the record name MASTREC. Also, blank out all the I-specification sequence number fields and the start, end, and decimal positions of the input fields from positions 15-16 and 44-52 respectively. Keep the I-specifications to help in identifying fields that must be renamed. You will delete these statements later.

```
IMASTER  AA  01
I                                      1   2 STORE
I                                      3   80SERIAL
I                                      1   8 CUST#
I                                     26  290CODE
I                                     30  342AMOUNT
```

Becomes

```
IMASTREC     01
I                                          STORE
I                                          SERIAL
I                                          CODE
I                                          AMOUNT
```

Additional fields that are not described in the database: As mentioned above, different programs might have contained different record layouts for the same record type in a file. The PTK showed us these layouts and, based on our knowledge of the system, we chose the correct layout to be turned into the external description.

There might be some programs where the definition of the record needed by the program does not match the external definition, or there may be specific redefinition of the input fields for program use.

In this case you can redefine the record layout of the external file by using a data structure to subdivide an input field so that the program can refer to the entire field or just the individual subfields. They can be used to redefine an area and group fields together. See the *RPG/400 User's Guide* for details. The data structure in the sample code is intended to show the way in which fields can be subdivided and gathered.

In the sample program we make the following change.

```
I                                      1   2 STORE
I                                      3   80SERIAL
I                                      1   8 CUST#    ----
I                                     26  290CODE         |
                                                          |
Becomes                                                   |
                                                          |
I                                      1   2 STORE        |
I                                      3   80SERIAL       |
I                                     26  290CODE         |
.                                                         |
.                                                         |
I         DS                                              |
I                                      1   8 CUST#    <---
I                                      1   2 LOCATN
I                                      3   80NUMBER
```

Alternatively, a logical file can be used in some cases to concatenate fields that have similar attributes.

Changing Calculation Specifications

You can still use the file name in C-specifications for any INPUT operations. There are no operation codes in RPG II for direct output to disk (EXCPT uses the O-specifications.). Change the C-specifications that perform disk operations so they use the new external file name. However, be aware that on AS/400 you have the option of reading from a record format, using RPG/400 operation codes in the calculations. You can also write directly to a record format. These new operations might allow you to simplify your program logic in some cases.

See the section on "Composite Keys" on page 88.

Changing Output Specifications

With an external file, output specifications must refer to the record name and not to the file name. You can do *one* of the following:

- Change the file name to the record name in the O-specifications.
- Use a format selector. See 9.8, "Format Selection" on page 67 for a discussion of the format selector.

The recommended method is to change to record names. In the example, we have changed the file name to the record name MASTREC.

Next, blank out the O-specification end positions in positions 40-43.

Figure 18 on page 86 shows how the program looks after the changes discussed above.

```
FMASTER  UF  E           K        DISK
.
.
IMASTEREC    01
I                                              STORE
I                                              SERIAL
I                                              CODE
I                                              AMOUNT
.
I            DS
I                                         1   8 CUST#
I                                         1   2 LOCATN
I                                         3   80NUMBER
.
.
C                    MOVELISTORE    KEY      8
C                    MOVE ISERIL    KEY
C            KEY     CHAINMASTER                    99
C  N99               MOVE CUST#     NEWCUS   8
C   99               MOVE KEY       NEWCUS
.
.
OMASTEREC         01
O                         NEWCUS
O                         CODE
O                         AMOUNT
```

Figure 18. RPG and Database - Partially Converted Program

11.1.6 Compiling the Program

We have not yet removed the field names on the input and output specifications; they are left for comparison only, to show the internal names that previously had been used.

Examine the compiler list to identify internally used field names, which are now not known to the external file description. These will be highlighted by error codes 4096 and 6109.

Figure 19 on page 87 and Figure 20 on page 88 show a compiler listing of the program after this first step of partial conversion.

```
                    S o u r c e   L i s t i n g          Page 1
          H
      100 FMASTER  UP  E                    DISK
           RECORD FORMAT(S):  LIBRARY PKLIB FILE MASTER.
                    EXTERNAL FORMAT MASTREC RPG NAME MASTREC
      200 IMASTREC     01
      300 I                                              STORE
* 4096                                                4096-******
      400 I                                              SERIAL
* 4096                                                4096-******
      500 I                                              CODE
      600 I                                              AMOUNT
      500  INPUT  FIELDS FOR RECORD MASTREC FILE MASTER FORMAT MASTREC.
  A000001                                                 1   2 LOCATN
  A000002                                                 3   80NUMBER
  A000003                                                 9  25 NAME
  A000004                                               P  26  280CODE
  A000005                                               P  29  312AMOUNT
          .
     1000 I       DS
     1100 I                                               1   8 CUST#
     1200 I                                               1   2 LOCATN
     1300 I                                               3   80NUMBER
          .
          .
     2000 C                    MOVELISTORE    KEY     8
     2100 C                    MOVE ISERIL    KEY
     2200 C            KEY     CHAINMASTER                    99
     2300 C  N99               MOVE CUST#     NEWCUS  8
     2400 C   99               MOVE KEY       NEWCUS
          .
          .
     4000 OMASTREC D        01
     4100 O                          NEWCUS
* 6109                           6109-******
     4200 O                          CODE
     4300 O                          AMOUNT
          OUTPUT  FIELDS FOR RECORD MASTREC FILE MASTER FORMAT MASTREC
  B000001  **NOT OUTPUT**            LOCATN     2  CHAR    2
  B000002  **NOT OUTPUT**            NUMBER     8  ZONE  6,0
  B000003  **NOT OUTPUT**            NAME      25  CHAR   17
  B000004                            CODE      28P PACK  4,0
  B000005                            AMOUNT    31P PACK  5,2
```

Figure 19. RPG and Database - Program after Partial Conversion - Page 1.

```
                    Source Listing          Page 2

   Additional Diagnostic Messages

* 7078   2200 FACTOR 1 LENGTH IS  8 BUT LENGTH OF KEY FIELD IS  2.

                    Message Summary
* QRG4096 Severity: 30  Number:  2
        Message . . . . :  RPG-Field-Name entry for externally
          described file is invalid. Specification line ignored.
* QRG6109 Severity: 30  Number:  1
        Message . . . . :  The Field name specified does not exist in
          Externally-Described record. Specification ignored.
* QRG7078 Severity: 30  Number:  1
        Message . . . . :  The Factor 1 length not same as First-Key
          field in file or record. Specification ignored.
```

Figure 20. RPG and Database - Program after Partial Conversion - Page 2.

Composite Keys

Many programs contain **composite keys** where the keyed access to the file was built from more than one field, usually by using a series of MOVE and MOVEL operations as illustrated in the example program.

In the following example, two fields (ISTORE and ISERIL) which have different definitions (alphanumeric and numeric respectively) are placed in a new alphanumeric composite field KEY, which is then used to access the file (MASTER). This is quite acceptable in RPG II; it compiles and operates without error.

```
C                   MOVELISTORE   KEY     8
C                   MOVE ISERIL   KEY
C           KEY     CHAINMASTER                   99
```

However, this is unacceptable with RPG/400 externally defined files with composite keys and will not compile, giving error 7078.

Each field of the key is separate and cannot be combined using MOVE and MOVEL, so the coding above has to be changed to use a KLIST and KFLD structure:

```
C           KEY     KLIST
C                   KFLD          ISTORE
C                   KFLD          ISERIL
C           KEY     CHAINMASTER                   99
```

The program will now compile without error.

A further problem now arises. This KEY field cannot be used as a data field in an operation.

The example requires an operation like this:

```
C   99              MOVE KEY      NEWCUS
```

The data field KEY has to be replaced with some other defined field, or by suitable MOVE and MOVEL operations with valid fields.

The best way to do this is to create a new field in a data structure, as follows:

```
I            DS
I                                        1   8 NEWKEY
I                                        1   2 ISTORE
I                                        3   80ISERIL
```

and replace the operation above with:

```
C   99               MOVE NEWKEY    NEWCUS
```

In spite of creating the data structure as shown, the field NEWKEY cannot be used to access the MASTER file as a composite key; the compiler error message 7078 will still be given for the statement:

```
C            NEWKEY   CHAINMASTER                  99
```

11.1.7 Adjusting Internal Field Names to Match Database Names

In the example, the fields STORE, SERIAL, and NEWCUS are unknown to the external file, which uses the fields LOCATN and NUMBER.

There are two ways to correct the unknown fields shown in the example:

1. The recommended way of making the names the same is to change each occurrence of the fields like STORE to LOCATN, and SERIAL to NUMBER. That is, make the field names comply with those chosen for the external file.

   Do not change the external file field names to fit the program, because this would mean changes to other programs using the file.

   Do the following:

   - Scan your program to see if the fields LOCATN and NUMBER are already defined and used elsewhere in the program. The compiler cross-reference listing can help you do this.
     - If these are already defined as fields in another externally described file, there *might* be a conflict. You must resolve this based on your knowledge of the file structures and the program logic. This could mean renaming fields in the externally defined files, recreating those files, and then making further program changes. See 11.1.8, "RPG and a Single Memory Area" on page 91 for details.
     - If LOCATN and NUMBER are used as fields in the program but not as a field in another externally described file, scan the program and replace all occurrences with another unused name. (Check the cross-reference listing to choose an unused name).
   - Now scan the program and do the following:
     - Remove the references to NEWCUS, because they are not really required for output.
     - Move the new data structure NEWKEY to CUST# in the calculation specifications. This will set the values of LOCATN and NUMBER as well.
     - Replace NEWCUS in the output specifications with the two fields LOCATN and NUMBER.

Also note that fields not used in input (like NAME) will not be flagged, but fields not used in output (like NAME) will be flagged as **NOT OUTPUT** in the list of output fields. These are not errors; they show that the fields concerned are not specified on the output specifications (lines 4200 and 4300), so they are not output to the file.

2. There is a new function which might be useful when adjusting external names. The "External Field Name" field in the I-specification (columns 21-30) can be used to specify the external name (EXTNAM). Columns 53-58 can be used to specify the corresponding internal name (PRGNAM). At run time, the program uses the internal name to access the field defined by the external name.

```
     I              EXTNAM                    PRGNAM
Example
     I              LOCATN                    STORE
```

Figure 21 on page 91 shows how the program looks after making the changes to resolve the field conflicts that are discussed above.

```
FMASTER  UF  E           K        DISK
.
.
IMASTEREC    01
I                                              LOCATN
I                                              NUMBER
I                                              CODE
I                                              AMOUNT
.
I         DS
I                                         1   8 CUST#
I                                         1   2 LOCATN
I                                         3   80NUMBER
I** ADD A DATA STRUCTURE TO REPRESENT THE COMPOSITE KEY AS DATA
I         DS
I                                         1   8 NEWKEY
I                                         1   2 ISTORE
I                                         3   80ISERIL
.
.
C*****************************************************************
C REPLACE MOVES WITH KLIST/KFLD FOR COMPOSITE KEY
C*****************************************************************
C**                   MOVELISTORE    KEY     8
C**                   MOVE ISERIL    KEY
C            KEY      KLIST
C                     KFLD           ISTORE
C                     KFLD           ISERIL
C            KEY      CHAINMASTER                    99
C**N99                MOVE CUST#     NEWCUS  8           NOT REQUIRED
C   99                MOVE NEWKEY    CUST#
.
.
OMASTEREC         01
O                           LOCATN
O                           NUMBER
O                           CODE
O                           AMOUNT
```

Figure 21. RPG and Database - Converted Program

11.1.8 RPG and a Single Memory Area

It might happen that a field name is used in a record format in one external file, and also used in a record format in a different file. If both of these files are read into a program, RPG sets up a single memory area for the field.

The compiler gives no warning or error message.

Thus a read from the first format followed by a read from the second format will overlay the field contents from the first format with the contents of the second format. This might cause undesirable results.

There is nothing new here for RPG II users, since RPG II behaves the same way.

If you need to keep two separate memory areas for the two different occurrences of the same field, you can either:

- Rename one of the fields in the database.
- Redefine one of the fields in the program. This can be done very simply by using the "External Field Name" function on the I-specification. This is a new function for RPG II users. See Chapter 5 of the *RPG/400 Reference* manual for details.

This same discussion holds true for display file fields as well.

11.2 RPG and Display Files

This section gives a step-by-step guide for the process of changing RPG programs to allow for the use of externally described display files. Refer to 13.2, "Multiple Requester Terminal Programs" on page 130 for a description of the additional steps needed to convert MRT programs into SRT programs.

Display files and disk files are handled in a very similar way on AS/400. Therefore, the steps needed to convert display files are much like the steps explained for disk files.

At this point we already have the display file DDS, which was generated by the migration utility. In fact the System/36 Environment uses the display file created from the DDS.

The DDS source is in the file QDDSSRC in the conversion library. (It was copied there from QS36DDSSRC, according to the directions in 3.7, "Moving Selected System/36 Source to New Library" on page 16.)

Recreate the display file in the conversion library.

The changes needed fall into two groups:

1. Changes needed to run the AS/400 program with program-described display files. These are the minimum changes needed and must be made before the program will compile and run with the RPG/400 compiler.
2. Additional changes needed to run the program and make the display file fully external. You may not get many benefits from this step unless you have common screen formats (for example, common heading, footing, or error screens) which are used by many programs. If this is not the case, we recommend you only complete the minimum changes.

11.2.1 Old Programs with Display Files that will not Convert

You may have programs with display files that will not convert to AS/400 display files because these programs were originally written for System/34 or earlier systems.

These files will be CONSOLE, CRT, or KEYBORD file types using the KEYnn and SETnn operations. While these will recompile in the System/36 Environment, they have no equivalent in RPG III.

In this case you will have to design and create a suitable display file and rewrite the program in RPG III.

11.2.2 Minimum Changes for Program-Described Display File

Section 9-12 of the *RPG/400 User's Guide* has a good discussion of program-described workstation files. We suggest that you have a copy available.

The following changes must be made in order to convert your application, whether it uses program-described files or externally described files. Refer to 11.2.3, "Additional Changes for Externally Described Display Files" on page 99 for additional changes needed for externally described display files.

Changing File Specifications

We recommend that you change the file name on the F-specification to the name of the external display file. (This is probably xxxxxxFM, where xxxxxx is the program name.) If you do this you will not need an OVRDSPF statement at run time to tie the internal and external names together. An alternative would be to rename the display file.

If the file is a demand file, replace D in column 16 with F for fully procedural, that is, read and written by explicit operations in the calculation specifications. (If column 16 is P for primary, leave it alone, unless asked to change it in "RPG/400 Display File Cycle Difference" on page 94.)

Code an F in column 19 (file format) to indicate that the file is a program-described file.

You might also need to specify a PASS *NOIND entry on the file continuation specifications. If the program uses a display file containing the INDARA DDS keyword, then PASS *NOIND must be specified. The display files generated by the migration utility are all created with INDARA, so if you are converting a program that uses such a file, you must specify PASS unless you remove the INDARA keyword from the file. However, removing INDARA will change the buffer that is returned to the program and will affect other programs that use the file, so we recommend leaving INDARA in the file and specifying PASS *NOIND in the program.

Remove the K continuation for the FMTS line if the program has one.

Changing Input Specifications

Change the file name to the same name used on the F-specification.

Changing Calculation Specifications

Change the file name to the same name used on the F-specification.

On System/36 and in the System/36 Environment, when the first input or output operation issued by your program in the first cycle was a READ, the program would receive either read under format (RUF) data, program data from the procedure, or a blank record. This initial input is normally used to trigger the output of the first screen format, but it may also be used to condition calculations before the first output. If your program uses this initial input, read and follow one of the recommendations in "RPG/400 Display File Cycle Difference" on page 94. Output specifications may also change as a result. Return to this point when you have finished.

When you perform a read on a WORKSTN file on the System/36, all formats currently on the display are read back into your program. All input and input/output fields are passed back to your program, even if they have not changed since the formats were written. **On AS/400, only the last format written is read back to your program.** Also, AS/400 passes back to your program only those fields that have changed.

These differences might require program changes if you write multiple formats or overlays to build up displays before reading. Multiple READ operations, one per format, can retrieve the data into the program.

The Modified Data Tag attribute DSPATR(MDT) is automatically specified when you migrate SFGR source members from System/36 to AS/400. It is a good idea to keep this keyword to be sure that all input fields are returned to your program.

Changing Output Specifications

Change the file name to the same name used on the F-specification.

Multiple Writes and the INVITE Keyword

If the display file is used only by native AS/400 programs and it is not a multiple device display file, then the INVITE keyword may be removed. It is not needed and might cause poor performance. If the display file is still being used by System/36 Environment programs, then the INVITE is needed, but you should still condition its use to occur only on the last write before a read to improve performance.

RPG/400 Display File Cycle Difference

The RPG/400 logic cycle differs from the RPG II logic cycle in the way that the initial workstation cycle is processed.

Before first detail input, the RPG II logic cycle issues a read for first detail input. This is typically a blank record.

With the RPG/400 compiler, a format must exist at the device before an input operation can occur.

This means that to convert your interactive program to native AS/400, you will have to change the program logic. These changes are not difficult. There are three different methods:

1. This is the simplest method, but will not work in all cases.

   If the first-time input is used to set on an indicator, which is then used only to output a format to the display (that is, if the indicator is not used to perform any calculations prior to writing the first format), then write out that first format with the 1P indicator at first detail output time. (The first detail output time occurs before the first detail input.)

   If you are using an EXCPT to write the first format, and the EXCPT is conditioned by the first-time input indicator, and if no other calculations are conditioned by the indicator, remove the EXCPT operation from the calculations and output the first format at 1P time. You may have to add extra output specifications if the same format is written to with EXCPT on subsequent cycles.

See Figure 22 and Figure 23 for before and after examples.

```
FAF01FM  CP  F     123            WORKSTN
IAF01FM  AA  01    1 C
IAF01FM  BB  02    1 CA
C   02                MOVEL'SECOND  'WRK13  13
OAF01FM  D        01
O                                   K3 'ONE'
O                          WRK13    14
OAF01FM  D        02
O                                   K3 'ONE'
O                          WRK13    14
```

Figure 22. Displays - System/36 Program with no First Cycle Calculations

```
F*****************************************************
F*   WRITE SCREEN AT OUTPUT TIME
F*****************************************************
FAF01FM  CP  F     123            WORKSTN
IAF01FM  BB  02    1 CA
C   02                MOVE 'SECOND  'WRK13  13
OAF01FM  D        1P
O                                   K3 'ONE'
O                          WRK13    14
OAF01FM  D        02
O                                   K3 'ONE'
O                          WRK13    14
```

Figure 23. Displays - AS/400 program with No First Cycle Calculations

2. This method is a little more work but is applicable in all cases. It is used by PTK. If you use PTK to convert your programs, you will need to carefully check the indicators used and make sure that the correct first output is done. Also make sure that any calculations for the first-time screen are processed only once.

   If your program logic is dependent on doing calculations before the first format is written to the device, use this method. It does not alter the input or output specifications and avoids extensive changes to the program logic. Also, it eliminates the need to create DDS for a dummy format and write out that dummy format just to allow the first-time input. (A dummy format would, for example, simply display "Press Enter to continue").

   First, make the file fully procedural (F in column 16) to allow the READ operation code to be added to the logic. Then add lines of code as shown in the example. If your display file was a demand file, these lines already may be present. There are some optional lines that might help you avoid testing the first-time indicator for a large number of lines of code that will not be processed on the first cycle.

See Figure 24 on page 96 and Figure 25 on page 97 for one example. See Figure 26 on page 98 and Figure 27 on page 99 for another example that handles the first-time processing in a better way because fewer changes may be needed to convert this program to use fully externally described screen files.

3. The third method of conditioning the write is to use the INZRCD DDS keyword. A read to a format with the INZRCD keyword causes a write of the same format to be done by workstation data management before the read is done. This method has not been tested.

```
     C*****************************************************
     F*   FIRST TIME BLANK SCREEN SETS AN INDICATOR WHICH
     F*   CONDITIONS CALCULATIONS AND INITIAL OUTPUT
     C*****************************************************
     FAF01FM  CP        123            WORKSTN
     F                                            KFMTS   AFEXMPFM
     I* FIRST-TIME BLANK
     IAF01FM  AA  01    1 C
     IAF01FM  BB  02    1 CA
     IAF01FM  CC  03    1 CB
     C   01               MOVEL'1ST TIME'WRK13  13
     C   02               MOVEL'SECOND  'WRK13
     C   03               SETON                    LR
     OAF01FM  D        01
     O                                  K3 'ONE'
     O                                   1 'A'
     O                        WRK13     14
     OAF01FM  D        02
     O                                  K3 'ONE'
     O                                   1 'B'
     O                        WRK13     14


This display file DDS will have been created during migration.
     A*****************************************************
     A   DISPLAY FORMAT FOR THE ABOVE PROGRAM
     A*****************************************************
     A                                    INDARA
     A                                    INVITE
     A          R ONE
     A            ID             1   B  2  2
     A            NAME          13   B 10 10
```

Figure 24. Displays - System/36 Program with First Cycle Calculations

```
     F*****************************************************
     F*   ADDITIONAL CODE ADDED TO AVOID THE FIRST-TIME READ
     F*   INPUT AND OUTPUT SPECIFICATIONS ARE UNCHANGED
     F*****************************************************
     FAF01FM  CF  F     123           WORKSTN
-->  F************************ REMOVE THE FORMATS REFERENCE
     F**                                        KFMTS   AFEXMPFM
     I* FIRST-TIME BLANK
     IAF01FM  AA  01   1 C
     I                                       1   1 ID
     IAF01FM  BB  02   1 CA
     I                                       1   1 ID
     IAF01FM  CC  03   1 CB
     I                                       1   1 ID
     C*****************************************************
-->  C* INDICATORS 04 AND 05 ARE USED - YOU WILL NEED TO CHECK
     C*      THE PROGRAM FOR VALID INDICATORS TO USE HERE.
     C* NOTE: THE DISPLAY FILE WILL READ ON THE SECOND AND
     C*      SUBSEQUENT CYCLES ONLY.
     C*****************************************************
-->  C   04               READ AF01FM                   05
-->  C  N04               SETON                     0401
-->  C                    SETOF                     05
     ******************************************************
     C   01               MOVEL'1ST TIME'WRK13  13
     C*****************************************************
-->  C*  OPTIONAL- BYPASS UNWANTED FIRST-TIME CALCS
     C*       IF THAT IS NOT ALREADY CODED INTO THE PROGRAM.
     C*  CAN BE ADDED AFTER ALL FIRST TIME CALCS COMPLETED
     C*****************************************************
-->  C   01               GOTO ENDTAG
     C   02               MOVEL'SECOND  'WRK13
     C   03               SETON                     LR
     C*****************************************************
-->  C* MATCHING TAG FOR GOTO
     C*****************************************************
-->  C           ENDTAG    TAG
     OAF01FM  D        01
     O                                  K3 'ONE'
     O                                   1 'A'
     O                        WRK13     14
     OAF01FM  D        02
     O                                  K3 'ONE'
     O                                   1 'B'
     O                        WRK13     14
```

Figure 25. Displays - AS/400 Program with First Cycle Calculations

```
FWORKSTN CP  F      64           WORKSTN
IWORKSTN NS       1 CS
I* FORMAT-SELECTOR
I                                      1   1 #FID
I                                      2   2 AEPTYP
I        NS       1 CA
I* FORMAT-DESCR
I                                      1   1 #FID
I                                      2  16 AEPDES
I* FORMAT-BLANK
I        NS
C*-------------------------------------------------
C*          M A I N L I N E
C*-----------------------------------------HHLLEE
C                    MOVE *BLANK    #MSG   40        CLEAR
C                    SETOF                      808199
C*
C           #FID     CASEQ'S'       SUB100           SELECTOR
C           #FID     CASEQ'A'       SUB200           DESCRIPTION
C                    CAS            SUB900           BLANK
C                    END
C*-------------------------------------------------
C           SUB900   BEGSR                      BLANK: INITIALISE
C*-----------------------------------------HHLLEE
C                    SETON                      80    PUT SELECTOR
C                    MOVE 'HALLO'   #MSG             1ST MESSAGE
C                    ENDSR
C*
C*
OWORKSTN D        80
O                                   K8 'SELECTOR'
O                         UDATE Y    8
O                         #MSG      48
O        D        81
O                                   K6 'DESCR '
O                N20      AEPDES    15
O                 81      #MSG      55
O        DR       LR
```

Figure 26. Displays - System/36 Program with First Cycle Calculations

```
FJCS104FMCF  F     164           WORKSTN
IJCS104FMNS       1 CS
I* FORMAT-SELECTOR
I                                       1   1 #FID
I                                       2   2 AEPTYP
I        NS       1 CA
I* FORMAT-DESCR
I                                       1   1 #FID
I                                       2  16 AEPDES
C*----------------------------------------------------
C*          M A I N L I N E
C*----------------------------------------------------
C* These 3 lines added for preprocessing.
C* 1st cycle will not read but do preprocessing,
C* then output
C* 2nd cycle will read last format written,
C* and continue with normal program logic.
C*-----------------------------------------------HHLLEE
C   01               READ JCS104FM                 02
C  N01               SETON                    01
C                    SETOF                    02
C*-----------------------------------------------HHLLEE
C                    MOVE *BLANK    #MSG   40        CLEAR
C                    SETOF                    808199
C*
C           #FID     CASEQ'S'       SUB100           SELECTOR
C           #FID     CASEQ'A'       SUB200           DESCRIPTION
C                    CAS            SUB900           BLANK
C                    END
C*----------------------------------------------------
C           SUB900   BEGSR                    BLANK: INITIALISE
C*-----------------------------------------------HHLLEE
C                    SETON                    80    PUT SELECTOR
C                    MOVE 'HALLO'   #MSG             1ST MESSAGE
C                    ENDSR
C*
C*
OWORKSTN D        80
O                                   K8 'SELECTOR'
O                         UDATE Y    8
O                         #MSG      48
O        D        81
O                                   K6 'DESCR '
O                N20      AEPDES    15
O                 81      #MSG      55
```

Figure 27. Displays - AS/400 program with First Cycle Calculations

11.2.3 Additional Changes for Externally Described Display Files

The changes described below are in addition to those that have already been described in the section 11.2.2, "Minimum Changes for Program-Described Display File" on page 93.

It is likely that more changes will be needed for displays than for disk files. On System/36 there is an external display description and format member (as SFGR S- and D-specifications), with the data passed between the program and the workstation file via a buffer, that does not require matching field names between the program and the display SFGR source. Therefore there is a possibility of extensive conflict between the two sets of field names in the program and the SFGR specifications.

Migration of this SFGR source to DDS further modifies screen field names if you have used array names that were allowed with the System/36. An array name such as ARRY,1 would be converted to ARRY$1 to be compatible with DDS.

So before you make changes to the programs you should review the display file DDS to:

- Resolve the record (format) names. These cannot be the same as any field names.
- Resolve all of the field names (which may be an extensive job!). Since most displayed fields originate from database records, you should use the field names that have been chosen for the database records wherever appropriate.
- Change output of literals and output field editing in the program to code these into the display file DDS for externally defined files. Literals in the DDS can be conditioned with indicators as in the original program. RPG/400 does not allow program editing of fields or output literals with externally defined display files.

EXAMPLE:

Displayed output in the original program:

```
0                        NUMBERZ   7
0                        AMOUNT   21 ' ,  0,   .  CR'
0                99              34 'ERROR MESSAGE'
```

Should be coded with in the display file DDS as:

```
A            NUMBER       7  00  4  2EDTCDE(Z)
A            AMOUNT       9S 20  5  2EDTWRD(' ,  0,   .  CR')
A  99                          20 10'ERROR MESSAGE'
```

and the editing and literal removed from the program:

Then recreate the display file using the changed DDS.

Changing File Specifications

Replace F in column 19 (File Format) with E to indicate that the file is externally described.

Blank out the record length.

Changing Input Specifications

Make the following changes:

- For the file entries: Change the file name entries to the record names coded in the display file DDS. Then blank out the sequence, number, and option fields in columns 15-18, and the record identifying codes in columns 21-41.
- For the field entries: Blank out the P/B/L/R, field location from and to, and decimal position entries in columns 43-52.

Replacing File Names with Format Names

With an external display file, input and output specifications must refer to the record name and not the file name.

On System/36, when a data stream from the display is read in, it is examined according to coded input specifications, and the corresponding record identifying indicator is set on. For an AS/400 externally described file, this does not happen. If the program uses the RPG cycle, the format read in will be the last format written out. If the program uses a procedural file, the format named in the READ will be returned. In any case, no test is done on the data returned to the program. The record indicator that is set will be the record identifying indicator (if any) specified in columns 19 and 20.

Adjustments are needed in cases where the program logic is like the following example.

The System/36 display file has a single, general purpose format where the fields displayed are conditioned by indicators. There is a single display format record called ORDER that displays:

- Order headers (indicator 01, with a record identifying code of A)
- Order details (indicator 02, with a record identifying code of B)
- Order totals (indicator 03, with a record identifying code of C).

When the display file is read back into the RPG II program, the record-identifying indicator is set on according to whether the identifying code is A, B, or C.

When such a program has the display file name replaced by the format name and is recompiled, error message RPG 4111 indicates that the record format is defined more than once. In this case the program must test the incoming record identifier and set on the corresponding indicator.

```
Thus:

     IHEADER  AA  01    1 CA
     I                                      1   1 ID
     I                                      ........
     IDETAIL  BB  02    1 CB
     I                                      1   1 ID
     I                                      ........
     ITOTAL   CC  03    1 CC
     I                                      1   1 ID
     I                                      ........

  All of this input will be replaced by, calculation
     specifications to test for the appropriate input.

     IORDER
     .
     C* TEST INCOMING RECORD TO SEE WHAT TO DO NEXT
     C* SET ON THE SAME RECORD IND AS THE INPUT DID BEFORE.
     C           ID        COMP 'A'                    01
     C           ID        COMP 'B'                    02
     C           ID        COMP 'C'                    03
     C*
```

Figure 28. Displays - Setting Record Indicator in Calculations

The format record name must not be the same as the name of a field in the format. If it is, the program will not compile, even though the display file will be created without error. Change the record name on the DDS if you need to avoid this situation.

Calculation Specifications

No changes are needed. Note that on AS/400 you can write and then read display formats directly from RPG/400 by using EXFMT. This might allow you to simplify your program logic and write new applications more easily.

Changing Output Specifications

With an external display file, output specifications must refer to the record name and not to the file name.

- Change the file name to the record name appropriate to the format.
- Remove all the lines that name the output format. These are the lines with K in column 42; for example, K8 'SELECTOR'.
- Remove any release operations; use LR or RETRN instead.
- The only valid entries for field descriptions are output indicators, field name, and blank after. Blank out any other fields.
- Output constants and literals and field editing must be removed. Code these in the display file DDS and condition them on indicators. (See 11.2.3, "Additional Changes for Externally Described Display Files" on page 99.)

Compiling the Program

We have not yet removed the field names on the input and output specifications; they are left so that when the program is compiled, we will be able to see both internal and external names.

Examine the compiler list to identify field names used internally that now are not known to the external file description. These field names will be highlighted by error codes 4096 and 6109, exactly the same as for external database files.

The following example uses an externally described display file CONVFM with record(format) CONVREC.

Fields such as KEY, DD, MM, YY, PART, and PROD are among those in the original program input and output specifications. The process described here is the same as that used for externally defined database files.

No calculations are shown, because this topic was dealt with under changes for externally-defined database files.

```
 F*****************************************************************
 FCONVFM  CF  E                    WORKSTN
 ICONVREC     01
 I                                            KEY
* 4096                                       4096-******
 I                                            DD
* 4096                                       4096-******
 I                                            MM
* 4096                                       4096-******
 I                                            YY
* 4096                                       4096-******
 I                                            PART
* 4096                                       4096-******
 I                                            MSG
* 4096                                       4096-******
 I*****************************************************************
  INPUT  FIELDS FOR RECORD CONVREC FILE CONVFM FORMAT CONVREC.
                                             1   1 ID
                                             2   4 KEY1
                                             5   7 KEY2
                                             8  13 DATE
                                            14  17 ITEM
                                            18  19 FL001
                                            20  29 NOTE
 .
 OCONVREC D        01
 O                        KEY
* 6109                 6109-******
 O                        DATE
 O                        PROD
* 6109                 6109-******
 O                        FL001
 O                        NOTE
  OUTPUT  FIELDS FOR RECORD CONVREC FILE CONVFM FORMAT CONVREC.
  **NOT OUTPUT**          ID         1  CHAR    1
  **NOT OUTPUT**          KEY1       4  CHAR    3
  **NOT OUTPUT**          KEY2       7  CHAR    3
                          DATE      13  CHAR    6
  **NOT OUTPUT**          ITEM      17  CHAR    4
                          FL001     19  CHAR    2
                          NOTE      29  CHAR   10

                      M e s s a g e   S u m m a r y
* QRG4096 Severity:  30   Number:    5
     Message . . . . :   RPG-Field-Name entry for externally-described
          file is invalid. Specification line ignored.
* QRG6109 Severity:  30   Number:    3
        Message . . . . :   The Field name specified does not exist in
          Externally-Described record. Specification ignored.
```

Figure 29. Displays - Compiler Output for External File

11.2.4 Adjusting Internal Field Names to Correspond with Display File Names

In the previous example the fields PART and PROD are unknown to the external file, which uses the field ITEM. There are two ways to correct the unknown fields shown in the example:

1. The recommended way of making the names the same is to change each occurrence of the fields like PART and PROD in the program to ITEM.

   (Do not try to change the name of the display file field to PART. This makes one of the names match, but still leaves the name PROD unmatched. You would then have to recreate the DDS with the name PART, and this would mean changes to any other programs using the display file.)

   Do the following:

   - Scan your program to see if the field ITEM is already defined. The compiler cross-reference listing can help you do this.
     - If ITEM is already defined as a field in another externally described file, there *might* be a conflict. You must resolve this based on your knowledge of the file structures and the program logic. See 11.1.8, "RPG and a Single Memory Area" on page 91 for details.
     - If ITEM is used, but not as a field in another externally described file, scan the program and replace all occurrences with another unused name. (Check the cross-reference listing to choose an unused name.)
   - Now scan and replace all occurrences of PART and PROD with ITEM.

   Fields not used in input (like ID) will not be flagged, but fields not used in output (like KEY1) will be flagged as **NOT OUTPUT** in the list of output fields. These are not necessarily errors; they show that the fields concerned are not specified on the output specifications, so they are not output to the file.

2. There is a new function that might be useful when adjusting external names. The "External Field Name" field in I-specification (columns 21-30) can be used to specify the external name (EXTNAM). Columns 53-58 can be used to specify the corresponding internal name (PRGNAM). At run time, the program uses the internal name to access the field defined by the external name.

```
     I              EXTNAM                      PRGNAM
For example:
     I              ITEM                        PROD
```

3. The field KEY is required to be output. This field must be replaced with KEY1 and KEY2 in the output. To reference KEY in the body of the program, a data structure must be created where KEY is structured from KEY1 and KEY2. The same applies to referencing the fields DD, MM and YY within the DDS-defined field DATE. This is shown in the next section.

Adding Additional Fields

Here is an example of the fields from the DDS used above for the input- and output-specifications of the record format CONVREC.

DDS FORMAT	ID	KEY1	KEY2	DATE	ITEM	FL001	NOTE
I-SPECS.	ID	KEY		DD MM YY	PART	MSG	
O-SPECS.	ID	KEY		DATE	PROD	NAME	

The definitions of the record needed by the program do not match the display file DDS. This is because of the overlapping, subdivided, or grouped fields, like the DATE and MSG fields in our example.

- The MSG field combines both FL001 and NOTE fields.
- The DDS has a field (ID) which is not referred to in the I-specifications or in the calculations.
- Two or more fields from the DDS (KEY1 and KEY2) are referred to by a single name (KEY) in the program. KEY is not a valid field name entry for the externally described file.
- A single field (DATE) from the external file is known as three fields (DD, MM, and YY) in the program.

In these cases you can redefine the record layout of the external file by using a data structure. Data structures can be used to subdivide an input field so that the program can refer to the entire field or just to the individual subfields, as shown below. They can be used to redefine an area and group fields together. See the *RPG/400 User's Guide* for details.

```
I        DS
I                                      1   6 KEY
I                                      1   3 KEY1
I                                      4   6 KEY2
I                                      7  12 DATE
I                                      7   8 DD
I                                      9  10 MM
I                                     11  12 YY
I                                     13  24 MSG
I                                     13  14 FL001
I                                     15  24 NOTE
```

This would be included in the program. Therefore, the adjusted program will now be:

```
FCONVFM  CF  E                    WORKSTN
ICONVREC     01
I                                              KEY1
I                                              KEY2
I                                              DATE
I                                              ITEM
I                                              FL001
I                                              NOTE
I***************************************************************
I            DS
I                                        1   6 KEY
I                                        1   3 KEY1
I                                        4   6 KEY2
I                                        7  12 DATE
I                                        7   8 DD
I                                        9  10 MM
I                                       11  12 YY
I                                       13  24 MSG
I                                       13  14 FL001
I                                       15  24 NOTE
.
.
OCONVREC D        01
O                         KEY1
O                         KEY2
O                         DATE
O                         ITEM
O                         FL001
O                         NOTE
```

Figure 30. Displays - Program Modified for External File

11.2.5 A Note on UDATE

There is a new way of handling the field UDATE on AS/400. It is coded on the DDS as a field-level keyword that displays the current job date. So instead of coding:

```
A            DATE          8   0 02003
```

Use the new method:

```
A                                  02003DATE EDTCDE(Y)
```

11.2.6 Removing Internal Field Descriptions

At this stage you might wish to remove the I- and O-specification lines containing the old internal field names. But be aware of the following requirements:

- Input specifications might still be required if:
  - Record identifying indicators are to be specified.

- A field within the record is to be renamed for the program.
- Control level or matching field indicators are to be used.
- Field indicators are to be used.

- Output specifications might still be required because:
  - Entries in positions 7 through 37 of the record identification line determine the conditions under which the records are to be written.
  - When using field description output specifications for externally described files, only the fields specified are placed on the output record. *ALL can be specified in the field name to include all fields in the record. See the *RPG/400 Reference* manual for further details.
  - Output indicators on the field description may be used. If this is the case, you cannot remove the field descriptions.

Recompile and check for a successful compilation. Of course, the program still must be tested thoroughly.

11.3 Additional RPG Considerations

11.3.1 General

Here are additional differences between System/36-compatible RPG programs and RPG/400 programs.

- BSCA, CONSOLE, CRT, and KEYBORD files are not supported in RPG/400. These files are not converted by PTK, so you will have to convert them manually.
- EXIT and RLABL operations are not supported in RPG/400, nor are any Assembler routines that are called by these operations. You will have to create equivalent functions and use RPG/400 CALL and parameter list as appropriate.
- Files used by RPG/400 programs must exist prior to being opened. There is no equivalent to the DISP-NEW keyword in the // FILE statement for output files.
- RPG/400 programs cannot mix access to display devices and ICF sessions together through a single WORKSTN file.
- INFDS for RPG/400 is different than for RPG II. For example, the location and values of a variety of fields, including file status return codes, have changed.
- Numeric fields are internally stored in packed format by default in RPG/400 unless the field is defined in a data structure subfield as zoned decimal. RPG II uses zoned format as the default. This change in defaults can cause parameter mismatches on program call interfaces.
- Numeric fields should be initialized before they are referenced, particularly if these are work fields to be output. All output fields **must contain valid data.**
- The use of USRDSPMGT in a display file might affect the use of other keywords or functions.

11.3.2 Changing an RPG II MRT Program to an RPG/400 SRT Program

The basic steps needed to convert an RPG II MRT program to an RPG/400 single requester (SRT) program are:

- Remove the K continuation lines for ID, IND, NUM, and SAVDS, so that RPG/400 will treat the display file as a single device file instead of a multiple device file. Leave the data structure (SAVDS) in the program, though, or you will get undefined field names.
- Remove all of the "release" operation codes in both calculations and output lines. The REL operation code and the release indicator on the output specification are not allowed on AS/400 for single device files.
- Recompile the program.

Other changes to the program might be required, particularly where logic has been developed to share a common output file for multiple users. In this case, we recommend using a user-related member within the output file with the name of the member, based in part, on the user ID.

11.3.3 A Note on the L0 Indicator

The L0 indicator is not supported on RPG/400 output specifications. If it is present it will give a compile error.

Since L0 is always on, it has no effect at output time. The program should work the same with the L0 indicator removed.

12.0 COBOL Considerations

At this stage we have generated DDS for the AS/400 external database files and for the AS/400 external display files. But the programs still contain program-described files, which may differ in record layout from program to program.

There are three methods of COBOL conversion:

1. Convert only the necessary differences between the System/36 Environment and the AS/400 native mode. This means that you do not include externally described files in your COBOL program, but you still change incompatible coding statements and add some program logic. This is the fastest way to get a native COBOL program.

   On the AS/400 there is no "MUST" for externally described files within a program. It is the programmers responsibility to avoid all decimal data errors. This means that the internally and externally described files have the same field description for numeric fields.

2. Convert the programs that contain externally described files, but still are using the old file description within the WORKING STORAGE SECTION. Otherwise add record formats using the COPY DDS statement. Also do any necessary changes and add some program logic. This process requires additional coding effort, but there are some situations where you can use only this kind of conversion.

   For example. If you have level 88 fields in your file description, there is no way to create the same function with DDS. The only way to solve this problem is to use your old internal file description in the WORKING STORAGE SECTION and move the data of the externally described file after the read to this area. Process your data and move the data back to your externally described file. Then write the record to your database.

3. Convert the programs that contain only externally described files and do the necessary changes. This will cost the most effort, and you should test your program very carefully. In most cases you have to change your program logic and your program flow to get the correct results.

When converting System/36 Environment COBOL programs to native AS/400, consider the following areas and make changes accordingly:

- Miscellaneous.
- Externally described database files.
- Display files.

To convert the program, copy the program out of your source file QS36SRC into the source file QLBLSRC. Change the source type of the file from CBL36 to CBL. There is no need to bring your COPY members into QLBLSRC, but in this case you have to qualify your COPY statements. Also if you have to change something in your COPY members, you must do this in your original version. Therefore, we recommend that you also copy your COPY members into QLBLSRC.

12.1.1 Special Cases for COBOL

Before you start to work with the PTK, you should note the following problems and rules for COBOL:

COBOL source code having REDEFINES and OCCURS within the same statement and level must be restructured to produce two statements with different levels; the REDEFINES statement at a higher level and the OCCURS at a lower level.

If the PTK does not recognize the COBOL filler fields, you will get the PTK message "Duplicate Fields". Change these fields to real field names if necessary. Do not select a filler field in the PTK without changing the name. Later in the conversion process you can create a logical file over the physical file. Within the logical file description you select only the fields you really need in your program. This prevents fields from being accessed like the COBOL FILLER keyword.

The PTK removes the dashes from long COBOL names and shortens their length to 10. This might lead to duplicate names.

```
Program Defined:              PTK Shows:
05 MASTER-CODE-ID             MASTERCODE
05 MASTER-CODE-REF            MASTERCODE      Duplicate
```

One of these DDS field names must be changed. The program-defined field names may remain the same, since they are converted in their original lengths to automatically use the DDS field-level ALIAS keyword. When specifying the substitute name in DDS, change any hyphen "-" to an underscore "_". COBOL will automatically convert the underscore character to a hyphen during program compilation. For example, a field name of MASTERCODE and a program-defined name of MASTER-CODE-ID would generate a DDS field name of MASTERCODE and with ALIAS(MASTER_CODE_ID).

For a more detailed example of such DDS, refer to Figure 33 on page 125.

PTK creates DDS that contains concatenated key fields. This is not allowed in COBOL when you use RECORD KEY IS EXTERNALLY DESCRIBED KEY within your SELECT clause. You really must change your DDS specifications so that they contain real key fields.

This is the created DDS for the key.

```
     CATKEY      CONCAT(FIELD1 FIELD2)
K    CATKEY
```

You should change the DDS like this.

```
     FIELD1
     FIELD2
K    FIELD1
K    FIELD2
```

Now you can use the file as an externally-described file within your COBOL programs.

12.2 Miscellaneous

In this section you will find some differences between System/36 Environment COBOL and AS/400 native COBOL. Adjustments must be made for these differences within all COBOL conversion steps.

12.2.1 COPY Books

The native compiler requires a qualified name for the COPY members if they do not reside in the same library and source file as the main source. In the System/36 Environment the compiler assumes that the source will always be in the file QS36SRC in the specified library. If you have COPY statements in your program, we recommend that you qualify these statements. The qualifier should always contain the library and the source file with your COPY members.

```
Old Statement: COPY XYZ.
New Statement: COPY XYZ OF MAINLIB-QS36SRC.
```

or

```
Old Statement: COPY XYZ OF COPYLIB.
New Statement: COPY XYZ OF COPYLIB-QS36SRC.
```

In this example we assumed that only the main program has been copied to the QLBLSRC source file. If you also have copied your COPY members into QLBLSRC, change your command from QS36SRC to QLBLSRC.

There are two good reasons to use qualified copy statements:

1. The performance of the COPY statement is faster because you do not need to search your whole library list.
2. You cannot get the wrong COPY members if your library list is not in the correct sequence.

12.2.2 PROCESS Statement

There are some differences between System/36 Environment COBOL and native COBOL in the PROCESS statement.

1. In System/36 Environment the parameter NUMBER is the default of the PROCESS statement. Native it is NONUMBER.
2. In System/36 Environment the parameter OPTION is the default of the PROCESS statement. Native it is NOOPTION.
3. NOLVL/LVL indicates the FIPS flagging in the System/36 Environment. NOFIPS/FIPS indicates the FIPS flagging in native mode.
4. FLAGW and FLAGE are used in the System/36 Environment to list the error messages of a compiler. FLAG(nn) is used in native mode.
5. NOCMPAT/CMPAT is not supported in native mode.
6. NOBLOCKIO is not supported in native mode.
7. LIBRARY is not supported in native mode.
8. NODUMP/DUMP is not supported in native mode.
9. DEBUG/NODEBUG is not supported in native mode.
10. GRAPHIC is not supported in native mode.

11. NOSYNTAX/SYNTAX is not supported in native mode.

For more information, see Chapter 2 of the *COBOL/400 USER'S GUIDE* and Chapter 3 of the *System/36-Compatible COBOL User's Guide and Reference.*

You are more flexible if you are not using the PROCESS statement and you key your parameter within the CRTCBLPGM command. You also can change the command for all programmers by using the CHGCMDDFT command. Now every programmer receives the same compiler output, but still has the ability to overwrite these parameters.

12.2.3 MEMORY SIZE Clause and Source/Object Computer

The MEMORY SIZE clause should be removed from the Configuration Section. Also change the SOURCE- and OBJECT-COMPUTER clauses from IBM-S36 to IBM-AS400.

12.2.4 Literals

Alphanumeric literals are enclosed in apostrophes on System/36. This produces a compiler warning message, "Unexpected literal delimiter found" on AS/400. To prevent this message, you might put the option APOST into the PROCESS statement or the *APOST into the CRTCBLPGM command. You also can use the PDM search and replace functions to change the literal delimiter from an apostrophe to a quotation mark.

12.2.5 USAGE IS COMPUTATIONAL

The ANSI standard states that this data type is mapped to the most efficient commercial data support for the specific machine. The following is what COMPUTATIONAL maps to:

- S/3,S/32,S/36 - Zoned Decimal, same as USAGE IS DISPLAY.
- S/38,AS/400 - Packed Decimal, same as USAGE IS COMP-3.
- S/370 - Binary, same as USAGE IS COMP-4.

When you convert a program from System/36 Environment , look for the following field definitions in your program:

```
01 FIELD-NAME       PIC S9(03) COMP VALUE ZERO.
```

In the System/36 Environment this field is a zoned decimal field. In native mode it is a packed decimal field. If you have fields like this in your internal file description, you must change them to get the correct file length. For example:

```
01 FIELD-NAME       PIC S9(03) VALUE ZERO.
```

Because packed decimal fields are used on the AS/400, there are some things that you should consider when declaring COBOL variables. The machine stores packed decimal data in byte aligned fields. For example:

```
01  A  PICTURE 99 USAGE IS COMP-3 VALUE 12.
```

would map in storage as a two-byte field

```
|01|2F|<- in HEX
```

Because four bits (one digit) are used for the sign of the number, you can see that all packed decimal fields are always capable of holding an *odd* number of

decimal digits. In order for the AS/400 to provide the illusion of having an even number of decimal digits, the compiler and translator must generate extra "AND BYTE" instructions to strip (force to zero) the left-most 4 bits. The recommendation here is that *all* COMP-3 fields be declared, where possible, to an odd number of decimal digits.

If you are converting COBOL programs from a S/36, it is highly recommended that you change heavily used computational items to packed decimal. The overhead you pay for using zoned decimal on the AS/400 is the additional conversions to and from packed decimal so the arithmetic operation can be performed (a two-thirds reduction in the number of instructions).

12.2.6 Signed Clauses

This does not effect your conversion, but we recommend that you use the sign "S" for numeric fields to reduce overhead within a computational operation. The compiler has to generate extra code to force the sign to a positive value, even if the result is always positive. For example:

```
01 FIELD-NAME       PIC S9(03) COMP-3 VALUE ZERO.
```

12.2.7 Workstation Control Area

The control area definition for the display files must be changed from a System/36 Environment definition to a native definition within the WORKING STORAGE section. In System/36 Environment, the control area definition of the workstation may look like this:

```
01 WS-CONTROL-AREA.
   05 WS-CMD-KEY           PIC 9(2).
      88 ENTER-KEY         VALUE 0.
      88 CMD1              VALUE 1.
      88 CMD2              VALUE 2.
      88 CMD3              VALUE 3.
      88 CMD4              VALUE 4.
      88 CMD5              VALUE 5.
      88 CMD6              VALUE 6.
      88 CMD7              VALUE 7.
      88 CMD8              VALUE 8.
      88 CMD9              VALUE 9.
      88 CMD10             VALUE 10.
      88 CMD11             VALUE 11.
      88 CMD12             VALUE 12.
      88 ROLL-UP           VALUE 90.
      88 ROLL-DOWN         VALUE 91.
      88 CMD-ENTERED       VALUE 1 THRU 24 90 91.
   05 TERMINAL-ID          PIC X(2).
   05 FILLER               PIC X(8).
   05 CURSOR-ROW           PIC 9(3).
   05 CURSOR-COL           PIC 9(3).
```

These are the main points of the System/36 Environment workstation control area:

- WS-CMD-KEY is a 2-byte numeric field.
- The level 88 field VALUES are also defined for numeric fields.
- The TERMINAL-ID is a 2-byte character field.

- CURSOR-ROW and CURSOR-COL contain the cursor position.

In native mode, the workstation control area must look like the following:

```
01 WS-CONTROL-AREA.
   05 WS-CMD-KEY            PIC X(2).
      88 ENTER-KEY           VALUE "00".
      88 CMD1                VALUE "01".
      88 CMD2                VALUE "02".
      88 CMD3                VALUE "03".
      88 CMD4                VALUE "04".
      88 CMD5                VALUE "05".
      88 CMD6                VALUE "06".
      88 CMD7                VALUE "07".
      88 CMD8                VALUE "08".
      88 CMD9                VALUE "09".
      88 CMD10               VALUE "10".
      88 CMD11               VALUE "11".
      88 CMD12               VALUE "12".
      88 ROLL-UP             VALUE "90".
      88 ROLL-DOWN           VALUE "91".
      88 CMD-ENTERED         VALUE "01" THRU "24" "90" "91".
   05 TERMINAL-ID            PIC X(10).
   05 RECORD-FORMAT-ID       PIC X(10).
```

These are the main points of the native workstation control area:

- WS-CMD-KEY is a 10 byte-character field.
- The level 88 field VALUES are also defined for character fields.
- The TERMINAL-ID is a 10-byte character field.
- CURSOR-ROW and CURSOR-COL are not supported in the workstation control area.

12.2.8 Cursor position

To get the cursor position you have to look into the I-O-FEEDBACK AREA. CURSOR-ROW and CURSOR-COL are not supported in AS/400 native COBOL. To get the cursor position you have to incorporate the following parts into your programs.

```
*****************************************************************
Add the following statement to your SPECIAL NAMES clause.
*****************************************************- ********
       I-O-FEEDBACK IS I-O-F.
*****************************************************************
Insert the following definition into your WORKING-STORAGE SECTION.
*****************************************************************
01  I-O-F-AREA.
    05  FILLER                PIC X(147).
    05  ROW-HEX               PIC X(1).
    05  COL-HEX               PIC X(1).
01  I-O-F-WORKING-AREA.
    05  ROW-BIN               PIC 9(1) BINARY VALUE ZERO.
    05  ROW-1 REDEFINES ROW-BIN.
        10 FILLER             PIC X(1).
        10 ROW-1-2            PIC X(1).
    05  COL-BIN               PIC 9(1) BINARY VALUE ZERO.
        10 FILLER             PIC X(1).
        10 COL-1-2            PIC X(1).
01  CURSOR-ROW                PIC 9(2) VALUE ZERO.
01  CURSOR-COL                PIC 9(2) VALUE ZERO.
*****************************************************************
Insert this statements after your read screen statements
into the PROCEDURE DIVISION.
*****************************************************************
       ACCEPT I-O-F-AREA FROM I-O-F
       MOVE ROW-HEX TO ROW-1-2
       MOVE COL-HEX TO COL-1-2
       MOVE ROW-BIN TO CURSOR-ROW
       MOVE COL-BIN TO CURSOR-COL.
*****************************************************************
Now you have the row and the column in your program and
you can work with the fields CURSOR-ROW and CURSOR-COL
as you have done in the System/36 Environment
*****************************************************************
```

12.2.9 Initial Value of Fields

If you have different field-level descriptions coded in your WORKING STORAGE Section, you have to initialize these fields. Otherwise you can get decimal data errors. For example, if you have coded the following:

```
01 FIELD-GROUP.
   05 NUM-FIELD          PIC 9(4).
   05 ALP-FIELD          PIC X(4).
```

Without initialization, the NUM-FIELD contains HEX'40'. HEX'40' = BLANK. This is because the OS/400 always initializes group fields with HEX'40'. But HEX'40' is not a valid numeric character. When you move this numeric field to another numeric field, you get a decimal data error. You should either use the VALUE parameter within the WORKING STORAGE, or to use INITIALIZE at the beginning of your program.

The same problem can occur for the internally described files. If you do not move data into numeric fields, these fields will contain HEX'40' as their initial value. We recommend that you use the INITIALIZE statement in every COBOL program. You also have to use the INITIALIZE statement for your externally

described files, because there is no way to get the COBOL-VALUE function with DDS into a program.

12.2.10 CALL and CANCEL

This section is for the users who want to convert directly from System/36 to AS/400 native mode. If you used the System/36 Dynamic Call PRPQ to get around the 64K limit, you may need to make changes to avoid problems with CANCEL in COBOL/400.

The major problems arise from situations where a mix of PRPQ "CALL" and System/36 COBOL "CALL" are being used. For example:

Program-A "CALLs" Program-B through the PRPQ.
Program-B then "CALLs" Program-C.
Program-C returns to Program-B, which returns to Program-A.
Program-A now "CANCELs" Program-B through the PRPQ.

The difference is that Program-C (since it was linked into Program-B) will also be canceled by the PRPQ. When this code is converted to COBOL/400 (using native CALL and CANCEL in all cases), Program-C will remain active in its last-used state. This causes problems for programs that expect it to be in an unused state.

If the subprogram is dependent on the values in working storage being initialized at every call, you may want to consider adding additional program code to do this using explicit CANCEL.

12.2.11 COBOL MAIN Stub Prior to CL Driver

The ANSI standard states that the highest level COBOL program in the run-unit is considered the main program and any attempt to exit that program via an EXIT PROGRAM is to be considered a STOP RUN. A STOP RUN in COBOL causes the main program and all the programs called by that program in the entire process to be canceled. This causes all of the files to be closed. One way to avoid this and still use a CL driver or menu for a set of COBOL programs that you want to terminate using EXIT PROGRAM is to create a simple COBOL program that has a CALL "CL-program" statement in the PROCEDURE division. This small program is now considered the main program of the run-unit, and all COBOL programs called out of the menu driver are now considered sub-programs. These sub-programs can now all terminate using EXIT PROGRAM, and the files can remain open across menu transitions.

12.2.12 Segmentation

Using segmentation on the AS/400 adds additional overhead to the program. This feature is provided for compatibility with other COBOL compilers and is not necessary because of the AS/400 virtual storage model. Remove the SEGMENT LIMIT statement from the OBJECT COMPUTER paragraph, and the segment numbers behind your section headers.

12.2.13 CALL variable-name

This does not affect your conversion, but it is recommended that you continue using literals with a CALL statement rather than a variable name that contains the name of the program to call. The compiler has to generate extra resolve logic to do the delayed binding of the call.

12.2.14 Use of Subprograms

On System/36, the overhead for CALL and PERFORM was the same. On AS/400, this is not true. CALL has significantly more overhead than PERFORM. A general guideline to follow is to do a substantial amount of work in a called sub-program. If possible, copy small routines into your main program and use the PERFORM statement.

12.2.15 Debugging

READY TRACE, RESET TRACE, and EXHIBIT are no longer supported. Remove them. Also, remove USE FOR DEBUGGING, and take advantage of the AS/400 symbolic debug capabilities.

12.3 Externally Described Database Files

In this section we describe ways to convert your programs so that they are using either only externally described files or a combination of externally and internally described files.

The advantages of using externally described files are set out in the *COBOL/400 User's Guide*. Several changes must be made to the source code of each program to change from program-described to externally described files. These are discussed below with the aid of some sample code. (If you have been using COPY members for your file description you can make the changes easily.) To change a COBOL program to make use of DDS, the following steps are necessary:

1. Change the ASSIGN clause from DISK to DATABASE.
2. Change RECORD KEY to EXTERNALLY-DESCRIBED-KEY.
3. Use "COPY DDS" to copy in DDS. If you want ALIAS names, use "COPY DD".
4. Make sure that the DDS matches your internal description.

Keep in mind that the file must already exist during compilation.

12.3.1 Converting to External

The following sections describe the changes to be made.

For these changes, see the following examples:

- Figure 31 on page 124, which is an internal record description with a FILLER (undefined) field
- Figure 32 on page 124, which is the same description except that the FILLER is replaced by a valid definition
- Figure 33 on page 125, which shows the DDS generated by the PTK
- Figure 34 on page 126, which is the resolved external description of the record as the compiler is likely to produce it.

Clean Up Different Definitions

Consider Figure 31 on page 124 and Figure 32 on page 124. The difference is that the first example specifies a FILLER because those fields are not used in this program, and the second example defines the FILLER field now as field (CUST-NAME). If you used the PTK to create the DDS, you have already decided which of the definitions to use in the DDS. Even if the DDS is brought in by the compiler, you should fix those different internal descriptions.

Change the FILE CONTROL Paragraph

Consider Figure 34 on page 126:

- The ASSIGN clause is changed from DISK to DATABASE.
- The RECORD KEY is (optionally) changed to EXTERNALLY-DESCRIBED-KEY.

Change File Description (FD)

- Add (optionally) a new record format under level 01. See also 12.3.2, "Group Items."
- Add a format-2 COPY DDS-format name. Or if you have used COPY, you might change the COPY statement. You might choose the COPY DDS or COPY DD, depending on the program logic. COPY DDS brings the field names as mentioned in the external files description into the program. COPY DD brings in the ALIAS names. Another possibility could be using the COPY DDS with the REPLACING phrase to change the DDS field names to those used by your program.
- Make further changes, if necessary, as described in "Group Items."

12.3.2 Group Items

As you see on the compiler list in Figure 34 on page 126, we have level 05 for the external record format name and level 06 for all the elementary items. DDS on physical files cannot group fields together.

Following are some ways to solve this problem.

Additional Record Format

Create an additional record format in your program under level 01 for the same file and use COPY DDS or COPY DD to get the external description. This will occupy the same storage area as the internal description. Type this just in front of the COPY DDS statement:

```
01 MASTER-RECORD-DDS.
```

The advantage is that no further changes are required in the program. But it is the programmer's responsibility to keep the attributes of the elementary items of both the external and internal descriptions the same.

Logical File

A logical file could join elementary items into a single field by using the field-level keyword CONCAT. Care must be taken regarding the data types. Refer to the index entry CONCAT in the *Data Description Specifications Reference* manual.

Renaming

Another possibility could be renaming. If your level nesting spans only one level you might choose level 66 to group elementary items:

```
05 FIRST-NAME  PIC X(15).
05 LAST-NAME   PIC X(20).
```

You could group them by renaming:

```
66  NAME RENAMES FIRST-NAME THRU LASTNAME.
```

Redefining

You might also consider redefining by moving the internal record description to the working storage section:

1. Move the internal description to the working storage section.
2. Insert a level 01 (DUMMY) record name (as under 12.3.2, "Group Items" on page 120).
3. Insert the COPY DDS in the file section.
4. Change file access operations from READ to READ INTO and from WRITE to WRITE FROM (if the program is not already coded to do so).

Alternatively, you could place the COPY DDS into the working storage section and take similar action as described above.

12.3.3 Key Fields

Key fields are used to make up the access path to physical data. A migrated System/36 indexed file is described at the record level with one key field only. As mentioned earlier, use of the external key fields is provided by specifying the following statement in the SELECT clause:

```
RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
```

If an alternate index file with noncontinuous keys is used, the program must be changed to use EXTERNALLY-DESCRIBED-KEY, because COBOL/400 does not support multiple data names on the RECORD KEY clause.

COBOL does not support concatenated key fields. If you have concatenated key fields within your DDS, you have to change the DDS. Neither fields renamed in DDS nor fields that are part of concatenated fields can be used as keys.

Often the index in the program is made up of more than one field, and those fields may be grouped. With the PTK there will be as many keys created in the DDS as were selected at field resolution time for physical files. The physical file will have that number of keys, too. Keeping those key fields rather than specifying only one large key field (as on System/36) will allow other programs to share the access path if they use only a few of the key fields for access.

12.3.4 Record Area

Your COBOL applications might rely on the contents of the record area after certain operations which, according to the ANSI standard, leave the record area undefined. In some cases on System/36 and in the System/36 Environment, this worked, but it cannot be guaranteed with COBOL/400. Your programs should not rely on the contents of the record area after a failed READ or a successful WRITE, REWRITE, or DELETE. Take the time to understand the ANSI standard and examine your programs to be sure they follow it in this regard.

12.4 Minimal Display File Changes

The following changes should be made within your source.

12.4.1 INVITE Keyword

If the display file is used only by native AS/400 programs and it is not a multiple device display file, then the INVITE keyword may be removed. It is not needed, and might cause poor performance. If the display file is still being used by System/36 Environment programs, then the INVITE is needed, but you should still condition its use to occur only on the last write before a read to improve performance.

12.4.2 INDARA Keyword

The migration aid also generates, through the SFGR specifications, the keyword INDARA (indicator area) in the DDS. Without change this will cause a run time error stating that there is a mismatch between program and file specifications.

There are two ways to solve this:

1. Remove the INDARA keyword from the DDS and recreate the file. This might require a change in the field layout and in the program's logic, since the indicators are now part of the record area, not a separate area.
2. Move the INDARA outside the record area. Change the SELECT clause to filename-SI.

```
SELECT SCREEN   ASSIGN TO WORKSTATION-FORMATS-SI
                ORGANIZATION IS TRANSACTION
                CONTROL-AREA IS FORMATS-CONTROL.
```

12.4.3 ASSIGN Clause

Remove any "format-type" values from the ASSIGN clause. (S = Display File; C = ICF File.)

12.4.4 Changes for Externally Described Display Files

In general, the same considerations as discussed in 12.3, "Externally Described Database Files" on page 119 apply to display files.

Field Names

$SFGR does not necessarily require field names. During migration to the System/36 Environment, display files are always created out of the SFGR

specifications. If names are missing, the migration tool inserts default names like FL0001 and FL0002. As long as the file is program-described, the display record is treated only as buffer. This is also true for externally described display files, but when using the format-2 COPY DDS you might get unexpected field names in your program.

This can be resolved by doing one of the following:

- Adding record format
- Renaming
- Redefining

as described under 12.3.2, "Group Items" on page 120.

In order to be consistent with all field names (any piece of information should exist only once in the system) it is recommended that you change the display file to use the database names or ALIAS names.

$SFGR also accepts certain special characters (like the dollar sign). Using the format-2 COPY DDS brings in those names causing the compiler to fail and give an "invalid character" message (LBL1041).

12.5 COBOL Examples

```
       FILE-CONTROL.
           SELECT MASTER ASSIGN TO DISK-MASTER.
                 RECORD KEY IS CUST-NUMBER.
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER
                 LABEL RECORD ARE STANDARD.
       01  MASTER-RECORD.
           03 CUST-NUMBER.
              05 CUST-ID.
                 07 CUST-ID1       PIC X(4).
                 07 CUST-ID2       PIC 9(4).
----->     03 FILLER               PIC X(12).
----->     03 CUST-AREA-ZIP        PIC 9(5).
           03 MASTER-CODE.
              05 MASTER-CODE-ID    PIC 9(2).
              05 MASTER-CODE-REF   PIC 9(2).
           03 MASTER-AMOUNT        PIC S9(5)V99.
```

Figure 31. COBOL - Partly Program-Described File

Two fields are not properly defined: the FILLER and CUST-AREA-ZIP, which should be a group item (see next example). The same structure could be coded in the working storage section.

```
       FILE-CONTROL.
           SELECT MASTER ASSIGN TO DISK-MASTER.
                 RECORD KEY IS CUST-NUMBER.
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER
                 RECORD CONTAINS 32 CHARACTERS
                 LABEL RECORD ARE STANDARD.
       01  MASTER-RECORD.
           03 CUST-NUMBER.
              05 CUST-ID.
                 07 CUST-ID1       PIC X(4).
                 07 CUST-ID2       PIC 9(4).
           03 CUST-NAME.
--->          05 CUST-AREA.
--->             07 CUST-AREA-STATE PIC X(12).
--->             07 CUST-AREA-ZIP   PIC 9(5).
           03 MASTER-CODE.
              05 MASTER-CODE-ID    PIC 9(2).
              05 MASTER-CODE-REF   PIC 9(2).
           03 MASTER-AMOUNT        PIC S9(5)V99.
```

Figure 32. COBOL - Fully Program-Described File

The FILLER field in Figure 31 has been replaced by a proper description.

```
A          R MASTERFMT
A            CUSTID1        4          ALIAS(CUST_ID1)
A            CUSTID2        4S         ALIAS(CUST_ID2)
A            CUSTAREAST    12          ALIAS(CUST_AREA_STATE)
A            CUSTAREAZI     5  0       ALIAS(CUST_AREA_ZIP)
A            MASTERCODI     2  0       ALIAS(MASTER_CODE_ID)
A            MASTERCODR     2  0       ALIAS(MASTER_CODE_REF)
A            MASTERAMOU     5  2       ALIAS(MASTER_AMOUNT)
A          K CUSTID1
A          K CUSTID2
```

Figure 33. COBOL - DDS for File MASTER

The DDS is created by the PTK. Refer to the sample programs in Figure 31 on page 124 and Figure 32 on page 124. The ALIASes are inserted automatically by the PTK. The "_" will be changed to "-" when the fields are used in a COBOL program.

```
       FILE-CONTROL.
      *    SELECT MASTER ASSIGN TO DISK-MASTER.
EXT-->     SELECT MASTER ASSIGN TO DATABASE-MASTER.
      *          RECORD KEY IS CUST-NUMBER.
EXT-->           RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER
                 RECORD CONTAINS 32 CHARACTERS
                 LABEL RECORD ARE STANDARD.
EXT--> 01  MASTER-RECORD-DDS.
EXT-->     COPY DDS-MASTERFMT  OF MASTER.
```

The following was generated through the COPY DDS statement:

```
*   I-O FORMAT:MASTERFMT FROM FILE MASTER  OF LIBRARY SLRES17

*  THE KEY DEFINITIONS FOR RECORD FORMAT  MASTERRECO
*  NUMBER          NAME         RETRIEVAL     TYPE    A
*   0001   CUSTID1                  ASCENDING   SIGNED
*   0002   CUSTID2                  ASCENDING   SIGNED
            05  MASTERFMT.
                06 CUSTID1           PIC X(4).
                06 CUSTID2           PIC S9(4).
                06 CUSTAREAST        PIC X(12).
                06 CUSTAREAZI        PIC S9(5)        COMP-3.
                06 MASTERCODI        PIC S9(2)        COMP-3
                06 MASTERCODR        PIC S9(2)        COMP-3.
                06 MASTERAMOU        PIC S9(3)V9(2)   COMP-3.
```

The following is the old internal description:

```
       01  MASTER-RECORD.
           03 CUST-NUMBER.
              05 CUST-ID.
                 07 CUST-ID1       PIC X(4).
                 07 CUST-ID2       PIC X(4).
           03 CUST-NAME.
              05 CUST-AREA.
                 07 CUST-AREA-STATE PIC X(12).
                 07 CUST-AREA-ZIP   PIC 9(5).
           03 MASTER-CODE.
              05 MASTER-CODE-ID    PIC 9(2).
              05 MASTER-CODE-REF   PIC 9(2).
           03 MASTER-AMOUNT        PIC S9(5)V99.
```

Figure 34. COBOL - Externally Described File

12.5.1 Additional COBOL Considerations

Here are additional differences between System/36 compatible COBOL programs and COBOL/400 programs:

- COBOL/400 programs cannot access fields defined in the File Section unless the file in question is open.
- Files used by COBOL/400 programs must exist prior to being opened. There is no equivalent to the // FILE statement's DISP-NEW for output files.
- The Dynamic Call PRPQ feature is not supported.

- System/36 COBOL allowed some variations in syntax. These are not supported under COBOL/400. For example:

```
      IF A-VALUE IS EQUAL TO TO TO B-VALUE. (repeated TO)
or MOVE A-VALUE B-VALUE.                    (missing TO)
```

- ANSI differences: There are a number of differences between ANSI 74 (the standard for the System/36 compiler) and ANSI 85 (used by the COBOL/400 compiler). These differences are documented in the *American National Standard Programming Language COBOL* (ANSI X3.23 - 1985) manual under "Differences Potentially Affecting". Appendix I in the *System/38 Compatible COBOL User's Guide and Reference* outlines the differences.
- ANSI also defined a number of additional File Statuses and clarified the use of others. This may affect users who monitor for specific file status values, or who rely on the program following an AT END or INVALID KEY path repetitively.
- MRTs: On System/36, separate copies of the LDA and the UPSI switches are maintained for each terminal. On AS/400 (COBOL/400), there is a single copy that "belongs" to the job.
- The mixing of ICF and local workstation support (mixed device files in System/38 terms) in a single transaction file is not supported.
- The run-time routines such as CBFTOD and CBSTOP are specific to the System/36 environment and are not supported by COBOL/400.
- COBOL/400 does not support indicator data structures that contain nonindicator data items. For example:

```
01 W-INDICATORS.
   05 W-I-TAB
      10 W-I-ENTRY          PIC X OCCURS 20.
   05 W-I-IND REDEFINES W-I-TAB
                            PIC 1 OCCURS 20 INDICATOR 1.
```

- In general the System/36 COBOL compiler does not always enforce ANSI standards. The COBOL/400 compiler, on the other hand, is far stricter in its compliance. There might be other techniques the System/36 compiler supported that do not work with COBOL/400.
- The use of USRDSPMGT in a display file might affect the use of other keywords or functions.
- Values for Extended File Status Key for displays have changed.

13.0 Additional Program and Utility Considerations

Most of the changes to programs needed in order to change to external files are discussed in other chapters. The topics here are general ones related more to program-to-program communication and program initiation. They are not in any particular order.

13.1 Program Communication and Program Structure

13.1.1 Calling One Program from Another

Most System/36 programs do not call other programs directly. A program normally ends and calls another program from a procedure. To return to the first program, it is necessary to start processing at the top of that program. If you need to start at a point in the middle of the program, it could be difficult.

With System/36 the calling program usually places the name of the next program in the LDA, where it is picked up by the OCL and substituted in a // LOAD statement.

This problem is easily solved on AS/400. CL and HLL programs can use the CALL statement to transfer between programs. The called program can then return to the next executable statement in the calling program.

If you want to process a single CL command from within your program, you can do so by calling the QCMDEXC program.

You might wish to simplify your System/36 programs by using these new AS/400 options.

Refer to the programmer's guides and reference manuals to get more information about calling other programs.

13.1.2 Passing Data to Another Program

This procedure is difficult on System/36. It can be done, for example, by means of a file. The first program can write a record. The second program can test the file after waiting for a few seconds, and then read any available records.

With AS/400 you can use data queues to transfer data. Data queue transfer is efficient. The receiving program waits for the data and processes it as it arrives. You do not have to leave the program to put data in a queue.

You can read about data queues in the *Control Language Programmer's Guide*.

13.1.3 Evoking a Program

An example here would be an order entry program that prints invoices. Instead of printing the invoice in the order entry program itself (which would increase the response time at the terminal), the user can exit the RPG program and start a separate print program as each order is entered. The users can then attach to the order program again. On AS/400 the same effect can be gained by using the

data queues described above. In fact, with data queues, more than one copy of the print program could be running at the same time.

Or, using the QCMDEXC command, you can submit an independent batch job without leaving your current program.

13.1.4 Using the Attention Key

On System/36, pressing the Attention key suspends the current program and displays a menu with a number of options, including the ability to start another session and run another procedure. On AS/400 (both System/36 Environment and native), equivalent support is provided through the Sys Req key and it's corresponding menu. There is additional support available for the Attention key.

Read the *Work Management Guide* and the *System Operations: Operator's Guide* for a more detailed explanation.

Some of the available options are:

- Use an alternate session. You can press the Sys Req key and transfer to an alternate job. The first job is suspended while the second is running. You can also remove access to the Sys Req key functions by revoking authority to *PNLGRP QGMNSYSR in library QSYS.
- Use the Attention key. Each user profile can have an Attention key program specified. When the Attention key is pressed, the current job is suspended and the user can be shown another menu or screen.
- Make the current job one of a group. With group jobs, up to 32 programs can operate from one device (with a maximum of 16 each for a job and the alternate job), although only one can be active at a time.

13.2 Multiple Requester Terminal Programs

On System/36, a single copy of a program in memory can serve multiple users by means of the MRT attribute. One copy of the executable statements is shared by all attached users. The users queue for entry to the code. Only one user is serviced at a time. Other users wait their turn until the current user enters a long wait (generally a workstation operation).

System/36 MRTs are supported for both display devices and ICF devices in the System/36 Environment.

Native AS/400 does not provide support for MRTs, and System/36 Environment MRT programs must be converted to run in native AS/400.

13.2.1 MRT Considerations in the System/36 Environment

There is no need to rewrite System/36 MRTs in order to run them in the System/36 Environment. Functional equivalence is provided. There are some performance considerations that require a good understanding of your specific MRT application.

- MRTs that are run frequently should be run as NEPs. There are performance improvements to be gained by changing frequently run non-NEP MRTs to NEP MRTs.

- There is usually not a significant performance payback from converting MRTs to native AS/400 programs. In some cases, such a conversion will make performance worse.

13.2.2 MRT Considerations for Native AS/400

There is no native AS/400 support for MRTs, nor is there any native equivalent to support processing of multiple requester devices from a single program. However:

- AS/400 provides support for a single program accepting input from multiple devices through the use of multiple device files, but this support is only applicable to devices acquired by the program. There is no AS/400 support to connect multiple requesters (interactive users) to a single program. RPG/400 and COBOL/400 support multiple device files. See the *RPG/400 User's Guide* and the *COBOL/400 User's Guide* for more information.
- AS/400 provides support for a batch job handling requests from multiple interactive jobs through the data queue support described in *Control Language Programmer's Guide*. However, this support is intended for batch-type work and does not support the use of requester devices in the batch job.

13.2.3 MRT Programs and Shared Files

There are certain situations where logic changes are needed to move away from multiple requester programs. These involve database files that are used by more than one job at the same time.

Suppose a System/36 MRT program has the following logic:

1. Read control record to get the starting relative record number for output.
2. Release control record.
3. Write records starting at the relative record retrieved in step 1.
4. Read control record and update with a fresh starting number.
5. Exit the cycle.

This works fine on System/36 and in the System/36 Environment, since the MRT synchronizes the use of the file. However, if the same logic is placed in an AS/400 program, and that program is concurrently run by multiple users, two users might retrieve the same starting relative record number, with undesirable results. (The same problem would occur on S/36 if the users each had an SRT program.)

To prevent this, change the program logic to lock or allocate the correct records to users until they are finished with them.

There are other changes that might be needed to handle cases related to the use of common output files. The cases below give a few possibilities:

- Users of a program might be sharing a common output file. The program writes each user's workstation ID (from the KID field, for RPG) to the output records so that later programs can tell who created the record.

  Consider retrieving the job attributes and passing (for example) the user ID from the calling CL program to the RPG program. The ID field in the output

record may need to be enlarged, since the System/36 work station ID is only two characters long.

- A program might write to a different file, depending on which display device the program was run from, using an OCL statement like:

```
// FILE NAME-TRANS,LABEL-TRAN?WS?.
```

  In this case, an OVRDBF statement could be used to place the output in a member of a file. The member name could be built by the CL program from the file name and a retrieved job attribute (like user ID). Later programs could process either individual members of the file, or the whole file, by using OVRDBF statements.

- Users of a program might share a file, but each might use a different part of the file. User 1 may use record 1-1000. User 2 may use records 1001-2000, and so on. This can be changed to a multiple member file, as in the case above.

13.3 Never-Ending Programs

For batch programs you can make a program never-ending by coding it to wait for an entry in a data queue.

For interactive programs there is no native AS/400 support to attach an interactive user to a program that is already running. An interactive program could be written as never-ending by always waiting for an additional request from its specific requester device.

13.4 Sort Programs

13.4.1 Sort and Format Data

The discussion, "Restructuring For Better Performance", on the use of sort or logical files is pertinent here. Use those guidelines to make a choice of method between the two.

The sort specifications you used on System/36 can also be used on AS/400.

Use the FMTDTA command to name the file to be sorted, the file that will contain the output records, and the source member containing the SORT specifications. **The output file must exist before executing the FMTDTA command;** the sort will not create an output file when using FMTDTA.

ADDROUT files differ between native AS/400 and the System/36 Environment. Both the System/36 and System/36 Environment SORT create ADDROUT files with a record length of 3 and entries starting from 0. The AS/400 SORT creates ADDROUT files with a record length of 4 and entries starting from 1, and it uses -1 to mark the last entry in an ADDROUT file. You must ensure that you use the correct RPG product when using SORT (RPG II for System/36-compatible SORT; RPG/400 for native AS/400 SORT), so that ADDROUT file processing works correctly. Also, if you write your own ADDROUT file processing routines in your applications, then these routines must be rewritten when converting from System/36 Environment SORT to native AS/400 SORT.

13.4.2 The #GSORT Utility

PTK converts the #GSORT utility to the FMTDTA command correctly, and creates in the QFMTSRC source file a member that contains the inline SORT specifications. The name of this member is the original System/36 procedure name plus a serial number starting at '01' and incrementing depending on the number and sequence of the #GSORT utilities within the procedure.

The following example shows conversion of the #GSORT utility.

```
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-CUSTMSTR
// FILE NAME-OUTPUT,LABEL-CUSTREPT,RECORDS-?F'A,CUSTMSTR'?
// RUN
      HSORTR    11A         X
      I C   1   2EQCCM                ALL CUSTOMER RECORDS
      FNC  22  23                     BRANCH
      FNC   6  11                     CUSTOMER NUMBER
      FDC   1 256                     CUSTOMER RECORD
// END

Converts to...
    CRTPF     FILE(MYLIB/CUSTREPT)
    FMTDTA    INFILE(*LIBL/CUSTMSTR) OUTFILE(MYLIB/CUSTREPT)  +
              SRCFILE(*LIBL/QFMTSRC) SRCMBR(procedurenn)  +
              OPTION(*NOCHK *NOPRT)
Where...
    'procedurenn' is a source member in the QFMTSRC file that contains
    the SORT control statements (between // RUN and // END).

NOTE: Certain keywords in the // FILE statement are ignored
      in this case RECORDS-?F'A,CUSTMSTR'? which is not required
      for AS/400. You will need to check any converted output to
      ensure that the original intentions of the sort are performed.
```

Figure 35. Example of Converting #GSORT to FMTDTA

13.4.3 Sort and Logical Files

The logical files used in your application programs may already maintain the data in a sequence required by one of your sort programs. In this case you can read the logical file into the application program without having to run the sort and without having to create the logical file.

If there is no logical file that maintains the data in the sequence you want, you can create a logical file in place of the sort. This might be the case for an end-of-year report, where the particular sequence is only needed once a year.

13.5 DFU Programs

DFUs have the same or equivalent functions in the AS/400 native environment, so they can be converted from one environment to the other.

PTK converts System/36 DFU procedures to CL as follows:

```
      S/36 DFU          AS/400 CL

    ENTER/UPDATE          CHGDTA
      INQUIRY             DSPDTA
        LIST              RUNQRY

Shown in the following examples

ENTER CUSTMSTR,CMENTER,,2000,,,,,#LIBRARY
                          CRTPF  FILE(MYLIB/CUSTMSTR)  +
                                 SRCFILE(MYLIB/QDDSSRC)
                          CHGDTA DFUPGM(#LIBRARY/CMENTER) +
                                 FILE(MYLIB/CUSTMSTR)
LIST CUSTMSTR,CMLIST,,NOSORT,,,,,MYQRYLIB
                          RUNQRY QRY(MYQRYLIB/CMLIST) +
                                 QRYFILE(*SAME) OUTTYPE(*PRINTER)
INQUIRY CUSTMSTR,CMINQRY,,,,,,,MYLIB,,CUSTMAST
                          DSPDTA DFUPGM(MYLIB/CMINQRY) +
                                 FILE(*LIBL/CUSTMAST)
```

Figure 36. Example of Converting System/36 DFUs

The ENTER DFU requires that the database file is created before any data can be entered.

Your System/36 DFUs are not converted and must be recreated in AS/400 native using STRDFU or STRQRY as appropriate. During the database conversion you will have resolved fields, records, and file names, which will have made recreating the DFUs necessary anyway.

In the examples above you must create the physical (or logical) files CUSTMSTR and CUSTMAST first, using PTK or your own definition. Then you must create the AS/400 DFU programs CMENTER and CMINQRY and the Query CMLIST in the appropriate libraries based on the original System/36 DFUs.

14.0 Finding Programs that Cause Decimal Data Errors

Finding the program that causes a decimal data error can be difficult. Every program adding records to or updating a file must be checked, and the programs can be either System/36 Environment or native programs.

The PTK will help you find problem programs that are updating single record format files. Use option 4 (Analyze Changed Data) on the PTK Main Menu.

There are three inputs required by the PTK:

- The single format file. This could be a single format, program-described System/36 file in QS36F, or physical file with external DDS (for example, a physical file resulting from the split of a multiple format).
- An AS/400 file that contains the record format description for the single format file above.
- A journal containing a record of the updates to the single format file. As each program updates or adds to the file, the journal contains the before- and after-images of the records and the name of the program. You have to set up the journal and journal receiver and start journaling on the file before you run the PTK. Remember to specify *BOTH to capture both before- and after-images in the journal.

The PTK uses the external description, the file itself, and the journal entries to detect offending programs. It writes a report showing which programs are causing the decimal data errors.

Again, the PTK cannot, at the time of writing, handle multiple format files or logical files. The PTK would run to completion with no error messages, but would not distinguish between the different record types, even with a format selector program. The PTK would behave as though all fields were present in all formats; a format with a numeric field in a particular position would be reported in error if another format had a numeric field in the same position. This is why a multiple format file must be split into distinct physical files, which are then examined separately. Journaling can be started on the separate physical files.

15.0 Converting from OCL to CL

This chapter discusses ways in which CL and OCL can be used on AS/400. Different approaches and methods are available for conversion.

15.1 Different Approaches

Three broad approaches are possible:

- Convert only the minimum needed to support other parts of the application that are being converted. Some changes may be required to handle different file structures. Some changes might be desirable for performance reasons. But there is no reason why System/36 Environment OCL cannot operate on AS/400 native files and programs.
- Move all OCL to its CL equivalent. Most OCL statements can be directly replaced by one or more CL statements which produce the same result. Most OCL procedures can be replaced by CL statements, again with a similar effect. This tends to be a one-for-one approach. One line of OCL is taken and replaced by corresponding CL, then the next line is taken, and so on until all procedures have been converted. Little thought may be given to the overall intent of the procedure. Although some simplifications may be made, no new AS/400 functions will be added.
- Understand the intent of the old OCL procedures. Look for simpler or better ways to implement the same intent with CL. Be prepared to add new program logic to simplify inter-program communication and reduce program initiation. Incorporate new file structures. Use new recovery, restart, and save-restore functions. Use new messaging and queueing functions. In other words, rewrite the procedures to take full advantage of the AS/400 environment. This might be particularly desirable if the procedures were originally written for S/34 or earlier equipment and contain statements that are allowed with S/36 SSP but not on AS/400.

15.2 Tools Available

If you intend to rewrite procedures, there is no substitute for a knowledge of OCL, the application to be converted, and CL.

PTK helps you convert System/36 OCL procedures to AS/400 CL source members and compiled CL programs through the following steps:

- Listing unsupported OCL statements
- Converting an individual OCL procedure with optional CL compilation, keeping the original OCL statements as comments
- Preparing an OCL procedures conversion list with options to work with and print the list
- Converting the conversion list with optional CL compilation
- Analyzing and listing suggested performance enhancements in the generated CL source
- Removing commented CL statements.

Use of this tool does require a working knowledge of the System/36 application being converted and may require changes to produce the same or required results depending on the output of the conversion process and the errors and messages produced.

We suggest that when using PTK to convert the OCL that you remove the System/36 OCL from the new CL programs once they have been tested. The comments might unnecessarily confuse the viewing and understanding of the CL program.

The Programmer Tools PRPQ provides a *User's Guide* document you can print by selecting option 10 from the PTK Main Menu. Similar information is displayed when you use the HELP facility.

15.3 Summary of Later Topics

The bulk of the information about OCL and CL is contained in the tables in the appendixes. These tables give the exact or nearest CL equivalent for System/36 OCL and procedural statements. They are:

- Appendix A, "UCS/Procedure Relation Table."
- Appendix B, "Procedure/CL Relation Table."
- Appendix C, "OCL/CL Relation Table."
- Appendix D, "OCC and CL Command Table."
- Appendix E, "Table of System/36 Substitution Expressions."
- Appendix F, "If Conditions and Their Equivalents."
- Appendix G, "Procedure Control Statements and Their Equivalents."

The remainder of this chapter is used to discuss some of the areas where extra guidance is needed. This might be because the PTK does not convert to an equivalent CL statement, or because several options are available, or because some extra code is needed to tidy up the PTK output. For example, you should always check the generated CL statements to ensure correct device names are used.

15.4 System/34 OCL Considerations

Many System/36 application systems have themselves been migrated from System/34 or even earlier systems and can contain OCL and system procedure statements that are not standard System/36 statements but are allowed for System/34 compatibility. Such systems will also make extensive use of utility statements such as $COPY, $DELET, $GSORT or #GSORT, and $MAINT with inline sort specifications and the ORGANIZE system procedure.

While these will migrate to the System/36 Environment, we suggest that these statements be edited on the System/36 before migrating to the AS/400.

For example, ORGANIZE is replaced with COPYDATA; $COPY with COPYDATA, SAVE, RESTORE, etc.; $DELET with DELETE, and so on. These changes make conversion to AS/400 CL much easier particularly if you use PTK.

See Appendix A, "UCS/Procedure Relation Table" on page 167 for more information.

15.5 Cleaning Up Your Procedures

Over time, many of your procedures will have accumulated redundant statements, usually by turning OCL lines into comments or having some OCL or system procedure statements that are not required with the AS/400 operating system. Statements such as '// REGION', KEYSORT, and '*/' (for an OCL statement made into a comment) should be removed before conversion.

You should review all comments and operator messages in your System/36 procedures before conversion to make sure that these are still valid within the application. Where the operator is asked for answers or other data through a series of operator messages, you should consider creating a display file for those answers that is input to the converted CL program.

Also remove any procedure sequence numbers, particularly if you are using PTK, because the conversion process in some cases will use sequence numbers as part of OCL statements being converted. This will create CL statements that will not give the expected result.

15.6 Operation Control Language Statements

This section discusses individual OCL statements and some of their parameters.

Where possible, suggestions are given for ways the parameters might be handled on AS/400. If you are using PTK for conversion, check the resulting CL source programs for those statements that do not convert in part or in whole, or to simplify the generated equivalents of the original OCL. This section should assist you in this process.

ALLOCATE

The ALLOCATE statement is converted to the ALCOBJ CL command. The following parameters are not supported:

- AUTO - AS/400 expects to process a single object on one device. You will have to change your procedures to conform.
- CONTINUE - no equivalent is suggested.
- WAIT - For ALCOBJ, specify the number of seconds to wait.

ATTR

The ATTR statement is converted to the CHGJOB CL command. The following ATTR parameters are not supported:

- CANCEL - On AS/400, the Sys Req key support is equivalent to the System/36 Attention key support. To disable the cancel option on the Sys Req menu, revoke the user's authority to the ENDRQS CL command.
- INQUIRY - On AS/400, the Sys Req key support is equivalent to the System/36 Attention key support. To disable the inquiry option on the Sys Req menu, revoke the user's authority to the TFRSECJOB CL command.

- MRTMAX - MRT programs should be replaced by single requester programs on AS/400.
- NEP - There is no AS/400 equivalent.
- RELEASE - This parameter is like an EVOKE statement. The closest AS/400 equivalent is the SBMJOB CL command. Some of the differences are described in the EVOKE statement discussion later in this section.
- MRTWAIT - There is no AS/400 equivalent.
- NOTIFY - There is no AS/400 equivalent.

Run priorities on AS/400 are different from those on System/36. On AS/400, priority 1 is the highest and priority 99 is the lowest. The initial default AS/400 priorities are 20 for interactive jobs and 50 for batch jobs. Check the value chosen by PTK and adjust it if you need a different value.

CANCEL

- The CANCEL statement is converted to the WRKSPLF command.
- Alternatively, to remove one or more spool file entries, use the DLTSPLF command. Keywords are available to select user, print device, and form type.

```
// CANCEL PRT,P7        DLTSPLF FILE(*SELECT) SELECT(*CURRENT P7)

// CANCEL PRT,ALL       DLTSPLF FILE(*SELECT) SELECT(*ALL *ALL)
```

CHANGE

- The CHANGE statement is converted to the WRKSPLF command.
- An alternative, you can use the CHGSPLFA or OVRPRTF command.

```
// CHANGE COPIES,5          CHGSPLFA FILE(*SELECT)
                                     SELECT(*CURRENT *ALL)
                                     COPIES(5)
```

DATE

- The // DATE statement is converted to a CHGJOB command. A warning message is issued. The date format used in the CHGJOB command must be the same as the date format of the job in which it is running. The system default for a job's date format is defined by the system value QDATFMT. It may be changed for an individual job using the CHGJOB command.

```
// DATE 021188              CHGJOB JOB(*) DATE('021188')
```

DEALLOC

- The DEALLOC statement is converted to the DLCOBJ command with the same keywords used in the corresponding ALCOBJ.

```
// DEALLOC UNIT-T1          DLCOBJ OBJ((*LIBL/QTAPE1 *DEVD
                                    *EXCLRD))
```

EVOKE

- The // EVOKE statement is converted to a SBMJOB command. The submitted job contains a CALL of the evoked procedure. Be aware that the

submitted job might not run at once but might have to wait in the queue for other jobs to finish.

See 13.1.3, "Evoking a Program" on page 129 for an alternative method that avoids this problem.

You must check that the job description used to submit the job contains those libraries needed to run the job.

Make sure that the submitted job is placed on the correct job queue in your system (the usual default is QBATCH in QGPL).

In addition, there are often many blank concatenated keywords on the converted statement that can be removed.

```
// EVOKE PROC55,              SBMJOB RQSDTA('CALL TESTLIB/PROC55
         TESTLIB,PARM1                PARM('PARM1')')
```

FILE

The FILE statement is converted by PTK. The parameters previously specified on the // FILE statement are specified either in the file DDS, at file creation time, or on an override CL command. There are a number of redundant // FILE statement parameters (such as EXTEND) that do not convert when using PTK. These are listed below:

- RECORDS and BLOCKS - Use the SIZE keyword when creating the file. Make sure that you also include any required extensions.
- LOCATION - You can request contiguous storage and a preferred access arm with the CONTIG and UNIT keywords of the CRTPF command. However, this is unnecessary and is not recommended. The system will tell you if the request cannot be met.
- JOB - Use ALCOBJ and DLCOBJ to control allocation and deallocation of a file.
- WAIT - Use the WAIT keyword on the ALCOBJ CL command. You could also use WAITFILE keyword on the CRTPF command.
- EXTEND - Automatic extension is performed by AS/400 depending on values in the SIZE keyword. Warnings can be issued at points specified on the SIZE keyword of the CRTPF command. On AS/400, the extendibility is determined at file creation time (the default is extendable) and cannot be overridden after the file is created.
- AUTO - There is no equivalent.
- BYPASS - Not used.
- DENSITY - There is no equivalent.
- IBLOCK - There is no equivalent.
- DUPKEYS - This is specified in the file DDS. The UNIQUE DDS keyword indicates that duplicate keys are not allowed.
- DBLOCK - See the SEQONLY keyword on the OVRDBF command. RPG automatically handles record blocking for sequential files.
- STORINDX - Not supported.

FILE OCL is converted to a number of CL commands by PTK conversion, depending on the FILE statement and whether the conversion can determine if this is an output file. This is usually determined by finding a RECORDS or BLOCKS parameter.

```
// FILE NAME-FILE1,          OVRDBF FILE(FILE1) TOFILE(VENDOR)
       LABEL-VENDOR,               LVLCHK(*NO) SECURE(*YES)
       DISP-SHR,           ALCOBJ OBJ(*LIBL/VENDOR *FILE *SHRUPD
       EXTEND-1000                     *FIRST)

// FILE NAME-FILE2,          CRTPF  FILE(QS36F/FILE2)
       LABEL-OUTPUTF,              SRCFILE(*LIBL/QDDSSRC)
       RECORDS-10000,              SRCMBR(*FILE) OPTION(*NOSRC
       RETAIN-T                    *NOLIST) ALWDLT(*NO) LVLCHK(*NO)
                           OVRDBF FILE(FILE2) TOFILE(OUTPUTF)
                                  LVLCHK(*NO) SECURE(*YES)
                           ALCOBJ OBJ(*LIBL/OUTPUTF *FILE *EXCL
                                      *FIRST)

// FILE NAME-FILE1,UNIT-I1   OVRDKTF FILE(FILE1)
       LABEL-MASTER                TOFILE(*LIBL/MASTER)
                                   VOL(*MOUNTED)

// FILE NAME-FILE1,UNIT-T1   OVRTAPF FILE(FILE1) SEQNBR(1)
                                   EXDATE(*PERM) VOL(*MOUNTED)
```

While these statements replicate the System/36 operating system file handling you may want to simplify them.

All files in AS/400 remain on disk until deleted, unless they are created in the QTEMP library. Put job files (J and S files) in QTEMP, and they will be deleted with QTEMP at the end of the job. Create the QTEMP files at the start of a job to reduce later initiation time, or keep a master set of files in your library and put CRTDUPOBJ into QTEMP.

For RETAIN-S files, such as in the example

```
// FILE NAME-COPYIN,LABEL-XXXXXX,RETAIN-S
```

you will have to include a DLTF command in the appropriate step in the CL program to effect the intended deletion.

The RECORDS and EXTEND parameters are NOT converted. The following examples show how these parameters can be handled:

```
An output file is created
   // FILE NAME-FILE1,          CRTPF FILE(*LIBL/FILE1)
          LABEL-MASTER,               SRCFILE(*LIBL/QDDSSRC)
          RECORDS-800,DISP-SHR,       SRCMBR(*FILE) OPTION(*NOSRC
          WAIT-YES,EXTEND-80,         *NOLIST) ALWDLT(*YES) LVLCHK(*NO)
          DFILE-YES,DUPKEYS-YES       SIZE(800 80 5) WAITFILE(*CLS)
                                 OVRDBF FILE(*LIBL/FILE1)
                                        TOFILE(*LIBL/MASTER)
                                 ALCOBJ OBJ(*LIBL/MASTER *FILE *SHRUPD
                                            *FIRST)
    NOTE that the RECORDS and EXTEND parameter values are in the
         SIZE parameter (the 5 extensions has been arbitrarily chosen).

   // FILE NAME-COPYIN,         This will be part of the CPYF command
          LABEL-XXXXXX,          CPYF FROMFILE(*LIBL/XXXXXX) --etc.--
          RETAIN-S               then add
                                 DLTF FILE(*LIBL/XXXXXX)
                                 for RETAIN-S
```

FORMS

- The OVRPRTF command has all the parameters of the // FORMS statement. There are also some additional useful parameters. Read the OVRPRTF command to find out about folding, fonts, justification, and other options.

```
// FORMS DEVICE-P7,LINES-55,   OVRPRTF FILE(*PRTF) DEV(P7)
         FORMSNO-CHECK1,CPI-15,        PAGESIZE(55) LPI(6)
         LINES-6,ROTATE-90,            CPI(15) DRAWER(2)
         DRAWER-2                      PAGRTT(90) FORMTYPE(CHECK)
```

INCLUDE

- The INCLUDE command is converted to the CALL command with corresponding parameters.

```
// INCLUDE PROC1 INVNUM        CALL PGM(MYLIB/PROC1) PARM('INVNUM')
```

INFOMSG

- The // INFOMSG statement is not converted. Change to the CHGMSG command.

```
// INFOMSG YES              CHGMSGQ MSGQ(*WRKSTN) DLVRY(*BREAK)
                                    SEV(00)
```

JOBQ

- The JOBQ OCL statement is converted to the submit job (SBMJOB) command.

  You must check that the job description used to submit the job contains those libraries needed to run the job. The submitted job is placed on the job queue QBATCH. This might have to be changed. Also, there are often many blank concatenated parameters on the converted statement that can be removed.

The priority parameters in System/36 are from 0 to 5, where 5 is the highest. In the AS/400 the values are 1 through 9, where 1 is the highest priority.

```
// JOBQ 4,PAYLIB,        SBMJOB JOB(PAY) JOBD(*USRPRF)
   PAYROLL,PARM1                USER(*CURRENT) JOBQ(*JOBD)
                                JOBPTY(2)
                                CMD(CALL PAYLIB/PAYROLL
                                    ('PARM1'))
```

LIBRARY

- The LIBRARY OCL statement is converted to the CHGCURLIB command. The current library remains in the list until you sign off or use library list commands with different values. Any permanent change will have to be done using the Change User Profile (CHGUSRPRF) command, which changes the CURLIB keyword, or by using the Change Job Description (CHGJOBD) command, which changes the INLIBL keyword.

```
// LIBRARY NAME-PAYLIB,       CHGCURLIB CURLIB(PAYLIB)
           SESSION-YES
```

LOAD

On AS/400, the CALL CL command provides the equivalent of the // LOAD and // RUN pair of System/36 OCL statements.

LOCAL

The LOCAL OCL statement is converted to the Change Data Area (CHGDTAARA) command. In some cases where parameter substitution is used to determine the contents of the LDA, the length parameter is not properly handled.

- BLANK - Use a blank character string.
- AREA - AS/400 supports as many data areas as you need. Two data areas, called *LDA and *GDA, are already available for use.

```
// LOCAL OFFSET-1,BLANK-8     CHGDTAARA DTAARA(*LDA (1 8)
                                        VALUE(' '))
```

LOG

- The LOG statement is converted to change job (CHGJOB) command.

  In the System/36 the LOG OCL indicates whether the OCL statements in a procedure are logged to the history file, regardless of the OCL logging indicator in the procedure.

  The AS/400 equivalent is to control logging to the job log of a specific job. The logging of CL commands run from a CL program is controlled by the LOGCLPGM keyword of the CHGJOB command, in conjunction with the value of the LOG keyword specified when the CL program was created (CRTCLPGM command). The logging of CL commands entered interactively is controlled by the LOG keyword of the CHGJOB command.

```
// LOG ON                     CHGJOB JOB(*) LOGCLPGM(*YES)
```

MEMBER

The MEMBER statement converts to the OVRMSGF command. The usages are not equivalent. LIBRARY, PROGRAM1, PROGRAM2, and USER2 are not supported. Here is additional information about creating and sending AS/400 messages.

- Create an empty message file with CRTMSGF, then add your message descriptions with ADDMSGD. If your System/36 application uses a second-level member, then you use the SECLVL keyword in the ADDMSGD command, because there is no separate second-level member on AS/400.

  The messages are sent with SNDUSRMSG.

- When you change your message member within a procedure you can do either of the following:
  - Explicitly specify the name of the message file in the SNDxxxMSG command.
  - Use the OVRMSGF command in the CL program.

```
                                    CRTMSGF MSGF(MYLIB/PAYMSGF)

                                    ADDMSGD MSGID(PAY0010)
                                            MSGF(MYLIB/PAYMSGF)
                                            MSG('PAYROLL EXECUTING')
                                            SECLVL('WEEKLY PROCESS')
                                            SEV(00)

// MEMBER USER1-PAYMSGF,
          USER2-SLVMSGF,
          LIBRARY-MYLIB             SNDUSRMSG MSGID(PAY0010)
// * 0010                                     MSGF(MYLIB/PAYMSGF)
```

The next message will be displayed on the screen:

```
PAYROLL EXECUTING
```

The second level message will be:

```
WEEKLY PROCESS
```

MSG

- The // MSG command is handled well, but there are other ways to send messages on AS/400. You can use the following commands: SNDMSG, SNDBRKMSG, SNDPGMMSG, and SNDNETMSG. The SNDUSRMSG, SNDPGMMSG, AND SNDNETMSG commands can only be used in a CL program.

```
// MSG W5,PLEASE SIGNOFF    SNDBRKMSG MSG(PLEASE SIGNOFF)
                                     TOMSGQ(W5) MSGTYPE(*INFO)

// MSG W3,GOOD MORNING      SNDMSG MSG('GOOD MORNING')
                                    TOMSGQ(W3)

// MSG W1,MOUNT TAPE        SNDUSRMSG MSG('MOUNT TAPE')
// PAUSE                              MSGTYPE(*INFO)
                                      TOMSGQ(W1)
```

NOHALT

- Handle NOHALT by monitoring messages in a CL program, using the MONMSG CL command.

OFF

- The OFF statement is converted to the SIGNOFF command.

  On System/36, DROP is the default. On AS/400, the default DROP value is an attribute defined in the device description. Either DROP(*YES) or DROP(*NO) may be specified on the SIGNOFF command.

  This command also allows you to print your job log.

```
// OFF DROP                  SIGNOFF LOG(*NOLIST) DROP(*DEVD)
```

PAUSE

The PAUSE statement converts to the SNDUSRMSG command with the same message text as the System/36 message that the operator is familiar with. You may want to use the SNDBRKMSG command as an alternative.

```
// PAUSE                     SNDUSRMSG MSG('PAUSE--WHEN READY, +
                                        ENTER 0 TO CONTINUE')
```

PRINTER

The priorities on System/36 are 0 through 5, with 5 as the highest. The priorities on AS/400 are 1 through 9, with 1 as the highest. Some parameters are not supported on the OVRPRTF command:

- CONTINUE - If you need to use the CONTINUE parameter, you have to understand shared file processing within a job. See the *Data Management Guide*.
- ACTIVITY - There is no AS/400 equivalent.
- TYPE - Not supported.
- EXTEN - Not supported.
- EOFMSG - There is no AS/400 equivalent
- TEXT - Not supported. Consider using the PRTQLTY keyword.

```
// PRINTER NAME-PRTA,DEVICE-P7,   OVRPRTF FILE(PRTA) DEV(P7)
           LINES-55,LPI-6,                PAGESIZE(55) LPI(6)
           CPI-10,ALIGN-YES,              CPI(10) ALIGN(*YES)
           SPOOL-NO,COPIES-2,             SPOOL(*NO) COPIES(2)
           CONTINUE-YES,HOLD-YES,         HOLD(*YES) OUTPTY(1)
           PRIORITY-4,FORMSNO-XX          FORMTYPE(XX)
```

PROMPT

On AS/400, the combination of the DCLF and SNDRCVF CL commands provides the closest equivalent. The following parameters are not directly supported:

- START - Parameters are used by name, not by position.
- LENGTH - Length parameters are not used because they are defined in the display file.
- PDATA - All parameters are variables used by a CL program that contains DCLF and SNDRCVF CL commands. See 15.15, "Prompting and Read Under Format" on page 157 for a discussion of differences from System/36.
- UPSI - There is no relationship between UPSI switches (1 through 8) and indicators (91 through 98), as in System/36.

RUN

On AS/400, the CALL CL command provides the equivalent of the // LOAD and // RUN pair of System/36 OCL statements.

START

- The START statement converts to the STRPRTWTR command. In some cases, you will need to add the FORMTYPE keyword.

```
// START PRT,P1,CHECK   STRPRTWTR DEV(P1) OUTQ(*DEVD)
                                          FORMTYPE(CHECK)
```

STOP

- The STOP statement is converted to the ENDWTR command.

```
// STOP PRT,P3              ENDWTR WTR(P3) OPTION(*CNTRLD)
```

SWITCH

- The // SWITCH statement is converted to the CHGJOB command. If you always want a job to start with a fixed set of switches, use CHGJOBD.

```
// SWITCH 100X11X0          CHGJOB SWS(100X11X0)
                            CHGJOBD JOBD(MYJOBD) SWS(100X11X0)
```

SYSLIST

- The SYSLIST OCL statement is converted to the OVRPRTF command. Alternatively, You can obtain the output from the AS/400 commands from a printer or screen by specifying the OUTPUT parameter in the related command.

```
// SYSLIST CRT                 OVRPRTF FILE(QSYSPRT) OUTQ(CRT)

// SYSLIST PRINTER             OVRPRTF FILE(QSYSPRT) OUTQ(PRINTER)

// SYSLIST CRT
// LISTLIBR DIR,LIBRARY,MYLIB  DSPLIB LIB(MYLIB) OUTPUT(*)

// SYSLIST PRINTER
// LISTLIBR DIR,LIBRARY,MYLIB  DSPLIB LIB(MYLIB) OUTPUT(*PRINT)
```

15.7 Procedure Control Expressions

Many of the expressions listed in Chapter 3 (Procedure Control Expressions) of the *System Reference for the System/36 Environment* were entered into a System/36 procedure and then converted with PTK.

Most expressions convert very well, although you should check the converted CL program to see that you have the equivalent result as the conversion results in multiple statements, including defined variables, for each OCL statement. In many cases the converted CL can be simplified to be more specific to the action required.

Appendix E has a table containing System/36 substitutions and how these can be resolved in a CL program.

Those expressions that do not convert are discussed below. Some of these are new ones provided for use in the AS/400 System/36 Environment.

?M'0011,1,20'? message substitution

The resulting expression uses a default message file name **MSGF(USRMSG).** Change this to the correct name.

?DEV'P7'? or ?DEV'?WS?'? device name substitution

This expression is new in the System/36 Environment to assist with mixed applications. It will not be needed if you convert your application entirely to native AS/400.

?FLIB? library name substitution

This expression is new in the System/36 Environment to assist with mixed applications. It is not needed if you convert your application entirely to native AS/400. The native AS/400 equivalent is the library list.

?MSGID? last error message substitution

This expression is new in the System/36 Environment to assist with mixed applications. It is not needed if you convert your application entirely to native AS/400. The native AS/400 equivalent is the MONMSG command.

?SLIB? library name substitution

Use RTVJOBA to find the name of the current library in the AS/400 list.

?F'S,name'? file size substitution

This is not required on the AS/400, so it can be discarded.

?MENU?, ?PROC?, ?VOLID?, etc.

There are some substitution expressions, like ?MENU?, ?PROC?, ?VOLID?, which have no equivalent on AS/400. These substitutions will either have to be removed or you will have to use data areas to store and retrieve the equivalent values.

15.8 Testing For Active Procedures

The IF ACTIVE function is used extensively in System/36 procedures, especially with batch or where there is dependent processing such as master file updating, but has no equivalent on the AS/400. This could create problems in the converted application system, depending on how you want to control the interdependent procedures (such as CL programs).

The following is given as a method for replicating the System/36 IF ACTIVE function:

- Create a data area in the procedure (CL program) PROCA.

```
CRTDTAARA DTAARA(MYLIB/PROCA) TYPE(*CHAR) LEN(1)
          VALUE('0') TEXT('Data area for PROCA active test')
```

- When this PROCA procedure (CL program) runs, the first step is to set a value into the data area to show active status

```
CHGDTAARA DTAARA(MYLIB/PROCA) VALUE('1')
```

- When this PROCA procedure ends, set the data area value to zero

```
CHGDTAARA DTAARA(MYLIB/PROCA) VALUE('0')
```

- In the dependent procedure (CL program) replace

```
// IF ACTIVE-'PROCA' * 'PROCA IS EXECUTING'
```

  with

```
DCL VAR(&ACTEST) TYPE(*CHAR) LEN(1)
.
.
.
RTVDTAARA DTAARA(MYLIB/PROCA) RTNVAR(&ACTEST)
     IF (&ACTEST *EQ '1') THEN(DO)
          SNDUSRMSG MSG('PROCA IS EXECUTING')
          ENDDO
```

  where the THEN(DO) to ENDDO has the required processing.

15.9 Evaluation

The EVALUATE procedure expression in the System/36 cannot be used in AS/400. Use the Declare Variables (DCL) and Change Variable (CHGVAR) commands to define variables and change their values.

Using OCL, you can create and change substitution variables easily. For example, EVALUATE P1='ABC' creates the variable, assigning it a value and an implicit length. On AS/400 this takes multiple statements, as is seen in the following example, where &Q is equivalent to P1. Note that &Q has to be specifically defined.

```
EVALUATE P1,5=3550
EVALUATE P1=?1?*2
```

```
PGM   PARM(&Q)
    DCL &Q *DEC 5
    CHGVAR &Q VALUE(3550)
    CHGVAR &Q VALUE(&Q * 2)
ENDPGM
```

Parameters in AS/400 are named, and there can be as many as you want in a procedure.

The arithmetic operations must be done by using the CHGVAR command. The division operator (/) must be preceded by a blank if the operand before it is a variable name. For example:

```
CHGVAR VAR(&Q) VALUE(&Q /2)
```

All the other arithmetic operators may be optionally preceded or followed by a blank.

15.10 Job Attributes and Job Control

A System/36 Environment procedure can use substitution expressions like ?CLIB?, ?DATE?, ?WS?, and ?USER? to obtain job attributes. To obtain these attributes in an AS/400 CL program, you must use the RTVJOBA (Retrieve Job Attributes) command. See Appendix E for further details.

15.11 Group Files

There are advantages to using group files in the System/36 Environment. You can save, restore, or delete all the files in a group using the SAVE, RESTORE, or DELETE procedures.

With AS/400 CL you will find that you can still process groups of objects by using generic parameter values for most commands. The generic value 'A*' just means "all objects starting with the character 'A'".

Instead of	use
`DELETE ALL,F1,,,,,,,A`	`DLTF FILE(QS36F/A*)`
`SAVE ALL,999,#SAVE,IBMIRD,A`	`SAVOBJ OBJ(A*) LIB(QS36F)`
`RESTORE ALL,#SAVE`	`RSTOBJ OBJ(A*) OBJTYPE(*FILE)`

Try to remove group files when converting to AS/400. This could be done by removing all the '.' characters. This can help simplify the CL programs by making the file names on the file specifications in your programs the same as the external name, thus reducing the need for OVRDBF (like the // FILE LABEL-) statements. However, this might introduce name conflicts that you will need to resolve. This would happen, for example, if you had two files named "A.BCD" and "ABCD". Also, use caution when converting from System/36 group file names to AS/400 generic names. Your commands might affect more files than you expected. For example, if files A.BCD, A.IJK, and AXYZ were on System/36, then DELETE ALL,F1,,,,,,,A would only delete A.BCD and A.IJK. But if the same files were in library QS36F on AS/400, DLTF FILE(QS36F/A*) would delete A.BCD, A.IJK, and AXYZ. If all of the files in a group have the same record formats, consider making them members of a single AS/400 file. The members can then be processed individually or together at run time by means of the OVRDBF statement.

Alternatively, you can get the same effect by:

1. Using repeated CL statements, one per file
2. Placing all the files in the group in a separate library, here they can be operated on together
3. Using the PDM in an interactive environment to perform the same operation on many different objects

On AS/400 you can still create files that have '.' as part of their name. Members in a file, too, can have names like System/36 group names. However, DDS, RPG, COBOL, and PL/I do not support a period as a valid character in any of their specifications, so when converting to native AS/400 you should remove the '.' and replace it with some other character, such as '$'.

15.12 Using the Local Data Area (LDA)

To change LDA using a CL program, you must use the CHGVAR command and the substring built-in function.

```
// LOCAL OFFSET-1,BLANK-*ALL
```

To clear the LDA you can use these commands:

```
CHGVAR VAR (%SST(*LDA 1 512)) VALUE ' '

CHGDTAARA DTAARA(*LDA (1 512)) VALUE ' '
```

To retrieve the information in the LDA from position 1 to 12, for example, and put it in a variable called &Q, use:

```
CHGVAR &Q %SST(*LDA 1 12)
```

Additionally, you can create named data areas in your library and pass parameters or data to them. This type of data area can be numeric, alphanumeric, or logical. You can also put in an initial value.

```
CRTDTAARA DTAARA(MYDTAARA) TYPE(*CHAR) LEN(128) VALUE('INITIAL')
```

Of course you can delete and change them, using the following commands:

```
CHGDTAARA
DLTDTAARA
```

15.13 System/36 System-Supplied Procedures

The System/36 procedures can be converted to AS/400 commands by using the table in Appendix B, "Procedure/CL Relation Table" on page 171. This appendix does not necessarily contain all of the System/36 procedures and their corresponding CL commands.

15.14 System/36 OCL Programming and CL Programming

On System/36, procedures, control commands, Operation Control Language (OCL) statements, and procedure control expressions are used to control all the jobs running in the system. On AS/400, the same functions are done by CL commands and CL programs. CL programs consist of compiled CL commands, and the commands themselves consist of the command statement, keywords,

and keyword values. Keyword values may be expressed as variables, constants, or expressions. Variables can be used as substitutes for most keyword values on CL commands. When a CL program variable is specified as a keyword value and the command containing it is run, the value of the variable is used as the keyword value. Every time the command is run, a different value can be substituted for the variable. Variables and expressions can be used as keyword values only in CL programs.

Program variables are not stored in libraries. They are not objects, and their values are destroyed when the program that contains them terminates. Variable names in CL programs must begin with an ampersand (&) followed by no more than 10 characters. The first character following the & must be alphabetic and the remaining characters alphanumeric, like &FILE or &DSPFL1.

In addition, variables can be used to:

- Pass information between programs.
- Pass information between programs and display devices.
- Conditionally process commands.
- Create objects.

For more information, see the *Control Language Programmer's Guide*.

15.14.1 Structure of a CL Program

The following table compares the structure of an OCL and a CL program, to give you an idea of how the main operations correspond:

Parts of a CL Program	Parts of a Procedure
PGM command (optional) Begins the program; declares parameters passed to it. PGM PARM(&A)	No equivalent in procedures. Parameters are not explicitly declared. No equivalent.
Program variables PGM PARM(&P1) DCL VAR(&P1) TYPE(*CHAR) LEN(4) CHVAR &P1 'ABC'	Substitution expressions not explicitly declared. ?1? // EVALUATE P1=ABC
Built-in functions %SST(*LDA 1 5) %SST(&PARM1 1 5) IF COND(%SWITCH(XX101X11)) ...	 ?L'1,5'? No direct equivalent // IF SWITCH-XX101X11 ...

Logic control commands
 	
IF
ELSE
DO
ENDDO
GOTO
	Procedure Control Expressions
	
	// IF
	// ELSE
	No equivalent
	No equivalent
	// GOTO

Program control commands
 	
CALL
 	
RETURN
TFRCTL
	Procedure control statements
	
	// INCLUDE
	or
	// LOAD
	// RUN
	// RETURN
	No equivalent

Functional commands DLTF CRTPF More examples in Appendix A	Procedure commands DELETE BLDFILE

Job control
 	
CHGJOB
 	
CHGCURLIB
More examples in Appendix B
	OCL statements
	
	// DATE
	// SWITCH
	// LIBRARY

ENDPGM command	No equivalent

The sequence, combination, and extent of these components are determined by the logic and design of your applications.

15.14.2 Passing Parameters

When you pass control to another CL program, you can also pass information to it for modification or use within the receiving program.

When you write a CL program, you must declare each parameter that can be passed to the program by specifying it on the PGM command and declaring it in the program.

When you call a CL program, you must specify exactly the same number of parameter values as are coded on the PGM command of the called program.

Here is an example of PROCA, which has a parameter passed to it, and which will pass a parameter to PROCB.

If they are System/36 Environment procedures, you can type:

```
PROCA PROGRAM1
```

If they are AS/400 CL programs, you can type:

```
CALL PROCA PARM(PROGRAM1)
```

Here is how you would code PROCA and PROCB in the System/36 Environment and in native AS/400:

System/36 Environment		Native AS/400	
PROCA	PROCB	PROCA	PROCB
-----	-----	-----	-----
// PROCB ?1?	// LOAD ?1?	PGM PARM(&PNAME)	PGM PARM(&PNAME)
	// RUN	DCL &PNAME *CHAR(8)	DCL &PNAME *CHAR(8)
		CALL PROCB +	CALL &PNAME
		PARM(&PNAME)	ENDPGM
		ENDPGM	

Parameters are passed by position, not name. For example if you call a program and pass parameters to it, your program will look like this:

```
CALL PROGC PARM(&A &B &C)
```

It passes three variables.

The program PROGC starts with:

```
PGM PARM(&X &Y &Z) /*  PROGB  */
```

The value of &A in PROGA is used for &X in PROGC, &B is used for &Y in PROGC, and so on. Also, the order of the DCL statements is unimportant. In addition to the position of the parameters, it is necessary to pay attention to their length and variable type.

When using System/36 procedures, the special value *ALL can be used on an INCLUDE OCL statement to pass all positional parameter values from a calling procedure to a called procedure. *ALL can also be used on the // RETURN statement to return all positional parameter values from the called procedure back to the calling procedure. In this way, the called procedure can directly change positional parameter values in the calling procedure.

AS/400 CL programs have a similar capability. The calling program must explicitly pass each parameter to the called program. If a CL variable is passed to a program and the called program modifies the variable, the modification is reflected in the variable in the calling program. However, if a constant value is passed to a program and the calling program modifies its value, the modification is not reflected in the calling program.

For example:

PGMA	PGMB
PGM DCL &VAR1 *DEC (15 5) DCL &VAR2 *DEC (15 5) CHGVAR &VAR1 VALUE(10) CHGVAR &VAR2 VALUE(10) CALL PBMB (&VAR1 &VAR2) CALL PGMB (1 2) ENDPGM	PGM (&PARM1 &PARM2) DCL &PARM1 *DEC (15 5) DCL &PARM2 *DEC (15 5) DCL &PARM2A *DEC (15 5) CHGVAR &PARM1 VALUE(&PARM1 + 1) CHGVAR &PARM2A VALUE(&PARM2 + 1) ENDPGM

PGMA calls PGMB, passing two values. On the first call, variables are passed. PGMB modifies the value of the first parameter by adding 1 to it. PGMA sets the variables passed to 10, so after the first call, variable &VAR1 has the value 11 in PGMA. PGMB also adds 1 to the value of the second parameter, but places the result in another variable. Since PGMB does not change the second parameter passed to it, &VAR2 still has a value of 10 in PGMA after the first call. On the second call, constant values are passed to PGMB. Even though PGMB modifies one of these values, the modification has no effect on PGMA.

15.14.3 IF and ELSE Commands

The procedure control statement IF has been improved in AS/400, because of the use of logic operators like *AND, *OR, and *NOT, and the DO, ENDDO commands, but there are some expressions which are not used (see Appendix F, "If Conditions and Their Equivalents" on page 193).

The IF command can help you in CL programs, because you can use it with the logic operator and DO commands.

The IF command is used to state a condition that, if true, specifies some other statement or group of statements in the program to be run. The ELSE command can be used with the IF command to specify a statement or group of statements to be run if the condition on the IF command is false. For example:

```
IF (&A *EQ &B) THEN(GOTO LABEL)
               ELSE(CALL PROG11)
```

In addition to the IF expressions like *EQ, *GT, and so on, there are other conditions that can be tested with IF in the System/36 Environment that must be changed when converting to CL. Some have equivalents, and others have to

create a group of CL commands to do the same job. For a table of these expressions, refer to Appendix F, "If Conditions and Their Equivalents" on page 193.

15.14.4 *AND, *OR, and *NOT Operators

These operators are reserved for logical operators used to specify the relationship between operands in logical expressions. These operators are not supported in System/36 OCL, but they might be useful to you in CL program logic.

Some examples using these expressions are:

```
((&DAY *EQ 30) *AND (&TIME *GT 1800))
((&DAY = 30) *AND (&TIME > 1800))
((&AGE *LT 18) *OR (&AGE *GT 60))
```

15.14.5 DO and ENDDO Commands

The DO command lets you process a group of commands together. The group is defined as all those commands between the DO and the corresponding ENDDO commands. This allows you to make your CL programs structured and avoid the GOTO and TAG statements you had to use in OCL procedures. For example:

```
IF (&A=&B) THEN(DO)
                ...
                CALL PGMX
                CHGVAR &A &D
                ...
                ENDDO
```

15.14.6 Mixing OCL and CL Programs

System/36 Environment procedures may contain a mixture of OCL statements and CL commands. These procedures must be run in the System/36 Environment. Just add CL statements to the procedures.

Generally, CL commands can be used wherever an OCL statement is valid. CL commands are not allowed where a utility control statement or source statement is required (after a // RUN and before // END or /*).

System/36 substitution expressions can be used on any part of a CL command, including the command name and keyword names. Additional substitution expressions have been defined for the System/36 Environment to assist in adding CL commands to procedures. These expressions allow you to:

- Determine the name of the System/36 Environment files library (?FLIB?).
- Determine if a message was issued as a result of a CL command (?MSGID?).
- Determine the AS/400 10-character device name that corresponds to a 2-character System/36 Environment device name (?DEVID?).

Refer to the *Concepts and Programmer's Guide for the System/36 Environment* and the *System Reference for the System/36 Environment* for more information about using these substitution expressions and CL commands in procedures.

CL commands can also be conditioned by // IF conditions. For example:

```
// IFF ?1?/  DSPFD ?1?
```

However, unlike OCL statements, it is not possible to condition parts of a CL command by placing each part on a separate line with a comma continuation character. If a CL command requires more than one line in the procedure, it can be continued with a plus (+) sign.

There are other restrictions. Only those CL commands that are allowed in an interactive environment are allowed in OCL procedures. The DCL and CHGVAR commands are not allowed. The MONMSG command is not allowed, so you cannot use it to test for error messages. In fact, AS/400 messages from CL commands in a procedure are simply ignored (they do not cause the procedure to fail). This is why you should use the new ?MSGID? substitution expression to determine whether an error has occurred while processing a CL command.

A procedure may be invoked from a CL program using the STRS36PRC CL command, as shown in Figure 37 on page 157. STRS36PRC will enter the System/36 Environment, run a procedure, and exit the System/36 Environment. The STRS36PRC command may not be used if a procedure is already running. Therefore, a CL program that uses STRS36PRC cannot be called from a procedure. This is to prevent recursive System/36 Environment jobs.

In the example, the first parameter passed from the CL program (&PARM01) becomes the first parameter ?1? in the OCL procedure. But even if ?1? is altered in the procedure, its new value is not automatically passed back to the CL program when you return. To pass a parameter back, use the *LDA.

This is the CL program that calls PROC111.

```
PGM         PARM(&PARM01)
DCL         VAR(&PARM01) TYPE(*CHAR) LEN(3)
MONMSG MSGID(CPF0000)
STRS36PRC   PRC(PROC111) PARM(&PARM01)
RTVDTAARA   DTAARA(*LDA (1 3)) RTNVAR(&PARM01)
ENDPGM
```

This is the procedure PROC111.

```
// * 'MESSAGE IS ?1?'
// PAUSE
// EVALUATE P1='XYZ'
// LOCAL OFFSET-1,DATA-'?1?'
```

Figure 37. Running a System/36 Environment Procedure from a CL Program

15.15 Prompting and Read Under Format

The two main purposes of the PROMPT command are to:

1. Prompt the workstation user for parameters needed in their program.
2. Allow the workstation user to type in data while the processing program is starting (performance).

The PDATA parameter no longer has a meaning on AS/400. Data and parameters are treated in the same way.

Your display file has already been created by the System/36 to AS/400 migration aid. It is described on the field level. Possibly it does not contain reasonable field names because on System/36 there is no requirement for field names. In this case you will find names like FL001 and FL002 after the migration aid generated them. It is up to you to decide whether the display file should be changed. If you have already changed your programs to use externally described display files, then you have already resolved those names.

15.15.1 Prompting for Parameters

If a PROMPT OCL statement specifies PDATA-NO, or omits the PDATA parameter, display station input is treated as procedure parameters. The following steps could be performed to convert the PROMPT OCL to CL:

1. Declare the display file (DCLF command) with optional RCDFMT(*ALL).

The CL compiler generates the variables for you if the display file is described on the field level (Figure 38 and Figure 39).

2. Issue a SNDRCVF against the declared file mentioning the record format name (member).

The display file is now sent to the workstation, and the CL program waits until the Enter key is pressed.

3. The compiler-generated variables will contain what the workstation user typed in.

```
A                                       DSPSIZ(24 80 *DS3)
A          R ORDERREC
A                                  5  4'ITEMNUMBER:'
A            ORDERNUM       3S 3B  5 18DSPATR(RI)
A                                  8  4'ITEMNAME:'
A            ITEMNAME      10A  B  8 18DSPATR(RI)
```

Figure 38. CL - DDS for SNDRCVF Prompting

```
                  DCLF       FILE(MYLIB/MYDSPF) RCDFMT(*ALL)
             &ORDERNUM       *DEC        3           3
             &ITEMNAME       *CHAR      10
                  SNDRCVF    DEV(*FILE) RCDFMT(ORDERREC)
                  ..other processing
Declared Variables
Name             Defined     Type            Length      References
&ITEMNAME           200      *CHAR             10            300
&ORDERNUM           200      *DEC               3 3          300
```

Note that the variables were picked up from the display file.

Figure 39. CL - Sample Program (Shortened)

15.15.2 Prompting for Data

If a PROMPT OCL statement specifies PDATA-YES, display station input is treated as data for the program's first input operation.

The same as above applies here, with the difference being that the SNDRCVF is replaced by the SNDF command in the CL program.

The display file should be recompiled with SHARE(YES).

Here are some ways for the application program to obtain data from the screen:

- change your application program to READ from the prompt format rather than from the file when reading the display file. Almost no change of the display file is required for this purpose.

  or

- Use the PASSRCD keyword on the file level in the display file. This will pass the specified record format to the application program. Almost no change of the application program is required for this purpose. A READ is required, but in RPG it will be to a blank format name. In COBOL, the READ will be to the file name.

In both cases, use the KEEP and ASSUME record level DDS keywords. KEEP causes the data management not to clear the screen when the program closes the display file. ASSUME causes data management to send the screen to the program when the program reads it (after OPEN). Almost no change in the application program is required for this purpose.

You might consider the record level INZRCD, which initializes the record on the screen for further typing.

For details, refer to the *Data Management Guide* and the *Data Description Specifications Reference*.

This prompting for data is shown in Figure 40 and Figure 41 on page 160.

```
--->                                    PASSRCD(ORDERREC)
--->                                    DSPSIZ(*DS3)
--->        R ORDERREC                  KEEP ASSUME
                                      5  4'ITEMNUMBER:'
            ORDERNUM       3S 3B   5 18DSPATR(RI)
                                      8  4'ITEMNAME:'
            ITEMNAME      10A  B   8 18DSPATR(RI)
```

Figure 40. CL - Sample Display File DDS Expanded for RUF

```
          DCLF       FILE(MYLIB/MYPRMTDF) RCDFMT(ORDERREC)
             &ORDERNUM      *DEC        3           3
             &ITEMNAME      *CHAR      10
                    SNDF       DEV(*FILE) RCDFMT(*ALL)
    /* Call the application program */
                    CALL       PGM(MYLIB/PROG3)
```

Figure 41. CL - Sample Program (Shortened) Expanded for RUF

15.15.3 Read Under Format (RUF)

The term read under format is no longer used on AS/400. In fact, there is no need for it because the storage restrictions of the System/36 no longer exist. Thus, program chaining using RUF is not required or recommended for native AS/400 applications. But for the purpose of conversion, there is a good technique to obtain the RUF.

Again use the KEEP and ASSUME keywords in the display file. Insert the PASSRCD keyword or change the program to mention a record format in the READ. Recompile the display file with SHARE(YES).

15.16 A Note about Auto Response

On System/36 you could specify an automatic response for certain messages system-wide by using the RESPONSE procedure. On AS/400 you can do the same by using the system reply list with the ADDRPYLE (Add Reply List Entry) command. There are more possibilities on AS/400 than on System/36. You can monitor for certain messages and take appropriate action in a CL program. You can also specify a message handling program within the message description. For details, refer to the *Control Language Programmer's Guide*.

15.17 Utilities

The PTK will convert most utilities and utility control statements (but not all), including the file OCL (other than the work file statements, if any), to corresponding CL commands.

You will need to check the generated CL commands to ensure that the result correctly replaces the intention of the utility control statements, because these are generally used to provide functions that the standard System/36 system procedures do not allow. We found that the PTK did not always convert the utility statements completely.

System/34 utilities that will function on the System/36 will, in general, convert less completely than the same System/36 utility when using PTK.

Where files are being created on output or being deleted (for example, with RETAIN-S parameter for the input file), you may have to provide the necessary CRTPF and DLTF CL commands, depending on the requirements and whether the characteristics of the output file are the same as the input file.

Some utilities, particularly $MAINT, do not convert as utilities when using PTK, but would convert when replaced by their corresponding system procedures. Refer to Appendix A, "UCS/Procedure Relation Table", for assistance in this process.

The following is an example of conversion of a $COPY utility. It is based on the output of PTK. EXAMPLE:

```
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-CASHM,RETAIN-S
// FILE NAME-COPYO,LABEL-CASHMTH,BLOCKS-10,RETAIN-T
// RUN
// COPYFILE OUTPUT-SAME,OMIT-EQ,POSITION-1,CHAR-'2',REORG-YES
// SELECT KEY,FROM-'12345',TO-'67890'
// END
```

Converts to

```
CPYF   FROMFILE(*LIBL/CASHM) TOFILE(QS36F/CASHMTH) +
         CRTFILE(*YES) COMPRESS(*YES) INCCHAR(*RCD 1 *NE 2) +
         FROMKEY(1 ('12345')) TOKEY(1 ('67890'))
DLTF   FILE(*LIBL/CASHM)
```

NOTE: File 'CASHMTH' will have the same characteristics as 'CASHM'

We recommend that you change the utility statements to corresponding System/36 system procedures, for example $COPY to COPYDATA, SAVE, RESTORE and then have PTK convert the system procedures.

15.18 Recommendations

Consider using the Programmer Tools PRPQ (5799-DAG). This tool will convert most System/36 procedures to CL programs without the need for manual conversion. *You must review the resulting CL programs to ensure that these perform the same functions as those intended by the original OCL.*

If you have procedures that contain control expressions, substitutions, system procedures, and utilities that do not convert, then use the appendixes in this document to convert to CL commands if possible. Where the conversion is not possible, you will either have to rewrite the CL program or create a program that will replace the function.

Alternatively, you could consider a full rewrite. The resources needed might be relatively low compared to the resources needed if extra functions have to be added to the application programs.

16.0 National Language Support Considerations

This section covers some considerations when your primary language is not US English.

16.1 Installing the PTK

As PTK provides only US English machine-readable information (MRI), you may have to be concerned with the multilingual environment at installation time. If you have a national language other than US English as the primary language, you need to have the US English MRI library named QSYS2924 as the secondary language into which the MRI of PTK is restored.

Install PTK as follows:

- Get the tape of PTK.
- Restore the objects using the RSTLICPGM command.

  ```
  RSTLICPGM LICPGM(5799DAG) DEV(TAP01) LNG(2924)
  ```

  where 2924 is the feature code of US English, and the device name "TAP01" may be different.

  - All objects of type *PGM go into the library QPTK.
  - All objects of type other than *PGM (MRI), go into the library QSYS2924.

When you use PTK, your job must have the library QSYS2924 in your system library list. This is done by the command CHGSYSLIBL.

```
CHGSYSLIBL LIB(QSYS2924) OPTION(*ADD)
```

Or the library QSYS2924 may be added to your system library list automatically at the sign-on time, because your display terminal is in US English, which is not the primary language in your system. Refer to Chapter 4 in the *Work Management Guide* to change the system library list at the sign-on time.

16.2 DBCS Considerations

When you convert your applications from double-byte character set (DBCS) System/36 to DBCS AS/400, you should be aware of several things that are not taken care of by PTK:

- DDS source files:

  PTK creates a DDS source physical file that is not a DBCS-capable file. Therefore, you will have to create another source physical file with DBCS capability to include DBCS text and column headings. Specify IGCDTA(*YES) on the CRTSRCPF CL command to create the file. After creating the DBCS-capable file, you can copy the DDS output of PTK to it.

- Database files

  PTK does not set the DBCS attribute in database files when it generates the DDS from internal descriptions. That means if you want to create the

DBCS-capable database file, you have to specify "O" in the data type field of the DDS source generated by PTK. (You can create the DBCS-capable file by specifying IGCDTA(*YES) in the CRTPF command, but this is mutually exclusive with SRCFILE parameter. If you use the DDS, you need to specify "O" in the data type field to create the DBCS-capable file.) All DDS generated by PTK should be examined to determine whether the DDS should have this data type.

In the meantime, all files in the DBCS System/36 are migrated to DBCS-capable files in AS/400. This is true even if a file contains only numeric fields. In AS/400 it is impossible to create a DBCS-capable file that has only numeric fields by using DDS. (The data type "O" cannot be specified for the numeric fields.) When you copy a file in the System/36 Environment that is DBCS-capable to an AS/400 database file that is not DBCS-capable, you get the information message CPF2826 saying DBCS data may be copied to a non-DBCS file. You can ignore this message, because all records are copied and there should be no DBCS data in the numeric-only fields file.

17.0 Systems Application Architecture (SAA) Considerations

OS/400 participates in IBM Systems Application Architecture (SAA) and joins OS/2, VM, and MVS as a major operating environment.

When you convert or redesign your application programs, it may be a good time to consider complying with SAA interfaces.

The major interfaces when you convert the application programs will be:

- Common programming interface (CPI)
  RPG/400 and COBOL/400 are SAA CPI languages.

  The native RPG and COBOL compilers have the capability of flagging statements that do not comply with the respective SAA CPI definitions. If you specify SAAFLAG(*FLAG) on the CRTRPGPGM or CRTCBLPGM CL command (the default is SAAFLAG(*NOFLAG)), the compiler will flag the statements that do not comply and summarize them under an "SAA Message Summary" in the compiler listing. This function is not available with the System/36 or System/36-compatible compilers.

  If you intend to comply fully with SAA from a CPI point of view, you should also consider using Stuctured Query Language/400 (SQL/400) and Procedures Language 400/REXX, which are the SAA data base and procedures language CPIs. Converting System/36 Environment applications to use SQL and REXX should probably be considered for only your most important applications because the effort will be nearly the same as completely redesigning the applications.

- Common user access (CUA).
  Your screen formats may be different than what SAA prescribes. For example, CMD7 is the standard key in System/36 to terminate, but it is F3 for SAA. Changing your user interface to comply with SAA rules may require significant effort, but this is the time to consider whether you want to participate in SAA from a CUA point of view.

Refer to the following SAA manuals for more details:

- *An Overview*
- *CUA Basic Interface Design Guide*
- *Writing Applications: A Design Guide*
- *Common Programming Interface:*
  - *Application Generator Reference*
  - *C Reference*
  - *COBOL Reference*
  - *Database Reference*
  - *Dialog Reference*
  - *FORTRAN Reference*
  - *Presentation Reference*
  - *Procedures Language Reference*

- *Query Reference*
- *Communications Reference*

Also, the ITSC publication, *Writing SAA Applications for AS/400*, contains explanations and examples of SAA-conforming applications.

Appendix A. UCS/Procedure Relation Table

The following table shows the relationship between System/36 utilities, with their utility control statements, and the corresponding System/36 procedure commands. If your applications use the utilities and utility control statements, you may use this table to find which System/36 procedure commands perform the same functions. Then refer to Appendix B, "Procedure/CL Relation Table" on page 171, to find the native AS/400 equivalent of the procedure commands.

System/36 Utility	System/36 Procedure Command
$BICR // TRANSFER	TRANSFER
$BMENU // MENU	BLDMENU

$COPY
// COPYFILE
// COPYALL
// COPYADD
	
	COPYDATA, LISTDATA, SAVE, RESTORE
	SAVE, RESTORE
	SAVE

$CPPE // ERR	ERR
$DDST // KEYSORT	KEYSORT

$DELET
// SCRATCH
// REMOVE
	DELETE

$DUPRD // COPYI1	COPYI1
$FBLD // FILE	BLDFILE, BLDINDEX
$FREE // COMPRESS	COMPRESS
$GSORT // SOURCE	SORT

continued on next page

System/36 Utility	System/36 Procedure Command
$HIST // DISPLY	HISTORY

$INIT
// UIN
// VOL
	INIT

$KASRT // SOURCE	SRTX
$LABEL // DISPLAY	CATALOG

$MAINT
// ALLOCATE
// COPY
 	
// CHANGE
// COMPRESS
// DELETE
// COPYLIBR
	
	ALOCLIBR, BLDLIBR
	FROMLIBR, TOLIBR, JOBSTR,
	LISTLIBR, LISTFILE, BLDLIBR
	CHNGEMEM
	CONDENSE
	REMOVE
	SAVELIBR, RESTLIBR

$MGBLD // MGBLD	CREATE
$RENAM // RENAME	RENAME
$SETCF // SETCF	SET
$TCOPY // TRANSFER	TAPECOPY
$TINIT // VOL	TAPEINIT

continued on next page

System/36 Utility	System/36 Procedure Command
$TMSERV	
// ARCHIVE	ARCHIVE
// LISTARCH	LISTFILE
// MOVEFLDR	MOVEFLDR
// RESTFLDR	RESTFLDR
// RETRIEVE	RETRIEVE
// RORGFLDR	ALOCFLDR, CONDENSE
// SAVEFLDR	SAVEFLDR
$UASC	COPYPRT
$UASF	COPYPRT
// SPOOL	

Appendix B. Procedure/CL Relation Table

The following table shows the relationships among procedures in System/36, System/36 Environment, and AS/400 Native CL commands. This table is not limited to the conversions performed by PTK, nor does it necessarily contain all the procedures and CL commands available.

Remember that native AS/400 CL commands may also be used in the System/36 Environment.

System/36 Procedure	System/36 Environment	Native AS/400
ALERT	(1)	CHGMSGD
ALOCFLDR	(1)	(3)
ALOCLIBR	(2)	(3)
ALTERBSC	(1)	WRKLIND (5)
ALTERCOM	(1)	WRKLIND (5) GO CFGDEVCMN
ALTERSDL	(1)	WRKLIND (5)
APAR	(1)	CRTAPAR
ARCHIVE	(1)	SAVDLO
ASM	(1)	(3)
AUTOC	AUTOC	CRTRPTPGM CRTS36RPT
BALPRINT	(1)	(1)
BASIC	(1)	STRBAS
BASICP	(1)	STRBASPRC
BASICR	(1)	CALL
BASICS	(1)	CRTBASPGM

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
BGUCHART	(1)	STRBGU
BGUDATA	(1)	STRBGU
BGUGRAPH	(1)	STRBGU
BLDFILE	BLDFILE	CRTPF (sequential) CRTLF (indexed)
BLDINDEX	BLDINDEX	CRTLF
BLDLIBR	BLDLIBR	CRTLIB, CRTSRCPF (named QS36PRC) RSTS36LIBM
BLDMENU	BLDMENU	STRSDA CRTMNU
CATALOG	CATALOG	DSPLIB DSPDKT DSPFD DSPOBJD QUSRTOOL/PRTLIBANL (6)
CHNGEMEM	CHNGEMEM	RNMM RNMOBJ
CNFIGICF	(1)	GO CFGDEVCMN
CNFIGSSP	(1)	CHGS36 G0 LICPGM
CNFIGX25	(1)	GO CFGDEVCMN
COBOL	(1)	CRTCBLPGM
COBOLC	COBOLC	CRTCBLPGM
COBOLONL	(1)	STRPDM
COBOLP	(1)	STRPDM
COBSDA	COBSDA	STRPDM STRSDA
COBSEU	COBSEU	STRPDM STRSEU

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
COMPRESS	(1)	(2)
CONDENSE	(1)	(2)
COPYDATA	COPYDATA	CPYF RGZPFM
COPYI1	COPYI1	DUPDKT
COPYPRT	COPYPRT	CPYSPLF DSPPFM WRKSPLF
CREATE	CREATE	CRTMSGF
DATE	DATE	CHGJOB
DEFINEID	(1)	CRTCTLBSC CHGCTLBSC GO CFGDEVCMN
DEFINEPN	(1)	WRKCTLD (5)
DEFINLOC	(1)	CRTCFGL CHGCFGL GO CFGDEVCMN
DEFINX21	(1)	WRKCTLD (5)
DEFINX25	(1)	CRTCFGL CHGCFGL WRKCTLD
DELETE	DELETE	DLTLIB DLTDKTLBL DLTF CRLDKT DLTDLO
DELNRD	(1)	WRKDDMF (5) DLTF
DFU	DFU	STRDFU
DISABLE	(1)	VRYCFG
DSU	DSU	STRPDM
EM3270	EM3270	STREML3270

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
ENABLE	(1)	VRYCFG
ENTER	ENTER	STRDFU CRTPF CHGDTA
ENTER#	ENTER#	STRDFU
EP3270	EP3270	STREML3270
ERR	ERR	SNDBRKMSG SNDPGMMSG TYPE(*ESCAPE) SNDUSRMSG
ES3270	ES3270	STREML3270
FORMAT	FORMAT	CRTDSPF
FORTRAN	(1)	(1)
FROMLIBR for S/36 interchange	SAVS36LIBM	SAVS36LIBM
FROMLIBR for AS/400 interchange	FROMLIBR	SAVLIB SAVOBJ
HELP	HELP	GO CMDHLP
HISTCOPY	(1)	(3)
HISTORY	(1)	GO CMDLOG
ICFDEBUG	(1)	STRDBG
IDDU	IDDU	WRKDTADCT (5)
IDDUDCT	IDDUDCT	WRKDTADCT (5)
IDDUDFN	IDDUDFN	WRKDTADFN (5)
IDDUDISK	IDDUDISK	WRKDBFIDD (5)

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
IDDULINK	IDDULINK	LNKDTADFN
IDDUPRT	IDDUPRT	DSPDTADCT
IDDURBLD	(1)	(3)
IDDUXLAT	(1)	(1)
IGC	IGC	(3)
INIT	INIT	CLRDKT INZDKT RNMDKT
INQUIRY	INQUIRY	STRDFU DSPDTA
INQUIRY#	INQUIRY#	STRDFU
IPL	(1)	PWRDWNSYS
ITF	ITF	STRITF
JOBSTR	JOBSTR	STRDKTRDR
KEYS	(1)	GO CMDKBD
KEYSORT	KEYSORT	(3)
LIBRLIBR	LIBRLIBR	CRTDUPOBJ CPYSRCF
LINES	LINES	OVRPRTF
LIST	LIST	GO CMDQRY RUNQRY
LIST#	LIST#	GO CMDQRY

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
LISTDATA	LISTDATA	CPYF CPYFRMDKT CPYFRMTAP
LISTFILE	LISTFILE	CPYF CPYFRMDKT CPYFRMTAP DSPDKT DSPTAP DMPTAP DSPLIB
LISTLIBR directory information	LISTLIBR	DSPLIB DSPFD QS36SRC DSPFD QS36PRC DSPOBJD QUSRTOOL/PRTLIBANL (6) CPYSRCF
LISTLIBR ALL	LISTLIBR	CPYSRCF QS36SRC *PRINT CPYSRCF QS36PRC *PRINT
LISTNRD	(1)	WRKDDMF
LOAD3601	(1)	(1)
LOG	LOG	CHGJOB
MSGFILE	(1)	GO CMDMSG
MSRJE	MSRJE	GO CMDRJE STRRJECSL SBMRJEJOB
NOHALT	NOHALT	MONMSG CHGRPYLE
OLINK	(1)	(3)
ORGANIZE	COPYDATA	CPYF
PASSTHRU	(1)	STRPASTHR
PASSWORD	PASSWORD	CHGPW

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
PATCH	(1)	STRSST
POST	(1)	GO CMDCPY
PRINT	PRINT	OVRPRTF CHGJOB CHGDEVDSP
PRINTKEY	(1)	CHGDEVDSP
PRTGRAPH	PRTGRAPH	STRBGU
QRY	QRY	GO CMDQRY WRKQRY
QRYDE	QRYDE	UPDDTA STRDFU
QRYRUN	QRYRUN	RUNQRY
REMOVE	REMOVE	DLTPGM RMVM CLRLIB DLTPGM
RENAME	RENAME	RNMOBJ
RESPONSE	RESPONSE	MONMSG CHGRPYLE WRKRPYLE
RESTFLDR	(1)	RSTDLO
RESTLIBR S/36 interchange	RSTS36LIBM	RSTS36LIBM
RESTLIBR AS/400 interchange	RESTLIBR	RSTLIB CPYFRMTAP
RESTNRD	(1)	RSTOBJ

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
RESTORE S/36 interchange	RSTS36F	RSTS36F
RESTORE AS/400 interchange	RESTORE	RSTOBJ
RETRIEVE	(1)	RSTDLO
RJFILE	RJFILE	CVTRJEDTA
RJTABLE	RJTABLE	GO CMDFCTE GO CMDFCR
ROLLKEYS	(1)	CHGPRF
RPGC	RPGC	CRTRPGPGM
RPGONL	GO S36PGMLNG	STRPDM
RPGP	GO S36PGMLNG	STRPDM
RPGR	RPGR	STRPDM RPGR
RPGSDA	RPGSDA	STRPDM STRSDA
RPGSDA	RPGSEU	STRPDM STRSEU
RPGX	RPGX	CRTRPGPGM
SAVE S/36 interchange	SAVS36F	SAVS36F
SAVE AS/400 interchange	SAVE	SAVOBJ
SAVEEXTN	(1)	CPYIGCTBL

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
SAVEFLDR	(1)	SAVDLO
SAVELIBR S/36 interchange	SAVS36LIBM	SAVS/36LIBM
SAVELIBR AS/400 interchange	SAVLIBR	SAVLIB
SAVENRD	(1)	SAVOBJ
SDA	SDA	STRSDA
SET	SET	CHGJOB CHGDEVD CHGPRF
SETALERT	(1)	CHGMSGD
SETCOMM	(1)	CHGDEVCMN
SETDUMP	(1)	GO CMDDBG GO CMDBKP
SEU	SEU	STRSEU
SLIB	SLIB	ADDLIBLE CHGCURLIB
SOFTWARE	(1)	GO LICPGM
SORT	SORT	FMTDTA
STARTM	(1)	VRYCFG
STOPGRP	(1)	ENDMOD
STOPM	(1)	VRYCFG
STRTGRP	(1)	VRYCFG STRMOD
SWITCH	SWITCH	CHGJOB

continued on next page

System/36 Procedure	System/36 Environment	Native AS/400
SYSLIST	SYSLIST	OVRPRTF
TAPECOPY	TAPECOPY	CPYFRMTAP CPYTOTAP
TAPEINIT	TAPEINIT	INZTAP
TAPESTAT	(1)	PRTERRLOG
TOLIBR S/36 interchange	RSTS36LIBM	RSTS36LIBM
TOLIBR AS/400 interchange	TOLIBR	RSTOBJ
TRACE	(1)	TRCINT
TRANSFER	TRANSFER	CPYFRMDKT CPYTODKT CPYSPRPF
UPDATE	UPDATE	STRDFU CHGDTA
UPDATE#	UPDATE#	STRDFU
WSU	(1)	(1)

NOTES:

(1) This procedure or command is not supported in this environment.

(2) This is checked for syntax only. No operation is performed.

(3) This procedure or command is no longer needed.

(4) CALL and GO command run objects created in native AS/400.

(5) The WRKxxxx commands work only in interactive mode.

(6) This is available in library QUSRTOOL in Release 1.2.

Appendix C. OCL/CL Relation Table

The following table shows the relationships among System/36 OCL statements and their counterparts in System/36 Environment and native AS/400. This table does not necessarily show all the CL commands available.

Remember that native AS/400 CL commands may also be used in the System/36 Environment.

System/36 OCL	System/36 Environment	Native AS/400
ALLOCATE	ALLOCATE	ALCOBJ
ATTR	ATTR	CHGJOB WRKJOB (5) SBMJOB RVKOBJAUT
CANCEL	CANCEL	CHGSPLFA DLTSPLF WRKOUTQ (5) WRKSPLF
CHANGE	CHANGE	CHGSPLFA CHGWTR WRKOUTQ (5) WRKSPLF
DATE	DATE	CHGJOB WRKJOB
DEALLOC	DEALLOC	DLCOBJ
EVOKE	EVOKE	SBMJOB CALL STRS36PRC RTVOBJA
FILE	FILE	OVRDBF OVRDKTF OVRTAPF
FORMS	FORMS	OVRPRTF
INCLUDE	INCLUDE	CALL

continued on next page

System/36 OCL	System/36 Environment	Native AS/400
INFOMSG	INFOMSG	CHGMSGQ
JOBQ	JOBQ	SBMJOB
LIBRARY	LIBRARY	ADDLIBLE CHGCURLIB CHGJOBD CHGPRF CHGLIBL CHGUSRPRF
LOAD	LOAD	CALL (4)
LOCAL	LOCAL	CHGDTAARA CHGVAR
LOG	LOG	CHGJOB CHGJOBD
MEMBER	MEMBER	OVRMSGF SNDxxxMSG
MENU	MENU	GO (4) CALL
MSG	MSG	SNDBRKMSG SNDMSG SNDNETMSG SNDPGMMSG
NOHALT	NOHALT	MONMSG
OFF	OFF	SIGNOFF
POWER	(1)	PWRDWNSYS
PRINTER	PRINTER	OVRPRTF
PROMPT	PROMPT	DCLF & SNDRCVF

continued on next page

System/36 OCL	System/36 Environment	Native AS/400
REGION	REGION	None
RESERVE	RESERVE	None
RESET	RESET	TFRCTL
RUN	RUN	CALL (4)
SESSION	SESSION	OVRICFDEVE CALL
SWITCH	SWITCH	CHGJOB
START	START	STRPRTWTR
STOP	STOP	ENDWTR
SYSLIST	SYSLIST	None OVRPRTF
VARY	VARY	VRYCFG WRKCFGSTS
WAIT	WAIT	DLYJOB
WRKSTN	WRKSTN	OVRDSPF

NOTES:

(1) This procedure or command is not supported in this environment.

(4) CALL and GO command run objects created in AS/400 native. On AS/400, the CALL command is the equivalent of the combination of the System/36 // LOAD and // RUN statements.

(5) The WRKxxxx commands only work in interactive mode.

Appendix D. OCC and CL Command Table

The following table shows the relationships among operator control commands (OCC) in System/36, System/36 Environment, and native AS/400 CL commands. This table does not necessarily contain all the operator control commands and CL available.

System/36 OCC	System/36 Environment	Native AS/400
CANCEL	CANCEL	
C P,spoolid	C P,spoolid	DLTSPLF
C P,FORMS	C P,FORMS	WRKOUTQ (5)
C P,USER	C P,USER	WRKOUTQ (5)
C J,jobname	C J,jobname	WRKJOBQ (5)
C J,ALL	C J,ALL	CLRJOBQ
C S.id	(1)	WRKUSRJOB (5)
C jobname	(1)	ENDJOB jobname *IMMED
CHANGE	CHANGE	
G COPIES	G COPIES	CHGSPLFA COPIES or WRKOUTQ (5)
G DEFER	G DEFER	CHGSPLFA SCHEDULE or WRKOUTQ (5)
G FORMS	G FORMS	CHGSPLFA FORMTYPE or WRKOUTQ (5)
G ID	G ID	CHGSPLFA OUTQ or WRKOUTQ (5)
G PRT	G PRT	CHGSPLFA PRTSEQ
G PRTY	(1)	CHGJOB
G SEP	G SEP	CHGWTR
G JOBQ	(1)	WRKJOBQ (5)
CONSOLE	(1)	CHGMSGQ QSYSOPR
HOLD	HOLD	
H P,spoolid	H P,spoolid	HLDSPLF or WRKWTR (5)
H P,prtid	H P,prtid	HLDWTR PRTID or WRKWTR (5)
H P,ALL	H P,ALL	HLDWTR PRTID(s) or WRKWTR (5)
H J,jobname	(1)	HLDJOB or WRKWTR (5)

continued on next page

System/36 OCC	System/36 Environment	Native AS/400
INFOMSG	INFOMSG	CHGMSGQ
JOBQ	JOBQ	SBMJOB
MENU	MENU	GO (4)
MODE	(1)	(1)
MSG	MSG	SNDMSG DSPMSG SNDBRKMSG
OFF	OFF	SIGNOFF
POWER	(1)	PWRDWNSYS
PRTY	(1)	CHGJOB WRKUSRJOB
RELEASE L P,spoolid L P,prtid L P,ALL L JOBQ L JOBQ,jobname	RELEASE L P,spoolid L P,prtid L P,ALL L JLF L JR	 WRKOUTQ (5) WRKWTR (5) WRKWTR (5) WRKJOBQ (5) WRKUSRJOB (5)
REPLY	(1)	DSPMSG
RESTART	(1)	ENDWTR STRPRTWTR
START S P,prtid S P,ALL S JOB S J S J,prtq S SERVICE S N S S S W	START S P,prtid S P,all (1) S J S J,prtq (1) (1) (1) (1)	 STRPRTWTR SWRPRTWTR *ALL RLSJOB RLSJOBQ WRKJOBQ (5) GO CMDPRBMGT VRYCFGSTS *LIN STRSBS or STRPRTWTR *ALL STRSBS or WRKCFGSTS *DEV (5)

continued on next page

System/36 OCC	System/36 Environment	Native AS/400
STATUS	STATUS	
D S,id	D S,id	WRKJOB (5)
D C	(1)	WRKCFGSTS (5)
D H	(1)	WRKCFGSTS (5)
D L	(1)	WRKCFGSTS (5)
D J	D J	WRKJOBQ (5)
D P	D P	WRKOUTQ (5)
D N	(1)	WRKCFGSTS (5)
D I	(1)	WRKCFGSTS (5)
D A	(1)	DSPAPPNINF or WRKCFGSTS (5)
D M	(1)	GO CMDRJE
D W	D W	WRKCFGSTS (5)
D WRT	D WRT	WRKWTR (5)
D T	(1)	WRKACTJOB (5)
D G	D G	DSPMSG
D U	D U	WRKACTJOB or WRKUSRJOB (5)
STOP	STOP	
P P	P P	ENDWTR or HLDWTR
P P,ALL	P P,ALL	ENDWTR *ALL
P job	(1)	WRKACTJOB (5)
P J	P J	WRKJOBQ or HDLJOBQ (5)
P N	(1)	WRKCFGSTS (5)
P S	(1)	ENDSYS
P W,id	(1)	WRKCFGSTS (5)
P W,ALL	(1)	ENDSBS
TIME	TIME	DSPSYSVAL QTIME DSPSYSVAL QDATE
VARY		
V xxx,id	V xxx,id	VRYCFG
V xxx,P	V xxx,P	VRYCFG *DEV
V xxx,I1	V xxx,I1	VRYCFG *DEV
V xxx,ctlid	(1)	VRYCFG *CTL
V xxx,,line	(1)	VRYCFG *LIN
V xxx,Tx	(1)	VRYCFG *DEV

NOTES:

(1) This procedure or command is not supported in this environment.

(4) CALL and GO command run objects created in AS/400 native.

(5) The WRKxxxx commands only work in interactive mode.

Appendix E. Table of System/36 Substitution Expressions

This table contains the System/36 substitutions and how they could be resolved within a CL program. This table does not necessarily contain all the substitution expressions and CL expressions available.

Conversion of substitution expressions depends on the context in which they occur. For example, ?12? will convert, but ??12?? will not.

System/36 Substitution	AS/400 CL
?n?	DCL VAR(&PRMn) TYPE(*CHAR)

?n'string'?	DCL VAR(&PRMn) TYPE(*CHAR)
	IF (&PRMn *EQ ' ') THEN(DO)
	CHGVAR &PRMn VALUE('string')
	ENDDO

?nT'string'?	DCL VAR(&WRKx) TYPE(*CHAR)
	DCL VAR(&PRMn) TYPE(*CHAR)
	IF (&PRMn *EQ ' ') THEN(DO)
	CHGVAR &WRKx VALUE('string')
	ENDDO
	ELSE CMD(DO)
	CHGVAR &WRKx VALUE(&PRMn)
	ENDDO

?nF'string'?	DCL VAR(&PRMn) TYPE(*CHAR) CHGVAR &PRMn VALUE('string')
?R?	DCL VAR(&RPLOn) TYPE(*CHAR) SNDUSRMSG MSG('Enter Parameter Req') MSGRPY(&RPLOn)

?nR?	DCL VAR(&PRMn) TYPE(*CHAR)
	IF (&PRMn *EQ '1') THEN(DO)
	SNDUSRMSG MSG('Enter Missing Parameter')
	MSGRPY(&PRMn)
	ENDDO

continued on next page

System/36 Substitution	AS/400 CL
?R'mic'?	DCL VAR(&WRKn) TYPE(*CHAR) SNDUSRMSG MSGID('USR' *TCAT 'mic') MSGF(*LIBL/USRMSG) MSGRPY(&WRKn)

?nR'mic'?	DCL VAR(&PRMn) TYPE(*CHAR)
	IF (&PRMn *EQ ' ') THEN(DO)
	SNDUSRMSG MSGID('USR' *TCAT 'mic')
	MSGF(*LIBL/USRMSG)
	MSGRPY(&PRMn)
	ENDDO

?Cn?	DCL VAR(&PRMn) TYPE(*CHAR) LEN(128)
	DCL VAR(&LEN01) TYPE(*DEC) LEN(3 0)
	DCL VAR(&WRKx) TYPE(*CHAR) LEN(32)
	CHGVAR &LEN01 VALUE(32)
	CNVTAGxx:
	IF COND(%SST(&PRMn &LEN01 1) *EQ ' ')
	THEN(DO)
	CHGVAR VAR(&LEN01) VALUE(&LEN01 - 1)
	IF (COND(&LEN01>0) THEN(GOTO CNVTAGxx)
	ENDDO
	CHGVAR VAR(&WRKx) VALUE(&LEN01)
	IF COND(%SST(&WRKx 30 3) *EQ value)
	to verify length
	THEN(DO)
	ENDDO

?C'string'?	DCL VAR(&LEN01) TYPE(*DEC) LEN(3 0)
	DCL VAR(&WRKx) TYPE(*CHAR) LEN(32)
	CHGVAR &WRKx VALUE('string')
	CHGVAR &LEN01 VALUE(32)
	CNVTAGxx:
	IF COND(%SST(&WRKx &LEN01 1) *EQ ' ')
	THEN(DO)
	CHGVAR VAR(&LEN01) VALUE(&LEN01 - 1)
	IF (&NUM04 *GT 0) THEN(GOTO CNVTAGxx)
	ENDDO
	CHGVAR VAR(&WRKx) VALUE(&LEN01)
	IF COND(%SST(&WRKx 30 3) *EQ value)
	to verify length
	THEN(DO)
	ENDDO

continued on next page

System/36 Substitution	AS/400 CL
?CD? Return Codes Examples 0000 2001 - 2024 2090 , 2091 2092 , 2093	DCL VAR(&RTGDA) TYPE(*CHAR) LEN(80) DCL VAR(&WRKX) TYPE(*CHAR) CHGVAR &WRKX VALUE(%SST(&RTGDA 1 2))
?CLIB?	DCL VAR(&WRKx) TYPE(*CHAR) DCL VAR(&LIBLIST) TYPE(*CHAR) LEN(275) RTVJOBA USRLIBL(&LIBLIST) CHGVAR &WRKx VALUE(%SST(&LIBLIST 1 10))
?DATE?	DCL VAR(&WRKx) TYPE(*CHAR) RTVJOBA DATE(&WRKx)
?DEV'unit'? ?FLIB?	System/36 Environment only Function not supported
?F'S,name'?	Function not Supported
?F'A,name'?	Function not Supported
?L'pos,length'?	DCL VAR(&WRKx) TYPE(*CHAR) DCL VAR(&NUM1) TYPE(*DEC) LEN(3 0) DCL VAR(&NUM2) TYPE(*DEC) LEN(3 0) CHGVAR VAR(&NUM1) VALUE('pos') CHGVAR VAR(&NUM2) VALUE('length') CHGVAR &WRKx VALUE(%SST(*LDA &NUM1 &NUM2)) or RTVDTAARA DTAARA(*LDA (&NUM1 &NUM2) + RTNVAR(&WRKx)
?Mmic?	DCL VAR(&MSG01) TYPE(*CHAR) LEN(75) RTVMSG MSG('USR' *TCAT 'mic') MSGF(USRMSG) MSGF(&MSG01)

continued on next page

System/36 Substitution	AS/400 CL
?M'mic,pos,length'?	DCL VAR(&MSG1) TYPE(*CHAR) LEN(75) DCL VAR(&NUM2) TYPE(*DEC) LEN(3 0) DCL VAR(&NUM3) TYPE(*DEC) LEN(3 0) CHGVAR VAR(&NUM2) VALUE('pos') CHGVAR VAR(&NUM3) VALUE('length') RTVMSG MSGID('USR' *TCAT 'mic') MSGF(USRMSG) MSGF(&MSG1) CHGVAR &MSG1 VALUE(%SST(&MSG1 &NUM2 &NUM3))
?MSGID?	System/36 Environment only Function not supported
?MENU? ?PRINTER? ?PROC? ?SFLIB? ?SLIB? ?SYSLIST?	Function not supported Function not supported Function not supported Function not supported Function not supported Function not supported
?TIME?	DCL VAR(&WRKx) TYPE(*CHAR) DCL VAR(&TIME1) TYPE(*CHAR) LEN(6) RTVSYSVAL SYSVAL(QTIME) RTNVAR(&TIME1) CHGVAR VAR(&WRKx) VALUE(&TIME1)
?USER?	DCL VAR(&WRKx) TYPE(*CHAR) RTVJOBA USER(&WRKx)
?VOLID?	Function not supported
?WS?	DCL VAR(&WSID1) TYPE(*CHAR) LEN(2) DCL VAR(&WRKx) TYPE(*CHAR) RTVJOBA JOB(&WRKx) CHGVAR VAR(&WSID1) VALUE(%SST(&WRKx 1 2))

Appendix F. If Conditions and Their Equivalents

System/36 Condition	CL Equivalent
IF /	IF *EQ
IF =	IF *EQ
IFT /	IF *EQ
IFT =	IF *EQ
IFF /	IF *NE
IFF =	IF *NE
IF >	IF *GT
IFF >	IF *NG
ACTIVE	Function not supported. However see 15.8, "Testing For Active Procedures" on page 149
BLOCKS	Function not supported Not relevant to AS/400
CONSOLE	For CONSOLE-YES DCL VAR(&JOB) TYPE(*CHAR) LEN(10) DCL VAR(&CONSOLE) TYPE(*CHAR) LEN(10) RTVJOBA JOB(&JOB) RTVSYSVAL SYSVAL(QCONSOLE) RTNVAR(&CONSOLE) IF COND(&JOB *EQ &CONSOLE) THEN(DO) ENDDO For CONSOLE-NO IF COND(&JOB *NE &CONSOLE) THEN(DO)

DATAF1-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKOBJ OBJ(Name) OBJTYPE(*FILE) MBR(Name)
	MONMSG MSGID(CPF0000) EXEC((CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN DO
	ENDDO

DATAI1-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR &RTNCD VALUE('0')
	CHKDKT LOC(Loc) LABEL(Name)
	MONMSG MSGID(CPF0000) EXEC((CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN (DO)
	ENDDO

continued on next page

System/36 Condition	CL Equivalent

DATAT-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKTAP DEV(TAPnn) VOL(*MOUNTED) SEQNBR(*SEARCH) +
	LABEL(Name)
	MONMSG MSGID(CPF0000) EXEC((CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN DO
	ENDDO

DSPLY	Function not supported
ENABLED	Function not supported

EVOKED-YES	DCL VAR(&TYPE1) TYPE(*CHAR) LEN(1)
	RTVJOBA TYPE(&TYPE1)
	IF(&TYPE1 *EQ '0') THEN(DO)
	ENDDO

EVOKED-NO	Change &TYPE1 by '1'
INQUIRY	Function not supported

JOBQ-YES	DCL VAR(&TYPE1) TYPE(*CHAR) LEN(1)
	RTVJOBA TYPE(&TYPE1)
	IF (&TYPE1 *EQ '0') THEN(DO)
	ENDDO

JOBQ-NO	Change &TYPE1 by '1'

LOAD-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKOBJ OBJ(LIB/Name) OBJTYPE(*PGM)
	MONMSG (CPF0000) EXEC(CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN(DO)
	ENDDO

MRTMAX	Function not supported

PROC-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKOBJ OBJ(LIB/Name) OBJTYPE(*PGM)
	MONMSG (CPF9801) EXEC(CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN(DO)
	ENDDO
	NOTE: Assumes procedure is a CL program object

SECURITY	Function not supported

continued on next page

System/36 Condition	CL Equivalent

SOURCE-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKOBJ OBJ(LIB/Source-file) OBJTYPE(*FILE) MBR(Name)
	MONMSG (CPF9800) EXEC(CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN(DO)
	ENDDO

SUBR-Name	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKOBJ OBJ(LIB/Name) OBJTYPE(*PGM)
	MONMSG (CPF9801) EXEC(CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN(DO)
	ENDDO
	NOTE: Assumes subroutine is a program object

SWITCH-11xxxxxx	DCL VAR(&SW02) TYPE(*GL)
	CHGVAR VAR(&SWS02) VALUE(%SWITH(11XXXXXX)
	IF (&SWS02 *EQ '1') THEN(DO)
	ENDDO

SWITCHn-0	DCL VAR(&WRKx) TYPE(*CHAR) LEN(1)
	RTVJOBA SWS(&WRKx)
	IF (&SST(&WRKx n 1) *EQ '0') THEN(DO)
	ENDDO

VOLID-NameID,S1	DCL VAR(&RTNCD) TYPE(*CHAR) LEN(1)
	CHGVAR VAR(&RTNCD) VALUE('0')
	CHKDKT LOC(*S1) *FIRST *WRAP VOL(NameID)
	MONMSG (CPF0000) EXEC(CHGVAR &RTNCD VALUE('1'))
	IF (&RTNCD *EQ '0') THEN(DO)
	ENDDO

Appendix G. Procedure Control Statements and Their Equivalents

System/36 Condition	CL Equivalent
ELSE	ELSE
CANCEL	Not supported. Use SNDPGMMSG to send escape to top-level program.
EVALUATE	CHGVAR
GOTO TAG	GOTO TAG
TAG NAME	LABEL NAME:
RESET	Not supported. TRFCTL can be used in some cases.
RETURN	RETURN
PAUSE	DCL VAR(&RPLYX) TYPE(*CHAR) LEN(1) SNDUSRMSG MSG('PAUSE-WHEN READY, ENTER RESP.') MSGRPY(&RPLY1)
// * 'message'	SNDUSRMSG MSG('message') MSGTYPE(*INFO)
// * mic	SNDUSRMSG MSGID('USR *TCAT 'mic') MSGF(usrmsg) MSGTYPE(*INFO)
// ** 'MESSAGE'	DCL VAR(&RPLY1) TYPE(*CHAR) LEN(1) SNDUSRMSG MSG('MESSAGE') TOMSGQ(QSYSOPR) MSGTYPE(*INQ) MSGRPY(&RPLY1)
// ** mic	DCL VAR(&RPLY1) TYPE(*CHAR) LEN(1) SNDUSRMSG MSGID('USR *TCAT 'mic') TOMSGQ(QSYSOPR) MSGF(usrmsg) MSGTYPE(*INQ) MSGRPY(&RPLY1)

Appendix H. List of Abbreviations

Abbreviations	Meaning
ANS	American National Standard
APPC	Advanced program-to-program communications
CL	Command language
COBOL 74	Common business-oriented language, 1974 standard
COBOL 85	Common business-oriented language, 1985 standard
CPF	Control program facility
CPI	Common programming interface
CUA	Common user access
DBCS	Double-byte character set
DDM	Distributed data management
DDS	Data description specifications
DFU	Data file utility
FRF	Field reference file
HLL	High-level language
I/O	Input/output
ICF	Intersystem communications function
IDDU	Interactive data definition utility
IPL	Initial program load
ITSC	International Technical Support Center
ITSO	International Technical Support Organization
LDA	Local data area
MDT	Modified data tag
MRI	Machine-readable information
MRT	Multiple requester terminal
MVS	Multiple virtual storage
NEP	Never-ending program
NLS	National language support
OCC	Operator control command
OCL	Operation control language
OS/2	Operating system/2
PAG	Process access group
PC	Personal computer
PDM	Programming development manager
PRPQ	Programming request for price quotation
RPG II	Report program generator 2
RPQ III	Report program generator 3
RUF	Read under format
SAA	Systems application architecture
SDA	Screen design aid
SFGR	Screen format generator routine
SNA	Systems network architecture
SNUF	SNA upline facility
SQL	Structured query language
SRT	Single requester terminal
SSP	System support program
UPSI	User program switch indicator
VM	Virtual machine
VTOC	Volume table of contents
WSU	Work station utility

Index

D

E

F

G

H

I

J

P

R

S

T

U

V

W

Special Characters

READER'S COMMENTS

Title: **Converting System/36 Environment Applications to Native AS/400**

Document Number: **GG24-3304-01**

You may use this form to communicate your comments about this publication, its organization or subject matter with the understanding that IBM amy use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Comments:

Reply Requested: ____ Yes ____ No

Name: _____

Job Title: _____

Address: _____

_____

_____

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 40 ARMONK, N Y

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM International Technical Support Center
Department 977, Building 663-3
Highway 52 and NW 37th Street
Rochester, Minnesota 55901 U.S.A.

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

GG24-3304-01

Converting System/36 Environment Applications to Native AS/400

GG24-3304-01

Printed in the U.S.A.