6-1997

Design and Development of a User Interface between MAPICS/DB and Factor 5.2

Raghu Pothuri

DESIGN AND DEVELOPMENT OF A USER INTERFACE BETWEEN MAPICS/DB AND FACTOR 5.2

by

Raghu Pothuri

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Industrial and Manufacturing Engineering

Western Michigan University
Kalamazoo, Michigan
June 1997

ACKNOWLEDGEMENTS

I extend my sincere appreciation to all of my committee members, for their support. Special thanks go to my advisor Dr. Tarun Gupta, for his continued guidance and participation, and more importantly for "not giving up on me". I would like to thank Dr. Frank Wolf, and Dr. Richard Munsterman for their review, and suggestions in preparing the final report.

I am grateful to all my family members for their continued support throughout this research. I dedicate this thesis to my wife Manjula for her motivation and support, and my beloved daughter Ramya, who was born during this research period and grew up along with it. I am very grateful to my parents who always kept me in the track which led to this accomplishment.

Finally I thank the Department of Industrial and Manufacturing Engineering at Western Michigan University for the opportunity given to me to pursue higher education in the United States of America.

Raghu Pothuri

DESIGN AND DEVELOPMENT OF A USER INTERFACE BETWEEN MAPICS/DB AND FACTOR 5.2

Raghu Pothuri, M.S.

Western Michigan University, 1997

An interface program was designed to transfer production data from MAPICS/DB, a MRP II software, to FACTOR 5.2, a Finite Capacity Scheduling software. Since FACTOR 5.2 is designed to be used in a stand-alone mode with its own independent data base, this interface program extracts data from the MRP II System and converts it to formats specified by the Scheduling System. To create a computer model of the manufacturing process, information about manufacturing orders, routings, and production facilities was obtained from MAPICS/DB files. Certain information needed by FACTOR 5.2, which is not available in MAPICS/DB, was supplemented by the program as auxiliary data. Three program modules were developed and were presented in the form of a menu to facilitate flexibility of data transfer.

There are several advantages in using this interface, which include faster what-if analysis, better management of production constraints by advance visibility of production conditions, and early preventive actions.

TABLE OF CONTENTS

Table of Contents-Continued

CHAPTER

Table of Contents-Continued

CHAPTER

Table of Contents-Continued

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Production Scheduling - Related Problems

The problem of scheduling a job shop is notoriously difficult, even for human intelligence. Scheduling a flexible manufacturing system is difficult because:

1. It is hard to compute from a cost function to a schedule.

2. The floor always deviates from whatever model one used to predict its behavior.

3. A manufacturing enterprise is so complex that algorithms take too much time to run

4. In some cases the floor is so complex that its behavior cannot be predicted.

The quality of a production schedule may involve many conflicting objectives. While maximizing throughput is certainly an important consideration, an ideal schedule will also have the following characteristics: (a) delivery dates are met; (b) inventory costs are maintained at acceptable levels; (c) equipment, personnel and other limited resources are well utilized and have balanced work loads; and (d) adaptations can be made quickly in the event of an unexpected change, equipment failure, raw material shortage, etc.

In order to deal with these problems, many companies send inventory plans directly to their MRP II systems. Although MRP II systems have planning as well as manufacturing transaction capabilities, the planning component often does not meet the complex requirements of today's production planner. MRP II systems generate material replenishment plans and provide a view of aggregate capacities, but the process of mapping demand requirements into realistic capacity constraints then becomes a time consuming, trial-and-error process. The outcome is usually a reactive manufacturing environment.

Production managers are under constant pressure to balance the trade-offs between high utilization of the facilities, work-in-process inventory levels, and shipping dates to their customers. Production planners require robust scheduling algorithms that focus on realistic capacity constraints, interactive what-if simulation, and the ability to quickly adapt to changes in demand and capacity. MRP II systems, while valuable for generating material plans and tracking product flows through manufacturing operations, fall short in this interactive, capacity-driven planning environment. MRP II is actually an oversimplification of the way the manufacturing plants actually operate. This oversimplification occurs in three ways.

First, the MRP II data base that defines the processing time of an operation is actually composed of three components: the actual processing time of the operation, the wait time or queue time to get into the operation, and the material transport time between op-

erations. This failure to separate processing times from the other delay times can lead to erroneous machine loading and poor resource utilization.

In the second oversimplification, the MRP II system does not keep track of all manufacturing resources required to produce the product. Frequently, the fixtures, operators, setup operators and material handling equipment are not included as manufacturing constraints that need to be controlled and scheduled. This failure leads to poor scheduling.

In the third, the MRP II system considers each operation in isolation and does not provide for the actual sequencing of products through the facility. So at any given operation, the floor-level personnel decide in what order to process the work through their operational area.

Finite Capacity Simulation - A Solution

We can arrive at workable solutions to these problems by simulation and analysis. Simulation replicates the functional structure of the factory in a mathematical model, mimics the behavior of the factory through time by exercising the model, observes the behavior of the model, and interprets those observations in the context of the original factory. The simulation model provides a computer replica of the department in the factory. The model plays through the schedule and provides performance information.

Finite Capacity Scheduling is one of the most powerful execution systems, which is increasingly being used in the manufacturing industry. This type of scheduling involves modeling production process and simulating the processing of each manufacturing order. Such a simulation is independent of the planned lead times used by MRP, inasmuch as the

probable actual lead time of each order is determined by whether it will be delayed by other orders at a capacity constraint or whether such delays will be avoided. In the course of simulation, expected operation start and completion times are determined. Visibility of future potential delays enables action plans to be developed far enough in advance of the undesirable situation that it can be prevented from developing.

The Finite Capacity scheduling software quickly generates schedules and presents scheduling data in an easy-to-understand format. This software also performs a what-if analysis and quickly regenerate a schedule. After performing and comparing multiple what-ifs, the scheduler should be able to compare alternative schedules and pick the one which best meets the organization's needs.

Why Systems Integration

Without integrated technology devoted to scheduling, the process of translating enterprise requirements into an executable production schedule is little more than a paper-and-pencil exercise. Manual methods, such as magnetic scheduling boards and standalone PC systems, only serve to introduce delay and confusion into the overall logistics process. Furthermore, deployment systems, which rely on timely and accurate production information to drive shipping plans, are not synchronized with manufacturing and often receive outdated schedules. All these factors wreak havoc on the distribution process, leading to significantly higher inventory costs and lower levels of customer service.

How to Implement FCS

Most of the Finite Capacity Scheduling packages available in the market have their own data input requirements in order to produce schedules and compare options, and are designed to be used in a standalone mode. The data about materials, machines and production orders already exist in the company's MRP II system. The American manufacturing industry has invested significant time, effort, and money in implementing MRP II systems, training users, and maintaining system data. All we need is some way of integrating the MRP II system with the scheduling system, by developing an application program interface, which extracts data from the MRP II system and converts it to formats specified by the scheduling system.

This study involves integrating a finite capacity scheduling system and a MRP II system, utilizing FACTOR 5.2 (Scheduling Software) and MAPICS/DB (MRP II application). Both of the software run independently on an IBM AS/400 platform. To create a computer model of the manufacturing process, information about production facilities is obtained by the interface program - the number of machines and people, their productivity, work plans and the like. From the manufacturing order file, products to be produced are obtained - order quantities, due dates, process routings, production times, and of course, the current state of progress of each order. The integration is performed in three steps:

1. Transferring manufacturing orders from MAPICS/DB to FACTOR 5.2.
2. Transferring work center information from MAPICS/DB to FACTOR 5.2.
3. Creating shift information in FACTOR 5.2.

The input-output relationship between FACTOR 5.2 and MAPICS/DB is designed to satisfy the data definitions and formats of the two applications. This interface application runs independently without affecting either of these applications. The integration process also includes customizing the screens so that the scheduler has the opportunity to transfer the specific information in which he is interested. For example, the scheduler can transfer a manufacturing order and the application automatically transfers only those resources pertaining to that order.

Objectives

The main objective in this study was to design a data interface application between MAPICS/DB and FACTOR 5.2 in order to achieve better capacity management of manufacturing resources. Other supportive objectives were as follows:

1. Understand the general operation of FACTOR 5.2 and MAPICS/DB.

2. Understand the data base relationships and file structures of FACTOR 5.2 and MAPICS/DB.

3. Find the input requirements of FACTOR 5.2 for the simulation process.

4. Investigate for the information in MAPICS/DB files which matches the input requirements for FACTOR, for example, the product information and the work center information etc.

5. Learn RPG/400, CL/400, and SQL/400.

6. Learn AS/400 utilities such as SEU and SDA for writing programs and designing screens.

7. Design user friendly screens for information input and output.

CHAPTER II

THE AS/400 ENVIRONMENT

Introduction to AS/400

Unlike other systems which require additional, costly software to provide them with database capabilities, the AS/400 was designed with database applications in mind. Its operating system automatically treats all data files as part of a large relational database system.

AS/400 Database Files

AS/400 allows us to define two types of database files: physical files and logical files. Physical files actually store data records. Logical files describe how data appears to be stored in the database. Logical files do not actually contain data records; instead, they store access paths, or pointers, to records in physical files. A logical file is always based upon one or more physical files.

A physical file contains vital information about customers, products, accounts, and so on. These files are organized into a data hierarchy of file-record-field. A file is a collection of data about a given kind of entry or object. A file, in turn, is broken down into records that contain data about one specific instance of the entity. Each record contains several discrete pieces of data about each entity instance.

Display Files

The dialogue between the user and the computer is mediated through display files. Display files define the screens that the program presents as it runs. Display files allow values keyed by the user in response to the screen to be input as data to the program. Thus, display files serve as the mechanism that allows the user and program to interact.

Display files are defined externally to the program that uses them. The procedure for creating a display file is similar to the procedure followed for creating a physical or logical file. Display files are coded on DDS specification sheets. Display files include entries at a file, record and field level, just like physical and logical file definitions.

Data Structures

Data structures can give flexibility in the handling of data in the following ways: (a) to allow the user to subdivide fields into subfields, (b) to restructure records with different field layouts, (c) to change field data types, (d) to define character fields longer than 256 bytes, and (e) to add a second dimension to arrays.

Data structures are defined on Input Specifications, following any record definitions. DS coded in positions 19-20 signals the beginning of a data structure. Data structure names follow the same rules as field names. Subfields comprising the data structure follow the data structure header line. Each subfield entry is defined by giving it a name (positions 53-58) and specifying its location within the data structure with "from" and

"to" values (positions 44-51). The locations of subfields may overlap, and the same position with a data structure may fall within the location of several subfields.

Introduction to RPG/400

IBM introduced the Report Program Generator (RPG) programming language in the early 1960's. In those days, RPG filled a niche by providing quick solutions to a common business task: generating reports needed within the business. Over a period of time, IBM made several major changes to RPG. During 1970s', several trends in data processing became apparent. Interactive applications began to mushroom which required a structured design in RPG programming. With the introduction of S/38 by IBM in 1988 came a new version of RPG, RPG/400. RPG/400 is a minor upgrade of RPG III, with new operations and enhancements.

Designing a RPG/400 Program

Designing a program includes:

1. Deciding what output you need from your program.
2. Deciding what processing will produce the output you need.
3. Deciding what input is required by and available to your program.

Data Files and the Data Hierarchy

A file is a collection of data about a given kind of entity or object. A file is broken down into records that contain data about one specific instance of the entity. Each record

contains several discrete pieces of data about each entity, called fields. A field generally represents the smallest unit of data that we want to manipulate within a program. All records within a file usually contain the same fields of data. A file occasionally may contain different record types, each with its own distinct format. In this case, each record usually contains a code field whose value signals which format that record represents.

Specification Forms in RPG

RPG programs consist of different kinds of lines, called specifications. Each type of specification has a particular purpose. File description specifications are used to identify the files the program is supposed to use. Calculation specifications are used to detail the arithmetic operations to be performed by the program. Output specifications provide details about the output required.

Since this study is focused mainly on the use of RPG in database link applications, we will discuss more about the functions in RPG that are commonly used for database file access and record manipulation.

Interactive Applications

Interactive applications are user-driven applications. As the program runs, a user at a workstation interacts with the computer selecting options from menus, entering data, responding to prompts, and so on. The sequence of instructions the program executes is determined in part by the user. The program continues until the user signals he is ready to quit.

Interprogram Communications

Interprogram communication is the crux of this study. As concern about program development efficiency has grown, programmers have become increasingly interested in developing small, stand-alone units of code (rather than writing monolithic programs of thousands of lines). This is called the modular programming approach. These small, self-contained modules of code can be connected by several functions as described below.

CALL Operation

The CALL operation passes control to the program named in Factor 2. Factor 2 may contain a literal specifying the program to be executed (the "called program"). Alternately, Factor 2 may contain a field, array element, or named constant that specifies the name of the program to be executed. When the program name is determined through a variable value, the program to be called is not fixed or constant, but may change from one call to the next.

When program execution reaches a CALL statement, control passes to the called program, which in turn begins to execute. The called program continues to execute until it reaches a RETRN statement. At this point, control returns to the calling program at the statement immediately following the CALL.

Passing Data Between Programs

A CALL operation would be of limited value if it did not permit the called and calling programs to share data. Within a single RPG program, all variables are globally defined; that is, the value of any variable can be accessed from anywhere within the program. However this global feature of variables does not extend across program boundaries.

RPG uses PARM (Identify Parameters) operation to indicate which field's values are to be shared between programs. A list of PARMs in the calling program must have a list of corresponding PARMs in the called program. Although the data names of the calling and called programs' PARMs do not need to be the same, corresponding PARMs in the two programs should have the same type and length since, in fact, these corresponding parameters are referencing the same storage location within the computer.

PARMs can appear only immediately after a CALL operation or following a PLIST operation. PLIST (Identify a Parameter List) is a declarative operation that identifies a list of parameters to be shared between programs. PLIST requires an identifying entry in Factor 1. That entry may be a PLIST name if the PLIST is within a calling program, or the reserved word *ENTRY if the PLIST is within a called program and signals the arguments the called program is to receive from the calling program upon its invocation.

Arithmetic Operations in RPG

RPG does not include a wealth of mathematical operations. The four basic arithmetic operations--add, subtract, multiply, and divide--with a few additional extras, represent the range of RPG's mathematical offerings.

ADD Operation

The ADD operation is used to add the value of two numbers and store the result in a numeric field. Factor1 (positions 18-27) and factor2 (positions 33-42) contain the values to be added. These values may be represented as either numeric fields or numeric literals.

Example:

```
REGPAY      ADD   OTPAY       TOTPAY      6 2
```

SUB Operation

The SUB operation is used to subtract factor2 from factor1. The result of the subtraction is stored in the result field. As with addition, factor1 and factor2 can be fields or numeric literals, while the result must be a numeric field.

Example:

```
GROSS       SUB   WITHLD      NETPAY      6 2
```

MULT Operation

The MULT operation is used to multiply the contents of factor1 and factor2 and store the answer in the result field. Numeric fields and/or literals can serve as multipliers, while the result must be stored in a numeric field.

Example:

```
SALES MULT        TAXRAT      SLSTAX      5 2
```

DIV Operation

The DIV(Divide) operation is used to divide factor1 by factor2 and the answer is stored in the result field.

Example:

```
TOTAMT     DIV    CNT    AVGAMT      6 2
```

Assignment Operations

Assignment operations allow the user to assign a value to a variable. RPG has four assignment operations, two used for numeric fields and two that are used most often with character fields.

Z-ADD (Zero and Add) Operation

The Z-ADD operation can be interpreted as "zero out the result field and add factor2 to it." The effect of this operation is to assign the value of factor 2 to the result

field. The most common use of this operation is to initialize or reinitialize a counter or accumulator to zero. Z-ADD operation always involves a factor 2 value and a result field.

Example:

```
Z-ADD       20      MAX  2 0
```

Z-SUB (Zero and Subtract) Operation

The Z-SUB works similar to Z-ADD, except that after zeroing out the result field, it subtracts the value of factor2 from the result field. Because this operation assigns the negative value of factor2 to the result field, its effect is to reverse the sign of a field.

Example:

```
Z-SUB       20      MIN  2 0
```

MOVE(Move) Operation

The primary use of the MOVE operation is to assign a value specified in factor 2 to a character result field. Factor 1 is not used with MOVE. The MOVE operation transfers characters from the sending field in factor 2 to the receiving field in the result, character by character, moving it through the fields from right to left.

Example:

```
MOVE        'ABCD'      EXAMPLE   4
```

MOVEL (Move Left) Operation

The MOVEL operation, which requires a factor2 and a result entry, works like a MOVE except that data transfer starts with the left-most characters of the sending and receiving fields and moves data, character by character, from left to right

Example:

```
MOVEL       'ABCD'EXAMPLE    4
```

Figurative Constants

RPG includes a special set of reserved words called figurative constants. Figurative constants are implied literals that can be used without a specified length. Figurative constants assume the length and decimal positions of the fields they are associated with. RPG's figurative constants are *BLANK(or *BLANKS), *ZERO(or *ZEROS), *HIVAL, *LOVAL, *OFF, *ON, and *ALL'X..'.

Moving *BLANK or *BLANKS causes a character field to be filled with blanks. Moving *HIVAL fills a character field with X'FFF..'(all bits on) and numeric field with all 9's and a negative sign. Moving *ZERO to a numeric or character field fills the field with 0's. Figurative constants *OFF, *ON represent character '0' and character '1', respectively.

CHAPTER III

FACTOR 5.2 MODELING

Introduction to FACTOR 5.2 Modeling

A computer simulation model is a mathematical and logical representation of the dynamic characteristics of a physical system. The purpose of FACTOR is to provide a format for creating and simulating a model of the manufacturing production system. Ultimately, FACTOR 5.2 will provide detailed schedules of the production system. Before a final production schedule is accepted, however, FACTOR 5.2 allows for experimentation with the model. This experimentation might be directed to reach such goals as increased system productivity, on time completion of orders, and higher resource utilization. Experimentation with the model can be used to understand the effects of unexpected events such as resource failures, hot orders, and material shortages. Therefore, FACTOR 5.2 predicts the behavior of the production system.

Modeling Components

A modeling component is a representation of a physical component of the production system. For example, the order modeling component is a representation of actual production orders. Technically, a FACTOR component is a type of record in

the FACTOR database. To represent a specific production order, a database record is created which contains information about the order.

The FACTOR 5.2 Database

A FACTOR 5.2 database contains the following elements:

1. Model component data describing one or more FACTOR models.
2. Results of simulating a FACTOR model.
3. Data generated by and used by FACTOR.
4. Input constructs used by FACTOR's capacity planning programs.
5. Input/output constructs used by FACTOR Output Analysis.

Populating FACTOR Database

FACTOR needs data to run the simulation and produce production schedules. Supplying data to FACTOR database is commonly referred to as "Populating FACTOR database." A complete understanding of the FACTOR database definition is required to successfully populate the FACTOR database. Auxiliary data must also be taken into account. Auxiliary data is data not provided in the standard FACTOR product but is added by the user to accurately model the production facility. The standard FACTOR database definition, coupled with the definition of the auxiliary data, completely defines the FACTOR database.

In our study, most of this data is transferred from MAPICS/DB files. To begin with, the data required by FACTOR should be identified in MAPICS/DB. Then an appropriate program should be developed to transfer the data into FACTOR database. Also, the input data should be mapped to the FACTOR database correctly and any data that is not available in the MRP system should be provided by the program as required by FACTOR. For example, the resource action code field on the job step record is to be entered by the program, because FACTOR does not provide an initial setting for this field, nor is it available in MAPICS/DB.

Alternative

The term "alternative" highlights the feature of FACTOR which allows alternate views of manufacturing, strategies for order release, and philosophies of scheduling, to be tested prior to the distribution of actual worklists to the shop floor. In technical terms, an alternative in FACTOR is a set of input data to a simulation application. The input data describes the manufacturing operations to be scheduled and, therefore, includes various modeling components such as orders, parts, process plans, and resources.

FACTOR supports the generation of multiple alternatives. As many as upto 1,000 alternatives are possible in a single database. Throughout this study, "XXX" will be used to indicate a generic three-digit alternative identifier. Every alternative specifies three sets of information--controls, input dataset references, and alternative date-time stamps. Controls direct the FACTOR simulator during the execution of the simulation and include information such as the simulation window (i.e., the simulation's start and end date), debug

trace information, scheduling window, simulation rules, efficiency factor, and output data collection flags. Input dataset names refer to the datasets which will be used as input during the simulation run. This input data describes the manufacturing plans, and resources, to name a few.

Orders

Orders in FACTOR represent the authorization to perform tasks using the resources of the manufacturing system. Typically, a FACTOR order represents a customer order or internal shop order for a specific quantity of a particular part. In FACTOR, the order information is stored in ORDERXXX file. In MAPICS/DB, the order particulars are stored in MOMAST file. Please refer to XXX represents all the fields in ORDERXXX file, with their parent fields in MOMAST file. ORDERXXX contains the following fields:

ORDID (Order ID)

Specifies an identifier used for the order. This is an alphanumeric value. The order ID uniquely identifies each order.

DESCR (Order Description)

Specifies a description of the order. This is an alphanumeric value.

PARTID (Part ID)

Specifies the part to be produced. This is an alphanumeric value

PPID (Process Plan ID)

Specifies the processing plan to use for routing this order through the manufacturing process

ORDSIZ (Order Size)

Specifies the quantity of the item to be produced. This is an integer number between zero and 32767. The number of loads that you enter into the simulation for an order is the order size divided by the initial load size. Remainder parts are placed in a load based on the excess code.

STATCD (Order Status)

Specifies the initial status of orders. Possible values for this field are:

N(New order). If you enter 'N', the order is new, and production will start on the release date.

I(In-process order). If you enter 'I', the order is in-process, which means that it is already on the floor in a partial state of completion. In-process loads will have a corresponding load status record.

U(Unconfirmed order). If you enter 'U' for the order status, the order is unconfirmed. Unconfirmed orders are orders that have been verified and may be ignored if necessary. There is a field on the alternative record which can be used to ignore unconfirmed orders.

X(Explicit-release order). If you enter 'X', for the order status, the order will be an explicit-release order which is used in conjunction with job step (the release job step). A single explicit-release order can be released several times using this job step.

LDSIZ (Initial Load Size)

Specifies the size of the transfer load or batch. This is an integer number between zero and 32747. The load size is the quantity of parts that travel together as a single entity through the production process. If more than one part is in a load, the first part of the load must wait until the last part of the load has finished the operation before the load can advance to the subsequent operation. The number of loads that you can enter into the simulation for an order is the order size divided by the initial load size. Remainder parts are placed in a load based on the excess code.

PRIO (Order Priority)

Specifies the relative priority of this order. This is an integer number between 0 and 32767. The order priority is used to give higher priority to some orders relative to others. The order priority can be used to sequence loads in a queue.

EXCSCD (Excess Code)

Specifies how to place excess parts in a load. Possible values are:

A(Add parts to the last load). If you specify 'A' for the excess code, then any remaining parts are placed in the last load of the order.

N(Form a new load with just the excess). If you specify 'N', the remaining parts are placed in a new load that is the size of the excess

W (Form a new load of the load size). If you specify 'W', a new load that is the size of the initial load size is created

SCHDFG (Schedule)

Specifies whether to collect schedule data during simulation for this order. Possible values are a)Y(Yes), b) N(No)

RELDAT (Release Date)

Specifies the simulation date that the order can be started into production.

RELTIM (Release Time)

Specifies the simulation time that the order can be started into production.

DUEDAT (Due Date)

Specifies the simulation date the order is due to be completed.

DUETIM (Due Time)

Specifies the simulation time the order is due to be completed.

Parts

The part component specifies the characteristics of the part being manufactured. In FACTOR, part information is stored in PARTXXX file. The part information is retrieved from MOMAST file, in MAPICS/DB. The part fields are described in detail below.

PARTID

Specifies the part ID for a particular production item. This is an alphanumeric value

DESCR

Specifies a description of the part. This is an alphanumeric value

FAMILY

Specifies the description of the family of which this part is a member. This is an alphanumeric value. The part family can be used for the setup time lookup table or to describe relationships to other parts for setup time

SUBFAMILY

Specifies the description of the subfamily of which this part is a member. This is an alphanumeric value. The part subfamily can be used for the setup time lookup table or to describe relationships to other parts for setup time

PROCPLANID

Specifies the process plan to be used to route orders for this particular part. This is also an alphanumeric value. If defaulted, the process plan ID on the part's order will be used.

STARTMATLID

Specifies the ID of the material inventory required for this particular part. This is an alphanumeric value. This value must be a valid material ID. The starting material ID can be used to specify the material to remove when a remove-from-material job step defaults the material ID. No material is removed without an appropriate job step.

ENDMATLID

Specifies the ID of the material inventory storage area for this part when completed. This is an alphanumeric value. This value must be a valid material ID. The ending material ID can be used to specify the material to add when an add-to-material job step defaults the material ID. No material is added without an appropriate job step.

PTTABLE

Specifies the ID of a part-based lookup table. This is an alphanumeric value. This value must be a valid lookup table ID. This lookup table is used when a job step uses step time rules 2 or 3. The lookup table is organized with the process plan ID as the first index and the job step ID as the second index.

Process Plans

A process plan specifies the sequence of operations which must be performed on a part(or a load of parts) in order to produce an end item. In FACTOR, each operation is defined in terms of one or more job steps. A job step describes an activity or action in the manufacturing process. A job step might require particular resources, such as machine or an operator, before the operation can be performed on the part. The process plan information is stored in file JOB STEPXXX. The values for JOB STEPXXX are retrieved from file MOROUT in MAPICS/DB. JOB STEPXXX contains the following fields:

PROCPLANID

Specifies the identifier of the process plan to which the job step belongs. This is an alphanumeric value.

TYPE

Specifies the type of this job step. Possible codes and descriptions are presented in Table 1.

JSID

Specifies the ID used to identify the job step. This is an alphanumeric value. Typically this will be based on an operation number. A job step defines a step in the processing sequence associated with a load of parts.

DESCR

Specifies a description of the job step. This is an alphanumeric value.

NEXTJSID

Specified the ID of the next job step to process after completion of this job step. If blank, then this job step is the last job step of the process plan. The modeler may define a job step or a series of job steps which are not referenced as the "next" job step by any other

job step. This allows the modeler to predefine alternate job step routings which may then be used at later executions.

Table 1

Job Step Codes and Descriptions

Type	Description
1	Operation
2	Assemble
3	Produce
4.	Setup
5	Move
6	Batch
7	Move between
8	Add-to-material
9	Remove-from-material
10	Select
11	Accumulate/Split
12	Change-Load-size
13	Setup/Operation
14	Release
15	Inspect
16-39	User Installable

SELECTRL

Specifies the conditions which must be true to execute this job step. This is an integer number from 0 to 39. Possible values are: (a) 0 - Always execute this job step; (b) 1 - If the first resource listed is available, then execute this job step—the resource must have a hold or step action code; and (c) 2-39 (User installable).

ALTJSID

Specifies the ID of an alternate job step to consider if the job step selection rule fails. This is an alphanumeric value. This value must be a valid job step ID. When there is more than one alternate job step from which to select, the select job step should be used. It provides additional capability in this regard. If no alternative job step is specified, the load will always perform the next job step

ALOCRL

Specifies the procedure to use for allocating (as applicable) resources, resource groups, and pools on this job step. This is an integer number between 0 and 39. Possible values are:

1. 0 - Allocate all resources, resource groups, and pools in any order as they become available.

2. 1 - Allocate all resources, resource groups, and pools in the order specified as they become available.

3. 2 - Allocate all resources, resource groups, and pools all at once when all are available.

4. 3 - Allocate all resources, resource groups, and pools and materials at once when all are available. This rule is only applicable to job step types involving materials namely types 2, 3, 8, and 9.

5. 4 to 39 - User installable.

STEPTMRL

Specifies how the step time is to be used for calculating the total duration of the job step. This is an integer from 0 to 39. Possible values are:

1. 0 - Duration is the step time on the job step.

2. 1 - Duration is the step time on the job step multiplied by the number of parts in the load.

3. 2 - Duration is from the entry in the part-based lookup table. the table entry is found by using the process plan ID and job step ID as indices.

4. 3 - Duration is from the entry in the part-based lookup table multiplied by the number of parts in the load. The table entry is found by using process plan ID and job step ID as indices.

5. 4 - Duration is from the entry in the setup lookup table. The table entry is found by using the information in the job step variant to determine the indices. The when-to-

setup rule is also used to determine if setup is necessary. This rule may only be used with job step types 4 and 1-3.

6. 6 - Duration is from the entry in the move-between lookup table. The table entry is found by using the origin and the destination as indices. This rule may only be used with job step type 7.

7. 7-39 - User installable.

STEPTIME

Specifies the time base to be used in calculating the duration of the job step (in hours). This is a floating point number greater than or equal to zero.

FREECHCKFG

Specifies whether to check and provide an error if an attempt is made to free a resource that is not held by the load. Possible values are:

'Y' Yes

'N' No

HOLDTEMPFG

Specifies whether temporary resources are to be held during the off shift overriding the temporary resource designation in the resource record. Possible values are: (a) 'Y', and (b) 'N'.

RESSCHDFG

Specifies whether to collect resource schedule data for this job step during the simulation run. Possible values are: (a) 'Y', and (b) 'N'.

RESACTN1

Specifies the type of action to take for the resource, resource group, or pool specified in the ID field (for this action). Possible values are:

1. 'A' Frees held resource after allocation.
2. 'B' Frees held resource before allocation.
3. 'C' Frees held resource at the end of job step.
4. 'H' Allocates and holds resource until freed.
5. 'S' Allocates and frees resource in same job step.
6. Blank No allocation or free occurs.

Actions determine whether a resource, resource group, or pool should be allocated or freed and when they should be freed. There are two types of allocation actions: step (S) and hold (H). For the step action, the resource begins and is freed automatically at the end of the same job step. For the hold action, the resource is allocated before the job step begins and is held until freed explicitly by a subsequent resource free action in the same or a subsequent job step. There are three types of free actions: before (B), after (A), and end (E). Free actions are used to free resources that have been allocated with a hold allocation action. For the before action, the resource is freed before the allocation of resources. For

the after action, the resource is freed at the time at which allocation is complete and the job step begins processing. For the end action, the resource is freed at the end of the current step.

RESNMBR

Specifies the number of units of the resource, resource group, or pool to allocate or free. For resources this field is always one. This is an integer number greater than or equal to zero.

RESID

Specifies the identifier of a resource, resource group, or pool to allocate or free. This is an alphanumeric value. This value must be a valid resource, resource group, or pool ID.

Setup/Operation Job Step

The setup/operation job step is stored in file JS13VRXXX. This record represents the operation process time on the load in addition to the setup time. The setup/operation job step is included because data from external production systems often represents setup and processing as a single database record. It will also allow the job step information to be presented to the shop-floor scheduler in a manner more similar to the MRP operations. The JS13VRXXX file contains the following fields.

RGID

Specifies the alphanumeric name of the resource or resource group to be set up.

WHENRL

Specifies the when-to-setup rule. Possible values are: (a) 0 - Always setup; (b) 1 - Setup based on the part/family/subfamily, start, & length fields; (c) 2 - Setup if part or job step is different; and (d) 3 to 39 - User installable.

BASEDCD

Specifies how to setup. Possible values are: (a) 'F' - Setup is based on a change in part family, (b) 'P' - Setup is based on a change in part ID, and (c) 'S' - Setup is based on a change in subfamily.

START

Specifies the starting character of the part ID for the part-based comparison. This is an integer number from 1 to 40.

LENGTH

Specifies the number of characters to compare. This is an integer number from 1 to 40.

TABLEID

Specifies the ID of the lookup table to use. This is an alphanumeric value. This value must be a valid lookup table ID. The lookup table may be defined to describe the setup time relationship between part types, part families, or part subfamilies.

RSETUPID

Specifies the alphanumeric name of the resource group, or pool required for setup. Allocation is performed only if setup is required. For example, if an operator is needed to perform the setup, the resource, resource group, or pool for the operator is specified.

STEPTMRL

Specifies a numeric code for interpreting the step time.

STEPTIME

Specifies a numeric value indicating the time base value required to set up the resource for the job step.

Resource Groups

The resource group feature allows resources which can perform the same function to be classified as a group without having the member resources lose their individual identity. A resource group is used in cases where a load on a job step needs a resource

from a group of resources, but it does not need a specific resource in the group. The resource group information is stored in file RESGRP000. RESGRP000 contains the following fields:

RESID

Specifies the ID used to identify the resource. This is an alphanumeric value.

RESTYPE

Specifies the type of the resource. This is used to categorize resources for output reporting purposes. This is an alphanumeric value.

DESCR

Specifies a description of the resource. This is an alphanumeric value.

SELRL

Specifies the procedure for selecting from the waiting queue. This is an integer number from 0 to 39. Possible values are:

1. 0 - No selection rule, use sequencing rule.

2.1 - Use the sequencing rule dynamically; re-sequence the queue using sequencing rule before each selection.

3. 2 - Select the request with minimum setup on remaining job steps.

4. 3 - Select the request from the same order. If no more requests from the order are in the queue, re-sequence using sequencing rule.

5. 4 to 10 - User installable; used by AIM.

6 11 - Select request with highest load priority. If tied, select request with minimum critical dynamic slack--dynamic slack is critical if it is less than the threshold on the alternative. If tied, select based on minimum downstream setup.

7. 12 - User installable.

MUSTCOMPFG

Specifies whether the job step must complete prior to the end of the current shift interval. Possible values are Y(yes) and N(no). If yes is chosen and job step will not end prior to the end of the current shift interval, then it will not be started.

SEQRL

Specifies the sequencing rule by which this resource's request queue is ranked. This is an integer number from 0 to 39. Possible values are: (a) 1 - FIFO (First to arrive at the job step), (b) 2 - LIFO (Last to arrive at the job step), (c) 3 - High to low load priority, (d) 4 - Low to high load priority, (e) 5 - Earliest order due date, (f) 6 - Earliest order release date, (g) 7 - Shortest time for the current job step, (h) 8 - Longest time for the current job step, (i) 9 - Longest time for any subsequent job step, (j) 10 - Least number of remaining job steps, (k) 11 - Lease estimated remaining processing time, (l) 12 - Least static slack (remaining time to due date), (m) 13 - Least average static slack over remaining

job steps, (n) 14 - Least average static slack over remaining processing time, (o) 15 - Least dynamic slack (remaining time to due date less the remaining processing time), (p) 16 - Least average dynamic slack over remaining job steps, (q) 17 - Least average dynamic slack over remaining processing time, (r) 18 to 22 - User installable; used by AIM, (s) 23 - Due date - remaining processing time, and (t) 24 to 39 - User installable description.

A sequencing rule of '0' means the global sequencing rule specified on the alternative record will be used.

MAXORUN

Specifies the maximum overrun (in hours) allowed for the resource, past the end of the current shift interval. This is a floating point number greater than zero.

ALLOCCD

Specified the allocation type of the resource. Possible values are: a) 'P'(Permanent) and b) 'T' (Temporary). If a load is holding a permanent resource that goes off shift, the load will be the first one to reallocate it when the resource is on shift again.

SFTID

Specifies one or more shift schedules to use for the resource. This is an alphanumeric value. Each value must be a valid shift ID. If no shift schedules are specified, the resource is assumed to be available for production at all times.

SUMFG

Specifies whether to collect resource summary data during the simulation for this resource. Possible values are: (a) Y(yes), and (b) N(no). Resource summary data includes information such as capacity statistics, load statistics, time in states, queue length statistics, time in queue statistics, etc.

SCHEDFG

Specifies whether to collect resource schedule data during the simulation for this resource. The possible values are Y(yes) and N(no). Resource schedule data includes information on each load that allocates and frees a resource, the date and time of the allocate and free, the job step in which the allocate or free occurred, etc.

FINALQFG

Specifies whether to collect data on the contents of this resource's queue at the end of simulation. Possible values are Y(yes) and N(no). Resource queue data includes information on each load that is in the queue.

LOADFG

Specifies whether to collect data on the loading of this resource during the simulation. Possible values are Y(yes) and N(no). Resource load data includes

information such as the standard hours, planned hours, load hours, backlog, time in different states, etc.

CHAPTER IV

MAPICS/DB & FACTOR 5.2 INTERFACE DESIGN

The Scheduler's Daily Use of FACTOR

The scheduler in a manufacturing industry is charged with the regular use of Factor 5.2 for the generation of production schedules. The integration of MAPICS/DB and Factor 5.2 would enable the scheduler to use simulation on an event-driven basis to analyze the impact of unforeseen circumstances on the shop floor, and to assess the effectiveness of strategies formulated in reaction to these circumstances.

The event-driven nature of this integrated package required developing options in the form of a menu. Three menu choices were generated based on the factor which limits the transfer of records from MAPICS/DB to FACTOR. The limiting factor may be an order number or a resource.

Structure of the Integrated Package

The main menu of the package consists of the following six items: (a) 1 - Transfer a single manufacturing order, (b) 2 - Transfer a work center, (c) 3 - Transfer the jobshop, (d) 4 - Use MAPICS/DB, (e) 5 - Use FACTOR 5.2, and (f) 90 - Sign Off.

This menu is compiled in AS/400 using "Screen Design Aid" (SDA). The source code for this menu is stored in a physical file named LRNMNUSRC. The actual menu is named TRANSFER and is placed in TRANSFER library. The menu is called using command "GO TRANSFER".

Transfer Based on a Single Manufacturing Order

Menu item (1) is used if the user is interested in the performance of a particular manufacturing order. A flow chart for this procedure is illustrated in Figure 1. Selecting this option activates a command named NEWFINTORD. NEWFINTORD is a CL command. The source code for this command is saved in member NEWFINTORD in the physical file named RCMDSRC.

NEWFINTORD Procedure

```
CMD         PROMPT('TRANSFER BASED ON MFG.ORDERS')
PARM        KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999) MIN(1)
                FULL(*YES) PROMPT(Alternative number)
PARM        KWD(ORDNO) TYPE(*CHAR) LEN(7) MON(7) FULL(*YES)
                PROMPT('MFG.ORDER NO')
```

The NEWFINTORD command is connected to a CL program named NEWINTORD. The purpose of the command is to collect the altern number and the manufacturing order number and pass them to the program.

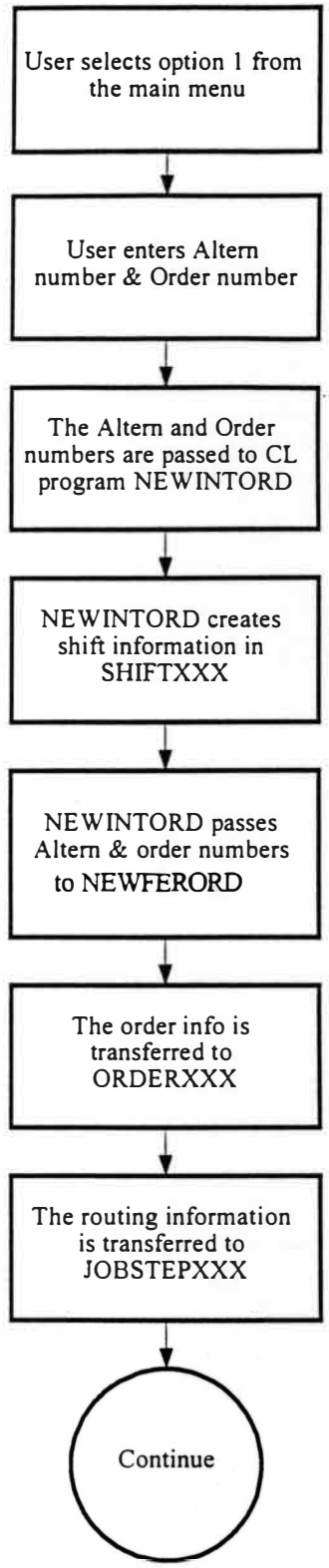

Figure 1. Flow Chart for Order Based Transfer.

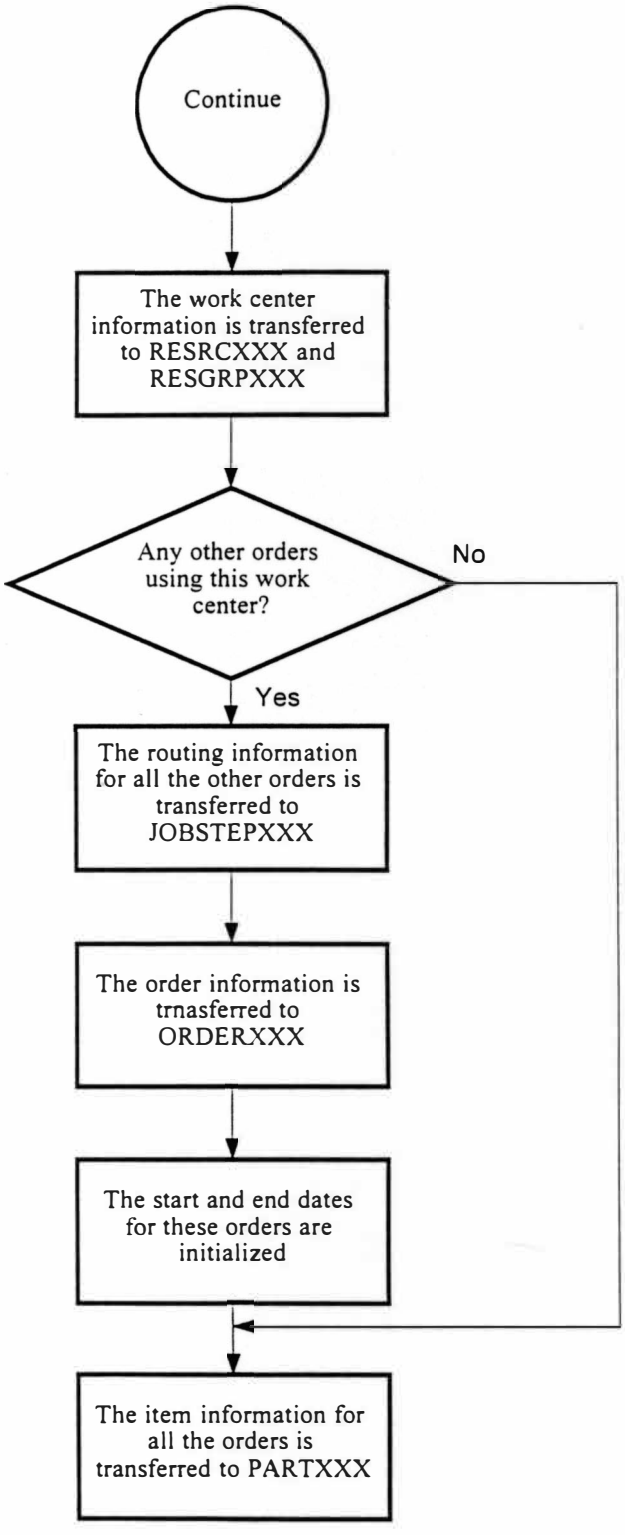

Figure 1.- Continued

NEWINTORD Procedure

First, the altern number and the manufacturing order number are placed into variables &CHALT, and &ORDER.

Then the variables are declared.

```
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&ORDER) TYPE(*CHAR) LEN(7)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(10)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)
```

The following CL statements monitor for message IDs and routes the program to the appropriate location.

```
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&ORDER) TYPE(*CHAR) LEN(7)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(10)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)
```

The shifts for a particular alternative in FACTOR are created, by copying the shift information from DFTSHIFT file to SHIFTXXX file where 'XXX' denotes altern number. The following CL statement sends the word "SHIFT" and the altern number, "XXX" to a program which pastes "SHIFT" and "XXX", and sends it back to our program as variable &OFILE.

```
CALL  PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF        FROMFILE(F52MFINT/DFTSHIFT)        TOFILE(*LIBL/&OFILE)
MBROPT(*REPLACE)
```

This procedure is repeated for all FACTOR files as shown below.

```
CALL        PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF      FILE(JOBSTEP000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
```

```
OVRDBF      FILE(JS13VR000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
OVRDBF      FILE(ORDER000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF      FILE(PART000) TOFILE(&OFILE)
```

The "Override with Data Base File (OVRDBF)" command is used to override a generic file with the appropriate &OFILE whenever the generic file is used. For example JOBSTEP000 is a generic name given to the job step file in the program, and it will be overridden by JOBSTEPXXX file whenever JOBSTEP000 is used in the program.

The next step is to transfer the variables to the main RPG/SQL program named NEWFERORD.

```
CALL        PGM(TRANSFER/NEWFERORD)
PARM        (&CHALT &ORDER)
```

The program NEWFERORD is the main program that transfers the data from MAPCIS/DB files to FACTOR 5.2 files. The source code for this program is saved in member TEMFERORD in file RINTSRC.

NEWFERORD Procedure

NEWFERORD is a file member which contains RPG code with embedded SQL statements. The SQL statements perform the actual data retrieval and insertion, whereas the RPG statements are used to tailor MAPICS/DB data to suit FACTOR 5.2 file structure.

The program begins with declaring three data structures for program variables. These data structures are named DBFLDS, RESORS, and RESRC. Then the parameters passed to this program (&CHALT, and &ORDER) are saved in program variables ALT, and ORDR.

```
*ENTRY      PLIST
            PARM ALT   3
            PARM ORDR  7
```

All of the FACTOR 5.2 database files are emptied of any previous data before the actual transfer. This is accomplished using the following SQL statements.

```
DELETE      FROM      ORDER000
DELETE      FROM      PART000
DELETE      FROM      JS13VR000
DELETE      FROM      JOBSTEP000
DELETE      FROM      RESRC000
DELETE      FROM      RESGRP000
```

A cursor is then declared for retrieving Order information.

```
DECLARE ORDCUR CURSOR FOR.
```

The following fields are selected from file MOMAST (Manufacturing Order Master File).

```
SELECT ORDNO, JOBNO, FITEM, ORQTY, SSTDT, ODUDT
```

A conditional statement is used to retrieve only the information pertaining to the Manufacturing Order of our interest.

```
WHERE MOMAST.ORDNO = :ORDR
```

The following fields in file ORDER 000 are then initialized to appropriate values.

```
MOVE        'A'       EXCSCD     1
MOVE        '0000'    RELTIM     4
```

```
MOVE      '2359'      DUETIM    4
MOVE      'N'         STATCD    1
MOVE      'Y'         SCHDFG    1
Z-ADD     *ZERO       PRIO      5 0
```

The order cursor is then opened.

```
OPEN ORDCUR
```

The values in the ORDCUR are fetched into program variables.

```
FETCH ORDCUR INTO :ORDID, :PARTID, :ORDSIZ, :RELDAT, :DUEDAT
```

The order size is rounded.

```
ADD 0.5 ORDSIZ
Z-ADD ORDSIZ RORDSIZ 60
```

The dates are moved to character strings.

```
MOVE RELDAT CRLDAT 6
MOVE DUEDAT CDUDAT 6
```

The values are then inserted into ORDER000 file.

```
INSERT INTO ORDER000
(ORDERID, DESCR, PARTID, PROCPLANID, ORDSIZE, LOADSIZE, EXCESSCD,
RELDATE, RELTIME, DUEDATE, DUETIME, STATUSCD, PRIORITY,
SCHEDFG)
VALUES
(:ORDID, :ODESCR, :PARTID, :ORDID, :RORDSZ, :RORDSZ, :EXCSCD,
:CRLDAT, :RELTIM, :CDUEDT, :DUETIM, :STATCD, :PRIO, :SCHDFG)
```

Updating ALTERN table

The ALTERN table contains information about different alternatives that exist in the database. The simulating start date, start time, end date and end time are initiated to CRLDAT, RELTIM ('0000'), CDUEDT and DUETIM ('2359'). This will limit the simulation period to the processing period of the order of our interest.

```
UPDATE ALTERN
SET STARTDATE = :CRLDAT, STARTTIME = '0000', ENDDATE = :CDUEDT,
ENDTIME = '2359'
WHERE ALTNO = :ALTNUM
```

Retrieving Process Plan Information

A cursor is declared to retrieve process plan information.

```
DECLARE PPCUR CURSOR FOR
```

The following fields are selected from files MOROUT, MOMAST, ORDER000, and ITEMASA.

```
SELECT    MOROUT.ORDNO, OPSEQ, WKCTR, SRMHU, SSLHU, SETCS,
          CYCOP, SRLHU, TQCTD, TBCDE, PLCDE, ITEMASA.CUMSY,
          ORDER000.ORDSIZE
FROM      MOROUT, MOMAST, ORDER000, ITEMASA
```

A conditional statement is used to retrieve only the job step information pertaining to the manufacturing order of our interest.

```
WHERE     MOROUT.ORDNO = ORDER000.ORDERID and
          ORDER000.ORDERID = MOMAST.ORDERNO and
          MOMAST.FITEM = ITEMASA.ITNBR and
          MOROUT.OPSEQ >= MOMAST.OPCUR
```

The information is retrieved in the order of ORDNO, and OPSEQ.

```
ORDER BY ORDNO, OPSEQ
```

The following fields in files JOBSTEP000 and JS13VR000are then initialized to appropriate values

```
Z-ADD     13          TYPE      2 0
Z-ADD     *ZERO       SELRL     2 0
MOVE      *BLANKS     ALTJS     8
Z-ADD     2           ALOCRL    2 0
```

```
Z-ADD     1          STEPRL    2 0
MOVE      'Y'        FREECK    1
MOVE      'N'        HOLDTM    1
MOVE      'Y'        RESCHD    1
MOVE      'S'        RACT1     1
Z-ADD     1          RNBR1     4 0
MOVE      *BLANK     ACTN      1
Z-ADD     *ZERO      NMBR      4 0
MOVE      *BLANKS    RESID     8
Z-ADD     1          WHENRL    2 0
MOVE      'P'        BASECD    1
Z-ADD     1          START     2 0
Z-ADD     40         LENGTH    2 0
MOVE      *BLANKS    TABID     8
MOVE      *BLANKS    RSETUP    8
Z-ADD     5          STRL13    2 0
```

The process plan cursor is then opened.

```
OPEN PPCUR
```

The values in the PPCUR are fetched into program variables.

```
FETCH PPCUR INTO :PROCPN, :JSID, :PPDESC, :SRMHU, :SSLHU,
SETCS, :CYCOP, :SRLHU, :TQCTD, :TBCDE, :PLCDE, :CUMSY, :ORDSIZ
```

A DO loop is defined for processing all records until the end-of-file (EOF) condition reaches.

```
SQLCOD DOWEQ*ZERO
```

The following SQL statements are used to determine the next job step.

```
SELECT MIN(OPSEQ)
INTO :NEXTJS
FROM MOROUT
WHERE ORDNO = :PROCPN AND OPSEQ > :JSID
SQLCOD IFEQ * ZERO
MOVE *BLANKS NEXTJS
ENDIF
```

The expected operation quantity is calculated based on the Cumulative Yield of the part (CUMSY).

```
CUMSY     IFEQ      *ZERO
Z-ADD     *ZERO     EOP
ELSE
CYCOP     DIV       CUMSY     X1
MULT      ORDSIZ    EOP
ENDIF
```

The setup labor time is calculated based on setup crew size(SETCS) and setup labor hours(SSLHU).

```
SETCS     IFEQ      *ZERO
Z-ADD     *ZERO     SETUTM
ELSE
SSLHU     DIV       SETCSX1
ADD       .0005     X1
Z-ADD     X1        SETUTM
ENDIF
```

The run labor hours(RUNLTU) and run machine hours(MACHTU) are calculated based on time basis code (TBCDE).

```
SELEC
```

When time basis code is 'blank' (hours/unit), run labor time (RUNTLU) and machine(MACHTU) time are calculated as follows:

```
TBCDE WHEQ *BLANK
EOP   MULT      SRLHU     X1
ADD             0005      X1
Z-ADD           X1        RUNTLU

EOP   MULT      SRMHU     X1
ADD       .0005       X1
Z-ADD     X1          MACHTU
```

When time basis code is '1'(hours/10 units), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
TBCDE     WHEQ      '1'
EOP       DIV       10        X1
X1        MULT      SRLHU     X2
ADD       .0005     X2
Z-ADD     X2        RUNTLU

X1   MULT      SRMHU     X2
ADD       .0005     X2
Z-ADD     X2        MACHTU
```

When time basis code is '2'(hours/100 units), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
TBCDE     WHEQ      '2'
EOP       DIV       100       X1
X1        MULT      SRLHU     X2
ADD       .0005     X2
Z-ADD     X2        RUNTLU

X1        MULT      SRMHU     X2
ADD       .0005     X2
Z-ADD     X2        MACHTU
```

When time basis code is '3'(hours/1000 units), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
TBCDE     WHEQ      '3'
EOP       DIV       1000      X1
X1        MULT      SRLHU     X2
ADD       .0005     X2
Z-ADD     X2        RUNTLU

X1        MULT      SRMHU     X2
ADD       .0005     X2
Z-ADD     X2        MACHTU
```

When time basis code is '4'(hours/10000 units), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
TBCDE     WHEQ      '4'
EOP       DIV       10000     X1
X1        MULT      SRLHU     X2
ADD       .0005     X2
Z-ADD     X2        RUNTLU

X1        MULT      SRMHU     X2
ADD       .0005     X2
Z-ADD     X2        MACHTU
```

When time basis code is 'P'(pcs/hr), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
TBCDE     WHEQ      'P'
SRLHU     IFEQ      *ZERO
Z-ADD     *ZERO     RUNLTU

ELSE
EOP       DIV       SRLHU     X1
ADD       .0005     X1
Z-ADD     X1        RUNLTU
ENDIF

SRMHU     IFEQ      *ZERO
Z-ADD     *ZERO     MACHTU
ELSE
EOP       DIV       SRMHU     X1
ADD       .0005     X1
Z-ADD     X1        MACHTU
ENDIF
```

When time basis code is 'H'(hrs/lot), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
EOP   IFEQ        *ZERO
Z-ADD     *ZERO     RUNTLU
Z-ADD     *ZERO     MACHTU
```

```
ELSE

EOP         SUB         TQCTD       X1
DIV         EOP         X1
X1          MULT        SRLHU       X2
ADD         .0005       X2
Z-ADD       X2          RUNLTU

X1          MULT        SRMHU       X2
ADD         .0005       X2
Z-ADD       X2          MACHTU
ENDIF
```

When time basis code is 'C'(cost/pc), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
EOP         MULT        SRLHU       X1
ADD         .0005       X1
Z-ADD       X1          RUNLTU
Z-ADD       SRMHU       MACHTU
```

When time basis code is 'M'(min/pc), run labor time (RUNTLU) and machine time(MACHTU) are calculated as follows:

```
SETUTM      DIV         60          X1
ADD         .0005       X1
Z-ADD       X1          SETUTM

EOP         DIV         60          X1
X1          MULT        SRLHU       X2
ADD         .0005       X2
Z-ADD       X2          RUNLTU

X1          MULT        SRMHU       X2
ADD         .0005       X2
Z-ADD       X2          MACHTU

ENDSL
```

The step time(STEPTM) is calculated based on prime load code(PLCDE). When prime load code(PLCDE) is '0'(no load hours are accumulated), the step time is calculated as follows:

```
PLCDE     WHEQ      '0'
Z-ADD     *ZERO     STEPTM
Z-ADD     *ZERO     STTM13
```

When prime load code(PLCDE) is '1'(use run machine hours), the step time is calculated as follows:

```
PLCDE     WHEQ '1'
Z-ADD     MACHTU    STEPTM
```

When prime load code(PLCDE) is '2'(use setup time), the step time is calculated as follows:

```
PLCDE     WHEQ '2'
Z-ADD     SETUTM    STEPTM
Z-ADD     SETUTM    STTM13
```

When prime load code(PLCDE) is '3'(use setup time + run machine hours), the step time is calculated as follows:

```
PLCDE     WHEQ      '3'
SETUTM    ADD       MACHTU    X2
ADD       .0005     X2
Z-ADD     X2        STEPTM
Z-ADD     SETUTM    STTM13
```

When prime load code(PLCDE) is '4'(use setup time + run machine hours), the step time is calculated as follows:

```
PLCDE     WHEQ      '4'
Z-ADD     RUNTLU    STEPTM
Z-ADD     *ZERO     STTM13
```

When prime load code(PLCDE) is '5'(use setup time + run labor hours), the step time is calculated as follows:

```
PLCDE     WHEQ     '5'
SETUTM    ADD      RUNTLU    X2
ADD       .0005    X2
Z-ADD     X2       STEPTM
Z-ADD     SETUTM   STTM13
ENDSL
```

The values are then inserted into JOBSTEP000 and JS13VR000 tables.

```
INSERT INTO JOBSTEP000
(PROCPLANID, TYPE, JSID, DESCR, NEXTJSID, SELECTRL, ALTJSID, ALOCRL,
STEPTMRL, STEPTIME, FREECHCKFG, HOLDTEMPFG, RESSCHDFG,
RESACTN1, RESNMBR1, RESID1, RESACTN2, RESNMBR2, RESID2,
RESACTN3, RESNMBR3, RESID3, RESACTN4, RESNMBR4, RESID4,
RESACTN5, RESNMBR5, RESID5, RESACTN6, RESNMBR6, RESID6)

VALUES
(:PROCPN, :TYPE, :JSID, :PPDESC, :NEXTJS, :SELRL, :ALTJS, :ALOCRL,
:STEPRL, :STEPTM, :FREECK, :HOLDTM, :RESCHD, :RACT1, :RNBR1, :RESID1,
:ACTN, :NMBR, :RESID, :ACTN, :NMBR, :RESID, ACTN, :NMBR, :RESID, :ACTN,
:NMBR, :RESID, :ACTN, :NMBR, :RESID)
```

The next process plan record is then fetched into program variables before the program loops.

```
FETCH PPCUR INTO:PROCPN, :JSID, :PPDESC, :SRMHU, :SSLHU, :SETCS,
:CYCOP, :SRLHU, :TQCTD, :TBCDE, :PLCDE, :CUMSY, :ORDSIZ
```

The ENDDO statement transfers control to the beginning of the loop again. The program runs until all there are no further records in MOROUT file that satisfy the conditions.

Retrieving Resource Information

The work center information from MAPICS is transferred into the FACTOR resource tables (RESRCXXX and RESGRPXXX) for a particular alternative. A set of FACTOR Resources are created for each production facility in the WRKCTR file. These Resources are members of the Resource Group.

The number of Resources created corresponds to the maximum shift capacity of the MAPICS/DB production facility. Resources are assigned to one or more Shift patterns which designate the days of the week and the hours during the day which they are scheduled to be available for production. Enough FACTOR resources to represent the maximum capacity on any shift are generated and placed in the group.

A cursor is declared to retrieve resources information.

```
DECLARE RGCUR CURSOR FOR
```

The following fields are selected from WRKCTR file.

```
SELECT      DISTINCT    WKCTR, WCDSC, DCAP1, DCAP2, DCAP3
            FROM        WRKCTR, JOBSTEP000
            WHERE       WRKCTR.WKCTR = JOBSTEP000.RESID1
```

The WHERE statement limits the information to the Manufacturing Order of our interest.

The following fields are then initialized to appropriate values.

Variable REALOC is initiated to 'N'.

The cursor RGCUR, is opened.

```
OPEN RGCUR
```

The values in the RGCUR are fetched into program variables.

```
FETCH RGCUR INTO :RGID, :DESCR, :DCAP1, :DCAP2, :DCAP3
```

A DO loop is defined for processing all records until the end-of-file (EOF) condition reaches.

```
SQLCOD      DOWEQ       *ZERO
```

The three shifts are named.

```
MOVEL       'FIRST'     SHIFT1      8
MOVEL       'SECOND'    SHIFT2      8
MOVEL       'THIRD'     SHIFT3      8
```

The other variables are initiated to appropriate values.

```
MOVEL       DESCR       RDESCR
MOVE        *BLANKS     RESRC
Z-ADD       *ZERO       SEQNUM
Z-ADD       1           IDNUM
Z-ADD       1           CURCAP
```

Maximum capacity is determined by comparing all shift capacities and selecting the maximum of all.

```
ADD         0.5         DCAP1
Z-ADD       CAP1        MAXCAP
ADD         .5          DCAP2
DCAP2       IFGR        MAXCAP
Z-ADD       DCAP2       MAXCAP
ENDIF

ADD         0.5         DCAP3
DCAP3       IFGT        MAXCAP
Z-ADD       CAP3        MAXCAP
ENDIF
```

A DO loop is defined to limit transactions to maximum capacity "MAXCAP".

```
CURCAP      DOWEQ       MAXCAP
```

Another DO loop is defined to limit resources to 20.

```
IDNUM       DOWEQ       20
CURCAP      ANDLE       MAXCAP
```

The resource is generated by concatinating variables RGID and ID.

```
MOVE        CURCAP      ID
RGID        CAT         ID:0            RESID       P
```

The shift ids are entered for each resource generated. The logic behind assigning shift ids is based on the shift capacity of the resources, DCAP1, DCAP2, and DCAP3. Every time the program loops, the value of the CURCAP counter is compared to the shift capacities. The program inserts blanks if the shift capacity is less than CURCAP value.

```
DCAP1       IFLT        CURCAP
MOVE        *BLANKS     SHIFT1
ENDIF

DCAP2       IFLT        CURCAP
MOVE        *BLANKS     SHIFT2
ENDIF

DCAP3       IFLT        CURCAP
MOVE        *BLANKS     SHIFT3
ENDIF
```

The values are then inserted into file RESRC000.

```
INSERT      INTO  RESRC000
            (RESID, RESTYPE, DESCR, SHIFTID1, SHIFTID2, SHIFTID3,
SHIFTID4, SELRL, SEQRL, ALLOCCD, MUSTCOMPFG, MAXORUN, SUMFG,
SCHEDFG, FINALFG, LOADFG)
VALUES
(:RESID, :RESTYP, :RDESCR, :SHIFT1, :SHIFT2, :SHIFT3, :SHIFT4, :SELRL,
:SEQRL, :ALLOCCD, :MSTCMP, :MAXOVR, :RSUMFG, :RSCDFG, :FINQFG,
:RLODFG)
```

The generated resource id is then put into the appropriate RESID variable for the group.

```
SELEC
IDNUM     WHEQ      1
MOVE      RESID     RES1
IDNUM     WHEQ      2
MOVE      RESID     RES2
IDNUM     WHEQ      3
MOVE      RESID     RES3
IDNUM     WHEQ      4
MOVE      RESID     RES4
IDNUM     WHEQ      5
MOVE      RESID     RES5
IDNUM     WHEQ      6
MOVE      RESID     RES6
IDNUM     WHEQ      7
MOVE      RESID     RES7
IDNUM     WHEQ      8
MOVE      RESID     RES8
IDNUM     WHEQ      9
MOVE      RESID     RES9
IDNUM     WHEQ      10
MOVE      RESID     RES10
IDNUM     WHEQ      11
MOVE      RESID     RES11
IDNUM     WHEQ      12
MOVE      RESID     RES12
IDNUM     WHEQ      13
MOVE      RESID     RES13
IDNUM     WHEQ      14
MOVE      RESID     RES14
IDNUM     WHEQ      15
MOVE      RESID     RES15
IDNUM     WHEQ      16
MOVE      RESID     RES16
IDNUM     WHEQ      17
MOVE      RESID     RES17
IDNUM     WHEQ      18
MOVE      RESID     RES18
IDNUM     WHEQ      19
MOVE      RESID     RES19
```

```
IDNUM     WHEQ      20
MOVE      RESID     RES20
```

The values of IDNUM, and CURCAP are incremented by 1 before the loop ends.

```
ADD  1    IDNUM
ADD  1    CURCAP
ENDDO
```

The loop continues until either 20 resources are generated or the current capacity (CURCAP) becomes more than maximum capacity (MAXCAP). Another IF statement is added before the values are entered into resource group table (RESGRP000).

```
CURCAP    IFLE  MAXCAP

INSERT INTO RESGRPOOO
            (RGID, DESCR, REALLOCFG, ALLOCRL, SUMFG, LOADFG,
SCHEDFG, SEQNUM, RESID1, RESID2, RESID3, RESID4, RESID5, RESID6,
RESID7, RESID8, RESID9, RESID10, RESID11, RESID12, RESID13, RESID14,
RESID15, RESID16, RESID17, RESID18, RESID19, RESID20)
VALUES
            (:RGID, :RDESCR, :REALOC, :ALLOCRL, :SUMFG, :LOADFG,
:SCHDFG, :SEQNUM, :RES1, :RES2, :RES3, :RES4, :RES5, :RES6, :RES7, :RES8,
:RES9, :RES10, :RES11, :RES12, :RES13, :RES14, :RES15, :RES16, :RES17, :RES18,
:RES19, :RES20)
```

The values of SEQNUM, and IDNUM are incremented before the the program loops for second time.

```
ADD       1         SEQNUM
Z-ADD     1         IDNUM
MOVE      *BLANKS   RESRC
ENDIF
ENDDO
```

Another record is fetched from RGCUR into variables before the program loops.

```
FETCH     RGCUR     INTO  :RGID, :DESCR, :DCAP1, :DCAP2,
                    :DCAP3
ENDDO
```

The cursor is then closed.

```
CLOSE RGCUR
```

Job Step Retrieval for Other Orders That Use the Transferred Resources

So far, we have transferred order information, job step information and resource information that pertain to one single order of our interest. FACTOR database is not complete with this data. For the simulation to be effective and efficient, we have to transfer other jobs that go into the resources transferred during the simulation period.

Transferring the information about other jobs demands us to transfer order information, job step information and part information about the other jobs also. If we go this way, the loop will never end, and we will end up transferring all jobs from MAPICS/DB. To avoid this, only those job steps, which go through the transferred resources are transferred to FACTOR. This is accomplished by the following SQL statements.

First, a cursor for job step retrieval is declared.

```
DECLARE RPCUR    CURSOR      FOR
```

Then the job steps that belong to any other jobs that go through the transferred resources are selected.

```
SELECT      MOROUT.ORDNO, OPSEQ, WKCTR, SRMHU, SSLHU, SETCS,
CYCOP,      SRLHU,      TQCTD,      TBCDE,      PLCDE,      ITEMASA.CUMSY,
MOMAST.ORQTY

FROM        MOROUT, MOMAST, ORDER000, ITEMASA, RESGRP000

WHERE       MOROUT.WKCTR    =       RESGRPOOO.RGID AND
```

```
        MOROUT.ORDNO    <>      ORDER000.ORDERID AND
        MOROUT.ORDNO    =       MOMAST.ORDNO AND
        MOMAST.FITEM    =       ITEMASA.ITNBR AND
        MOROUT.SCODT    >       :RELDAT AND
        MOROUT.SSTDT    <       :DUEDAT
ORDER BY ORDNO
```

After the job steps are transferred, the next step is to complete the FACTOR database by providing information about orders and parts. A cursor is declared to retrieve order information.

```
DECLARE OR2CUR CURSOR FOR
```

The following fields are selected from files MOMAST, MOROUT, ORDER000, and JOBSTEP000:

```
SELECT      MOMAST.ORDNO,      MOMAST.JOBNO,      MOMAST.FITEM,
MOMAST.ORQTY,
MOROUT.SSTDT, MOROUT.SCODT
FROM MOMAST, MOROUT, JOBSTEP000, ORDER000

WHERE       MOMAST.ACREC    =       'A'     AND
JOBSTEP000.PROCPLANID       =       MOROUT.ORDNO    AND
MOROUT.ORDNO                =       MOMAST.ORDNO    AND
JOBSTEP000.PROCPLANID       <>      ORDER000.ORDID
```

The WHERE statement limits the selected orders to those whose job steps were transferred. It also limits the transfer to only active orders and only those that were not transferred in the first place.

The order cursor, OR2CUR is then opened.

```
OPEN OR2CUR
```

The values in OR2CUR are fetched into program variables.

```
FETCH       OR2CUR      INTO :ORDID,    :ODESCR,    :PARTID,    :ORDSIZ,
:RELDAT, :DUEDAT
```

A DO loop is defined for processing all records until the end-of-file (EOF) condition reaches.

```
SQLCOD      DOWEQ       *ZERO
```

The order size is rounded.

```
MOVE        0.5         ORDSIZ
Z-ADD       ORDSIZ      RORDSIZ     60
```

The dates are moved to character strings.

```
MOVE        RELDAT      CRLDAT      6
MOVE        DUEDAT      CDUDAT      6
```

The values are then inserted into file ORDER000.

```
INSERT      INTO  ORDER000
(ORDERID, DESCR, PARTID, PROCPLANID, ORDSIZE, LOADSIZE, EXCESSCD,
RELDATE, RELTIME, DUEDATE, DUETIME, STATUSCD, PRIORITY,
SCHEDFG)
VALUES
(:ORDID, :ODESCR, :PARTID, :ORDID, :RORDSZ, RORDSZ, :EXCSCD, :CRLDAT,
:RELTIM, :CDUDAT, :DUETIM, :STATCD, :PRIO, :SCHDFG)
```

The release date(RELDATE) and the due date(DUEDATE) are equated to the release and due date of the particular job step. This measure is taken to avoid unnecessary looping in the program.

The the next order record is fetched, before the program loops.

```
FETCH OR2CUR INTO    :ORDID,    :ODESCR,    :PARTID,    :ORDSIZ,
:RELDAT,:DUEDAT
```

Part Transfer

The final step is transferring part information from MAPICS/DB to FACTOR. A cursor is declared to retrieve the part information.

```
DECLARE PARTCUR CURSOR FOR
```

The following fields are selected from files MOMAST, and ORDER000.

```
SELECT      DISTINCT    FITEM, FDESC
FROM        MOMAST, ORDER000
WHERE       MOMAST.FITEM    =       ORDER000.PARTID
```

The WHERE statement limits the part records to only those records which belong to the transferred orders.

The following auxiliary data is defaulted to appropriate values:

```
MOVE        *BLANKS     FAMILY      20
MOVE        *BLANKS     SUBFAM      20
MOVE        *BLANKS     PPID        20
MOVE        *BLANKS     STMATL      20
MOVE        *BLANKS     ENMATL      20
MOVE        *BLANKS     PTTAB       8
```

The PARTCUR is then opened.

```
OPEN  PARTCUR
```

The values in the PARTCUR are fetched into program variables.

```
FETCH       PARTCUR     INTO :PARTID, :PESCR
```

A DO loop is defined for processing all records until the end-of-file(EOF) condition reaches.

```
SQLCOD      DOWEQ       *ZERO
```

The values are inserted into PART000 file.

```
INSERT      INTO  PART000
(PARTID, DESCR, FAMILY, SUBFAMILY, PROCPLANID, STRTMATLID,
ENDMATLID, PTTABLE)
VALUES
(:PARTID, :PDESCR, :FAMILY, :SUBFAM, :PPID, :STMATL, :ENMATL, :PTTAB)
```

The next part record is fetched before the program loops.

```
FETCH       PARTCUR INTO :PARTID, :PDESCR

ENDDO
```

The part cursor is closed.

```
CLOSE PARTCUR
DONE        TAG
SETON       LR
```

At this point, the control is transferred back to the program NEWINTORD. If the program runs without any problems, the message "Transfer completed normally" is sent to the screen.

```
SNDPGMMSG       MSGID(CPF9898)      MSGF(QCPFMSG)MSGDTA('Transfer
completed normally') MSGTYPE(*ESCAPE)
GOTO            CMBLBL (DONE)
FUNCCK:         SNDPGMMSG MSGID(CPF9898) MSGF(QCPMSG)
                MSGDTA('Function check trapped in program NEWFERORD)
                MSGTYPE(*ESCAPE)
                GOTO CMDLBL(DONE)
NOTEX:          SNDPGMMSG       MSGID(CPF9898)
                MSGF(QCPFMSG)
                MSGDTA('The order file, part file, or job step file does not exist')
                MSGTYPE(*ESCAPE)
                GOTO CMDLBL(*DONE)

RELAY:          RCVMSG MSGQ(*PGMQ) MSGTYPE(*LAST)
                    MSGDTA(&MSG)
                MSGTYPE(*ESCAPE)
DONE:       ENDPGM
```

CHAPTER V

OTHER INTERFACE OPTIONS

Transfer Based on Resources

The second item in the main menu is designed to transfer information from MAPICS/DB to FACTOR 5.2 about a particular work center or resource in which the user is interested. A flow chart for this procedure is illustrated in Figure 2. Selecting this option activates a command named NEWFINTRES. NEWFINTRES is a CL command. The source code for this command is saved in member NEWFINTRES in the physical file named RCMDSRC.

NEWFINTRES Procedure

```
CMD     PROMPT('TRANSFER BASED ON RESOURCES')
PARM    KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999)
        MIN(1) FULL(*YES) PROMPT('Alternative number')
PARM    KWD(MACHINE) TYPE(*CHAR) LEN(6) MIN(1)  FULL(*YES)
        PROMPT('ENTER RESOURCE NUMBER')
PARM    KWD(STRDAT) TYPE(*CHAR) LEN(6) MIN(1)    FULL(*YES)
        PROMPT('ENTER START DATE')
PARM    KWD(ENDDAT) TYPE(*CHAR) LEN(6) MIN(1)    FULL(*YES)
        PROMPT('ENTER END DATE')
```

The NEWFINTRES command is connected to a CL program named NEWINTRES. The purpose of the command is to collect the altern number, machine number, start date, and end date, and pass them to the program.

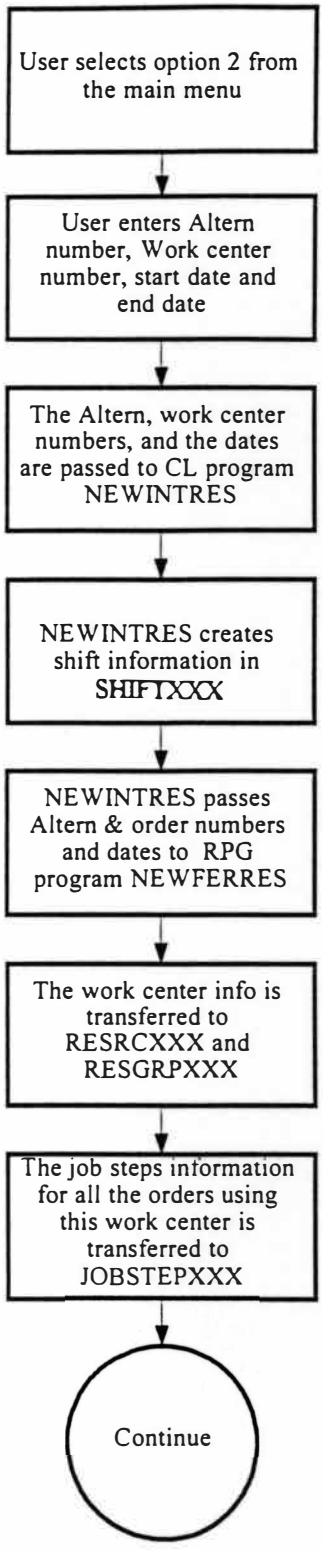

Figure 2. Flow Chart for Work Center Based Transfer.

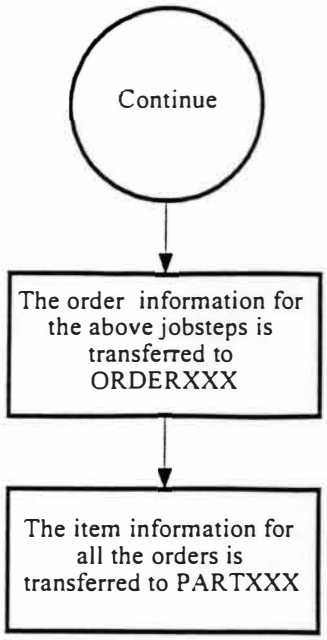

Figure 2 - Continued

NEWINTRES Procedure

First, the altern number, machine number, start date, and end date are placed into variables &CHALT, &MACHINE, &STRDAT, &ENDDAT.

```
PGM   PARM(&CHALT &MACHINE &STRDAT &ENDDAT)
```

Then the variables are declared.

```
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&MACHINE) TYPE(*CHAR) LEN(5)
DCL   VAR(&STRDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&ENDDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(10)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)
```

The following CL statements monitor for message IDs and route the program to the appropriate location.

```
MONMSG MSGID(CPF9999) EXEC(GOTO CMDLBL(FUNCCHK)
MONMSG MSGID(CPF9898) EXEC(GOTO CMDLBL(RELAY)
MONMSG MSGID(CPF1085) EXEC(GOTO CMDLBL(NOTEX)
```

The shift information is copied from file DFTSHIFT to SHIFTXXX file, where 'XXX' denotes the altern number

```
CALL PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF        FROMFILE(F52MFINT/DFTSHIFT) TOFILE(*LIBL/&OFILE)
            MBROPT(*REPLACE)
```

Then the file overrides are done on all Factor 5.2 files as explained in Chapter IV.

```
CALL        PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF      FILE(JOBSTEP000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
OVRDBF      FILE(JS13VR000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
OVRDBF      FILE(ORDER000) TOFILE(&OFILE)

CALL        PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF      FILE(PART000) TOFILE(&OFILE)
```

The actual transfer of data from MAPICS/DB to FACTOR 5.2 is done by calling the program NEWFERRES. The source code for this program is saved in member TEMFERRES in file RINTSRC.

```
CALL  PGM(TRANSFER/NEWFERRES)
PARM (&CHALT &MACHINE &STRDAT &ENDDAT)
```

NEWFERRES Procedure

NEWFERRES is a member in file RINTSRC, which contains RPG code with embedded SQL statements. The code begins with declaring three RPG data structures named DBFLDS, DBASES, and RESRC. Then the parameters that are passed to this program are saved in program variables ALTNO, MACH, CSTDAT, and CENDAT.

```
*ENTRY    PLIST
          PARM        ALTNO     3
          PARM        MACH      5
          PARM        CSTDAT    6
          PARM        CENDAT    6
```

Updating ALTERN Table

The ALTERN table is updated to have the simulation start date initiated to CSTDAT, start time to '0000', end date to CENDAT, and end time to '2359'.

```
UPDATE ALTERN
SET    STARTDATE = :CSTDAT, STARTTIME = '0000'
       ENDDATE = :CENDAT, ENDTIME = '2359'
       WHERE ALTNO = :ALTNUM
```

All of the FACTOR 5.2 database files are emptied of any previous data.

```
DELETE FROM ORDER000
DELETE FROM PART000
DELETE FROM JOBSTEP000
DELETE FROM JS13VR000
DELETE FROM RESRC000
DELETE FROM RESGRP000
```

Retrieving Resource Information

Since this menu item is designed to transfer MAPICS/DB data based on a machine number, the resource information is transferred first. A cursor is declared for this purpose.

```
DECLARE RGCUR CURSOR FOR
```

The following fields are selected from WRKCTR file.

```
SELECT      WKCTR, WCDSC, DCAP1, DCAP2, DCAP3
FROM        WRKCTR
```

The conditional statement WHERE is used to retrieve only the information pertaining to the machine number of user's interest.

```
WHERE       WKCTR = :MACH
ORDER BY WKCTR
```

Retrieving Process Plan Information

After the machine information is transferred to RESRC000 and RESGRP000 files, the next step is to transfer the job steps that use this machine. A cursor is declared to retrieve these job steps.

```
DECLARE     PPCUR CURSOR FOR
```

The following fields are selected from files MOROUT, MOMAST, ITEMASA, and RESGRP000.

```
SELECT      MOROUT.ORDNO, OPSEQ, WKCTR, SRMHU, SSLHU,
            SETCS, CYCOP, SRLHU, TQCTD, TBCDE, PLCDE,
            ITEMASA.CUMSY,
            MOMAST.ORQTY
FROM        MOROUT, MOMAST, ITEMASA, RESGRP000
```

The conditional statement is used to retrieve only those job steps from MOROUT, that fall between the start date and end date of simulation.

```
WHERE     MOROUT.WKCTR = RESGRP000.RGID AND
          MOROUT.SSTDT < :ENDDAT AND
          MOROUT.SCODT > :STRDAT AND
          MOROUT.ORDNO = MOMAST.ORDNO AND
          MOMAST.FITEM = ITEMASA.ITNBR
ORDER BY ORDNO, OPSEQ
```

The retrieved data is then processed the same way, as explained in NEWFERORD procedure, and hence not repeated in this section.

Retrieving Order Information

A cursor is declared to retrieve order information

```
DECLARE ORDCUR CURSOR FOR
```

The following fields are selected from MOMAST, MOROUT, and JOBSTEP000.

```
SELECT    DISTINCT MOMAST.ORDNO, JOBNO, FITEM, ORQTY,
          MOMAST.SSTDT, MOMAST.ODUDT
          FROM MOMAST, MOROUT, JOBSTEP000
WHERE     MOMAST.ACREC = 'A' AND
          JOBSTEP000.PROCPLANID = MOMAST.ORDNO AND
          MOMAST.ORDNO = MOROUT.ORDNO
```

The SELECT DISTINCT statement is used to avoid duplicate order records. The conditional statement ensures that only active orders are selected which match the job step records in file JOBSTEP000.

Retrieving Part Information

A cursor is declared for part retrieval.

```
DECLARE PARTCUR CURSOR FOR
```

The following fields are selected from files MOMAST, and ORDER000.

```
SELECT    DISTINCT FITEM, FDESC
FROM      MOMAST, ORDER000
WHERE     MOMAST.FITEM = ORDER000.PARTID
```

The WHERE statement limits the retrieval to only those records that belong to the transferred orders. The part cursor is closed after all the records are transferred.

```
CLOSE     PARTCUR
```

The program is terminated by using the following statements.

```
DONE      TAG
SETON     LR
```

The retrieved data by the SQL statements is processed the same way, as explained in NEWFERORD procedure in Chapter IV, and hence not repeated in this chapter.

Transfer the Entire Shop Floor Information From MAPICS/DB

Menu item (4) is used if the user is interested in observing the performance of the entire shop floor over a period of time. A flow chart for this procedure is illustrated in Figure 3. Selecting this option activates a command named NEWFINTTTL. NEWFINTTTL is a CL command. The source code for this command is saved in member NEWFINTTTL in the physical file named RCMDSRC.

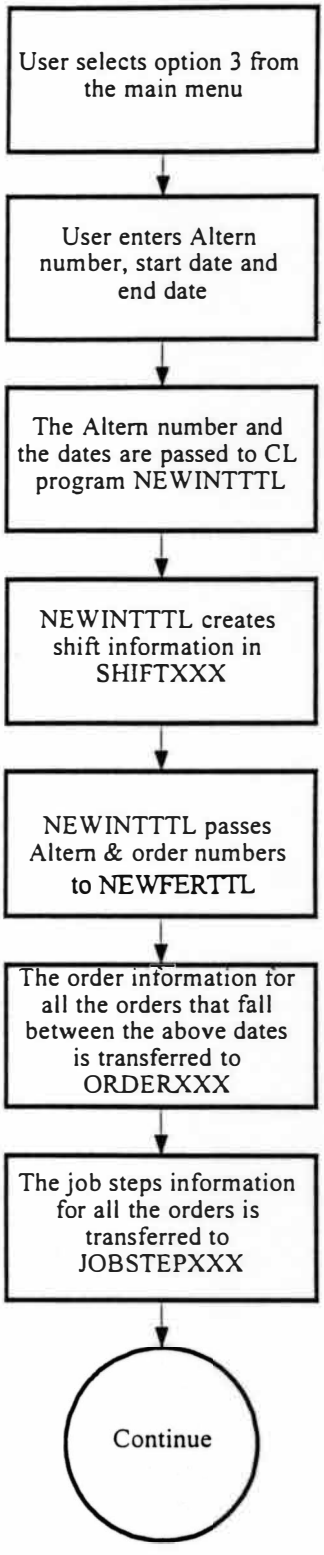

Figure 3. Flow Chart for Entire Shop Floor Transfer.

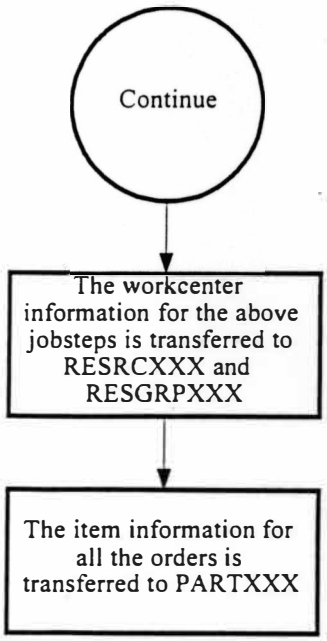

Figure 3 - Continued

NEWFINTTTL Procedure

```
CMD     PROMPT('TRANSFER ENTIRE JOB SHOP')
PARM    KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999)
        MIN(1) FULL(*YES) PROMPT('Alternative number')
PARM    KWD(STRDAT) TYPE(*CHAR) LEN(6) MIN(1) FULL(*YES)
        PROMPT('ENTER START DATE')
PARM    KWD(ENDDAT) TYPE(*CHAR) LEN(6) MIN(1) FULL(*YES)
        PROMPT('ENTER END DATE')
```

The NEWFINTTTL command is connected to a CL program named NEWINTTTL. The purpose of the command is to collect the altern number, start date, and end date, and pass them to the program.

NEWINTTTL Procedure

First, the altern number, start date, and end date are placed into variables &CHALT, &STRDAT, &ENDDAT.

```
PGM   PARM(&CHALT &STRDAT &ENDDAT)
```

Then the variables are declared.

```
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&STRDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&ENDDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(10)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)
```

The following CL statements monitor for message IDs and route the program to the appropriate location.

```
MONMSG MSGID(CPF9999) EXEC(GOTO CMDLBL(FUNCCHK)
MONMSG MSGID(CPF9898) EXEC(GOTO CMDLBL(RELAY)
MONMSG MSGID(CPF1085) EXEC(GOTO CMDLBL(NOTEX)
```

The shift information is copied from file DFTSHIFT to SHIFTXXX file, where 'XXX' denotes the altern number.

```
CALL PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF          FROMFILE(F52MFINT/DFTSHIFT) TOFILE(*LIBL/&OFILE)
              MBROPT(*REPLACE)
```

Then the file overrides are done on all FACTOR 5.2 files as explained in Chapter IV.

```
CALL          PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF        FILE(JOBSTEP000) TOFILE(&OFILE)

CALL          PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
OVRDBF        FILE(JS13VR000) TOFILE(&OFILE)
```

```
CALL      PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
OVRDBF    FILE(ORDER000) TOFILE(&OFILE)

CALL      PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF    FILE(PART000) TOFILE(&OFILE)
```

The actual transfer of data from MAPICS/DB to FACTOR 5.2 is done by calling the program NEWFERTTL. The source code for this program is saved in member TEMFERTTL in file RINTSRC.

```
CALL  PGM(TRANSFER/NEWFERTTL)
PARM (&CHALT &STRDAT &ENDDAT)
```

NEWFERTTL Procedure

NEWFERTTL is a member in file RINTSRC, which contains RPG code with embedded SQL statements. The code begins with declaring three RPG data structures named DBFLDS, DBASES, and RESRC. Then the parameters that are passed to this program are saved in program variables ALTNO, CSTDAT, and CENDAT.

Only the key SQL statements which were used to retrieve the information, are mentioned in this section. The default field initialization, and the process plan calculations were discussed in Chapter IV.

```
*ENTRY    PLIST
          PARM      ALTNO     3
          PARM      CSTDAT    6
          PARM      CENDAT    6
```

Updating ALTERN Table

The ALTERN table is updated to have the simulation start date initiated to CSTDAT, start time to '0000', end date to CENDAT, and end time to '2359'.

```
UPDATE ALTERN
SET    STARTDATE = :CSTDAT, STARTTIME = '0000'
       ENDDATE = :CENDAT, ENDTIME = '2359'
       WHERE ALTNO = :ALTNUM
```

All the FACTOR 5.2 database files are emptied of any previous data.

```
DELETE FROM ORDER000
DELETE FROM PART000
DELETE FROM JOBSTEP000
DELETE FROM JS13VR000
DELETE FROM RESRC000
DELETE FROM RESGRP000
```

Retrieving Order Information

Since this menu item is designed to transfer MAPICS/DB data based on start and end dates, the information about all the orders that fall in between these dates is transferred first. A cursor is declared for this purpose.

```
DECLARE OR3CUR CURSOR FOR
SELECT ORDNO, JOBNO, FITEM, ORQTY, SSTDT, ODUDT
FROM MOMAST
WHERE ACREC = 'A' AND SSTDT <= :ENDDAT
```

Retrieving Part Information

The part information is retrieved for all the orders transferred, as mentioned below:

```
DECLARE PARTCUR FOR
```

```
SELECT DISTINCT FITEM, FDESC
FROM MOMAST, ORDER000
WHERE MOMAST.FITEM = ORDER000.PARTID
```

Retrieving Process Plan Information

The process plan information for the transferred orders is retrieved using the following SQL statements:

```
DECLARE PPCUR CURSOR FOR
SELECT MOROUT.ORDNO, OPSEQ, WKCTR, SRMHU, SSLHU, SETCS,
       CYCOP, SRLHU, TQCTD, TBCDE, PLCDE, ITEMASA.CUMSY,
       ORDER000.ORDSIZE
FROM MOROUT, MOMAST, ORDER000, ITEMASA
WHERE MOROUT.ORDNO = ORDER000.ORDERID AND
       ORDER000.ORDERID = MOMAST.ORDNO AND
       MOMAST.FITEM = ITEMASA.ITNBR AND
       MOROUT.OPSEQ >= MOMAST.OPCUR
```

Retrieving Resource Information

The work center or resource information is retrieved for all the job steps transferred, as follows:

```
SELECT    DISTINCT    WKCTR, WCDESC, DCAP1, DCAP2, DCAP3
          FROM        WRKCTR, JOBSTEP000
          WHERE       WRKCTR.WKCTR = JOBSTEP000.RESID1
```

CHAPTER VI

CONCLUSION

Benefits of the Interface Application

In this study, an Interface Application has been developed to transfer data from MAPICS/DB, a MRP II application and FACTOR 5.2, a Finite Capacity Scheduling application. The manufacturing industry can benefit from this Interface in many ways. Some are mentioned below:

1. With the Interface Application, the completion time of each production order can be projected considering material availability and finite capacity resources. Advance visibility of future production conditions greatly facilitates management of constraints. Early preventive action to avoid delays at capacity constraints results in shorter lead times, even in complex production environments.

2. The flexible nature of the Interface Application will provide manufacturing managers with a powerful new way of asking and answering "what if" questions. When the model is executed in the computer, simulated time advances in the model just as the plant would actually operate. Statistics are automatically collected by the model to report on bottleneck operations, equipment usage, levels of work-in-process inventory and factory throughput for various products.

3. Alternative schedules can be developed and compared instantly, based on the efficiency information gathered.

4. Data entry is minimized to just auxiliary information. Without the Interface Application, FACTOR 5.2 would have required manual entry of the entire manufacturing data about orders and resources.

5. By understanding the implications of integrating the scheduling system and the MRP II system, one can understand the concept of Just-In-Time(JIT) operations. This will lead towards reducing and/or eliminating unnecessary operations or data flows in a closed loop manufacturing environment where the customer is very important.

Future Research Suggestions

This Interface Application can be considered as the first step towards a completely integrated manufacturing environment The aim of this integration is to make Computer Integrated Manufacturing more dynamic by providing additional channels of information flow.

The current Interface Application can transfer information from the MRP II system to the scheduling system. The scheduling system is used as a mirror which reflects an image of the manufacturing environment based on the information transferred from the MRP II system. Further research is recommended to be able to automatically send important information back to the MRP II system. For example, the availability of the projected start date of orders based on the actual availability of capacity, will enable the MRP II systems to implement Just-In-Time(JIT) purchase of raw materials.

Appendix A

User's Guide for the MAPICS/DB and FACTOR 5.2 Interface

Instructions for MAPICS/DB - FACTOR 5.2 Interface Application

Computer requirements:

The MAPICS/DB and FACTOR 5.2 Interface package runs on the AS/400 computer system, with OS/400 operating system. MAPICS/DB and FACTOR 5.2 software must be installed on the system prior to running the application. Required software include RPG/400, CL/400, and SQL/400.

Library list:

To execute the program, certain libraries are to be added to the library list. These libraries are (1) TRANSFER (contains all the necessary programs needed to run the program), (2) FACTOR52 (contains FACTOR 5.2 software), (3) NEW (contains FACTOR Database library), (4) AMFLIBR (MAPICS/DB library that contains all data files for RA environment), (5) AMALIBA (MAPICS/DB library that contains all application files for RA environment).

All the above libraries can be added to the current library list by using ADDLIBLE command.

Operating Instructions

(1) Log on to AS/400 using your login name and password.

(2) Call the transfer menu by typing Go transfer and press enter.

(3) Type GO TRANSFER and press enter key.

(4) The TRANSFER menu consists of the following items
1)Transfer a single manufacturing order
2) Transfer a work center
3) Transfer the jobshop
4) Use MAPICS/DB
5) Use FACTOR 5.2
90) Sign Off

Transferring a single manufacturing order: Select choice '1' by typing '1' and pressing enter. A second display screen appears titled 'Schedule based on Manufacturing Orders'. This display screen has two input fields: (a) Alternative number, and (b) Mfg. order number. Type '000' for the alternative number. The cursor goes to the next field. Then enter a valid manufacturing order number. A valid manufacture order number is an existing manufacturing order number in MAPICS/DB. If a valid manufacturing order number is not entered, the program will send an error message to the screen and will cease to run. Press enter when the required data is entered. This program will take about three to four minutes to transfer the manufacturing order from MAPICS/DB to FACTOR 5.2. After it is done, the program displays a message "Transfer completed normally" and the main menu appears again.

Transfer a work center: Select choice '2' by typing '2' and pressing enter. A second display screen appears titled 'Schedule based on Manufacturing Orders'. This display screen has four input fields: Alternative number, resource number, start date, and end date. Type '000' for the

alternative number. The cursor goes to the next field. Then enter a valid resource number. A valid manufacture order number is an existing machine number in MAPICS/DB. If a valid machine number is not entered, the program will send an error message to the screen and will cease to run.

The start date and end date have to be entered in the format 'yymmdd' (year-month-date). For example, December 15th 1995 has to be entered as '951215'. Press enter when the required data is typed. This program will take about three to four minutes to transfer the information from MAPICS/DB to FACTOR 5.2. After the transfer is done, the program brings the main menu back with a message "Transfer completed normally".

Transfer the job shop: Select choice '3' by typing '3' and pressing enter. A second display screen appears titled 'Transfer job shop information'. This display screen has four input fields: Alternative number, start date, and end date. Type '000' for the alternative number. The cursor goes to the next field. The start date and end date have to be entered in the format 'yymmdd' (year-month-date). For example, December 15th 1995 has to be entered as '951215'. Press enter when the required data is typed. This program will take about three to four minutes to transfer the information from MAPICS/DB to FACTOR 5.2. After the transfer is done, the program brings the main menu back with a message "Transfer completed normally".

Appendix B

File Names and Contents

Table 2

File Names and Contents

Name	Type	Description
TRANSFER	*LIB	Contains all the objects
NEWINTORD	*PGM	Support for order transfer
NEWFERORD	*PGM	Order transfer program
NEWINTRES	*PGM	Support for work center transfer
NEWFERRES	*PGM	Work center transfer program
NEWINTTTL	*PGM	Support for total transfer
NEWFERTTL	*PGM	Total transfer program
NEWFINTORD	*CMD	Order transfer command
NEWFINTRES	*CMD	Work center transfer command
NEWFINTTTL	*CMD	Total transfer command
QCMDSRC	*FILE	File of command definitions
QRPGSRC	*FILE	Source code for RPG programs
QCLSRC	*FILE	Source code for CL programs

Appendix C

Program Screens

MAPICS/DB-FACTOR 5.2 TRANSFER MENU

Select one of the following:

1. Transfer a single manufacturing order
2. Transfer a work center
3. Transfer the job shop
4. Use MAPICS/DB
5. Use FACTOR 5.2

90. Sign off

Selection or command

F3= Exit F12= Cancel
F13 = User Support F16 = System main menu

Figure 4. MAPICS/DB and FACTOR 5.2 Interface Main Menu

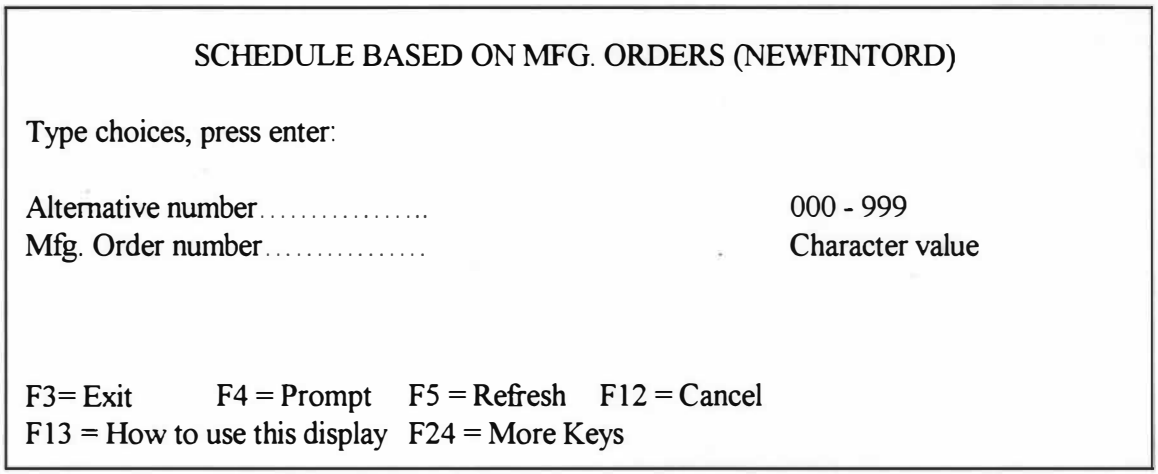

Figure 5. MAPICS/DB and FACTOR 5.2 Interface - Order based transfer screen

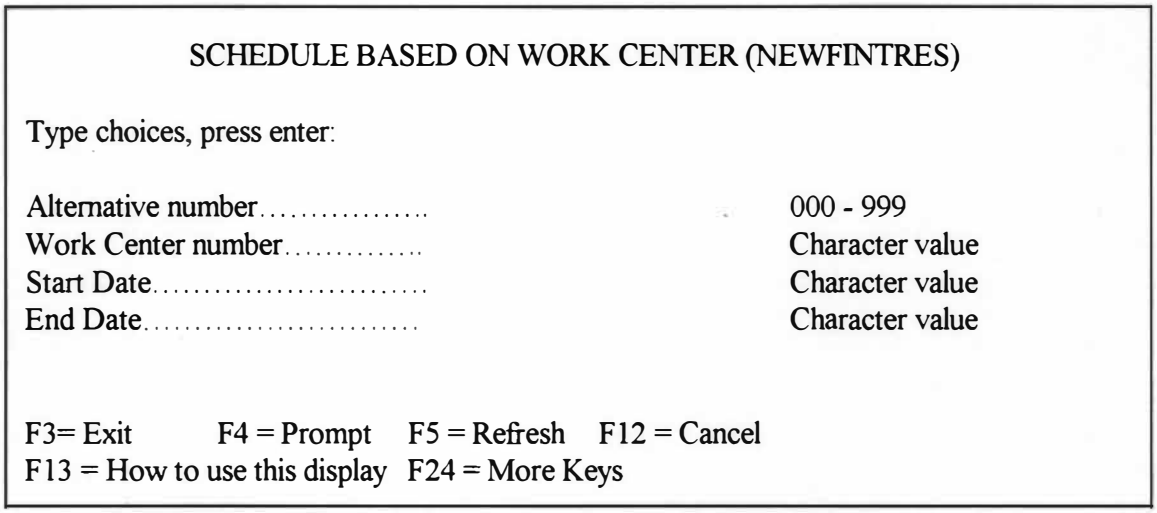

Figure 6. MAPICS/DB and FACTOR 5.2 Interface - Work Center based transfer screen

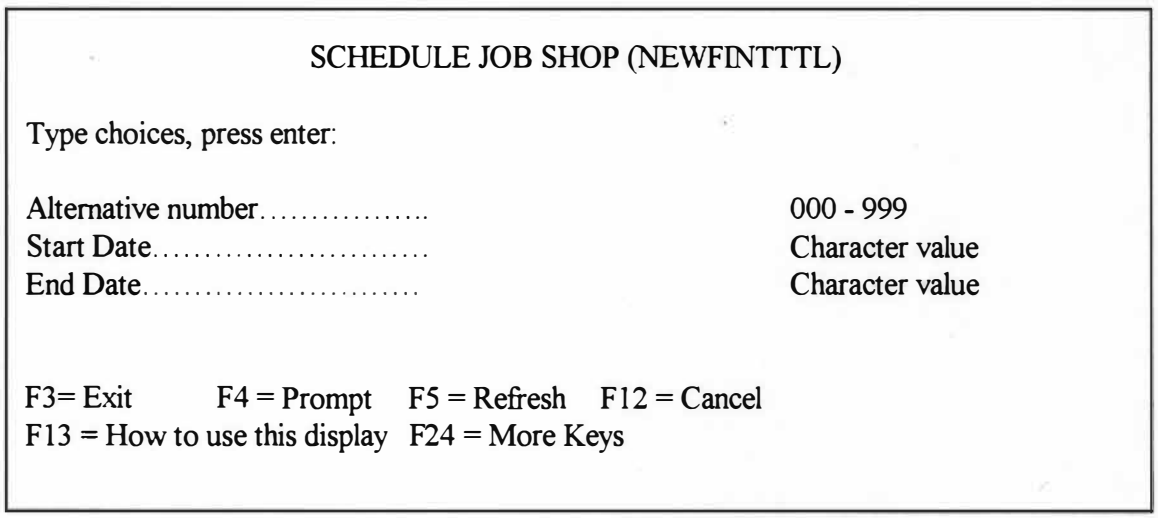

Figure 7. MAPICS/DB and FACTOR 5.2 Interface - Job Shop transfer screen

Appendix D

Program Source Code

Source Code for NEWFINTORD Command

```
CMD     PROMPT('TRANSFER BASED ON MFG. ORDERS')

PARM    KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999) +
        MIN(1) FULL(*YES) PROMPT('Alternative number)

PARM    KWD(ORDNO) TYPE(*CHAR) LEN(7) MIN(1) +
        FULL(*YES) PROMPT('MFG. ORDER NO')
```

Source Code for NEWFINTRES Command

```
CMD     PROMPT('SCHEDULE BASED ON WORK CENTER')

PARM    KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999) +
        MIN(1) FULL(*YES) PROMPT('Alternative number)

PARM    KWD(MACHINE) TYPE(*CHAR) LEN(5) MIN(1) +
        FULL(*NO) PROMPT('Resource Number')

PARM    KWD(STRDDAT) TYPE(*CHAR) LEN(5)  +
        MIN(1) FULL(*YES) PROMPT('Start Date')

PARM    KWD(ENDDAT) TYPE(*CHAR) LEN(5)  +
        MIN(1) FULL(*YES) PROMPT('End Date')
```

Source Code for NEWFINTRES Command

```
CMD     PROMPT('SCHEDULE JOB SHOP')

PARM    KWD(ALTNO) TYPE(*CHAR) LEN(3) RANGE(000 999) +
        MIN(1) FULL(*YES) PROMPT('Alternative number')

PARM    KWD(STRDDAT) TYPE(*CHAR) LEN(5)  +
        MIN(1) FULL(*YES) PROMPT('Start Date')

PARM    KWD(ENDDAT) TYPE(*CHAR) LEN(5)  +
        MIN(1) FULL(*YES) PROMPT('End Date')
```

Source Code for NEWINTORD Program

```
PGM   PARM(&CHALT &ORDER)
/*  Declare the input parameters */
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&ORDER) TYPE(*CHAR) LEN(7)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(7)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)

/* Monitor for messages */
MONMSG MSGID(CPF9999) EXEC(GOTO CMDLBL(FUNCCHK))
MONMSG MSGID(CPF9898) EXEC(GOTO CMDLBL(RELAY))
MONMSG MSGID(CPF1085) EXEC(GOTO CMDLBL(NOTEX))

/* Shift Creation */
CALL PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF FROMFILE(F52MFINT/DFTSHIFT) TOFILE(*LIBL/&OFILE)  +
        MBROPT(*REPLACE)

/* File Overrides */
CALL PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF FILE(JOBSTEP000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
OVRDBF FILE(JS13VR000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
OVRDBF FILE(ORDER000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF FILE(PART000) TOFILE(&OFILE)

/* Call the RPG program to do the actual transfer */
CALL PGM(TRANSFER/NEWFERORD)  +
PARM(&CHALT &ORDER)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG)  +
        MSGDTA('Transfer completed normally') MSGTYPE(*COMP)
GOTO CMDLBL(DONE)

FUNCCHK:    SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG)  +
            MSGDTA('Function check trapped in program NEWFERORD')  +
            MSGTYPE(*ESCAPE)
            GOTO CMDLBL(DONE)
```

```
NOTEX:      SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
            MSGDTA('The order file, jobstep file, or part file does not exist') +
            MSGTYPE(*ESCAPE)
            GOTO CMDLBL(DONE)

RELAY:      RCVMSG          MSGQ(*PGMQ)          MSGTYPE(*LAST)
MSGDTA(&MSG)
            SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
            MSGDTA(&MSG) MSGTYPE(*ESCAPE)

DONE:       ENDPGM
```

Source Code for NEWINTRES

```
PGM   PARM(&CHALT  &MACHINE &STRDAT &ENDDAT)
/*  Declare the input parameters */
DCL    VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL    VAR(&MACHINE) TYPE(*CHAR) LEN(5)
DCL    VAR(&STRDAT) TYPE(*CHAR) LEN(6)
DCL    VAR(&ENDDAT) TYPE(*CHAR) LEN(6)
DCL    VAR(&OFILE) TYPE(*CHAR) LEN(7)
DCL    VAR(&MSG) TYPE(*CHAR) LEN(72)

/* Monitor for messages */
MONMSG MSGID(CPF9999) EXEC(GOTO CMDLBL(FUNCCHK))
MONMSG MSGID(CPF9898) EXEC(GOTO CMDLBL(RELAY))
MONMSG MSGID(CPF1085) EXEC(GOTO CMDLBL(NOTEX))

/* Shift Creation */
CALL PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF FROMFILE(F52MFINT/DFTSHIFT) TOFILE(*LIBL/&OFILE) +
       MBROPT(*REPLACE)

/* File Overrides */
CALL PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF FILE(JOBSTEP000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
OVRDBF FILE(JS13VR000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
```

```
OVRDBF FILE(ORDER000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF FILE(PART000) TOFILE(&OFILE)

/* Call the RPG program to do the actual transfer */
CALL PGM(TRANSFER/NEWFERRES) +
PARM(&CHALT &MACHINE &STRDAT &ENDDAT)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
        MSGDTA('Transfer completed normally') MSGTYPE(*COMP)
GOTO CMDLBL(DONE)

FUNCCHK:  SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG)  +
          MSGDTA('Function check trapped in program NEWFERORD') +
          MSGTYPE(*ESCAPE)
          GOTO CMDLBL(DONE)
NOTEX:    SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG)  +
          MSGDTA('The order file, jobstep file, or part file does not exist') +
          MSGTYPE(*ESCAPE)
          GOTO CMDLBL(DONE)

RELAY:    RCVMSGMSGQ(*PGMQ) MSGTYPE(*LAST) MSGDTA(&MSG)
          SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG)  +
          MSGDTA(&MSG) MSGTYPE(*ESCAPE)

DONE:     ENDPGM
```

Source Code for NEWINTTTL

```
PGM   PARM(&CHALT &STRDAT &ENDDAT)
/*  Declare the input parameters */
DCL   VAR(&CHALT) TYPE(*CHAR) LEN(3)
DCL   VAR(&STRDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&ENDDAT) TYPE(*CHAR) LEN(6)
DCL   VAR(&OFILE) TYPE(*CHAR) LEN(7)
DCL   VAR(&MSG) TYPE(*CHAR) LEN(72)

/* Monitor for messages */
MONMSG MSGID(CPF9999) EXEC(GOTO CMDLBL(FUNCCHK))
MONMSG MSGID(CPF9898) EXEC(GOTO CMDLBL(RELAY))
MONMSG MSGID(CPF1085) EXEC(GOTO CMDLBL(NOTEX))
```

```
/* Shift Creation */
CALL PGM(GTFLNM) PARM(SHIFT &CHALT &OFILE)
CPYF FROMFILE(F52MFINT/DFTSHIFT) TOFILE(*LIBL/&OFILE) +
        MBROPT(*REPLACE)

/* File Overrides */
CALL PGM(GTFLNM) PARM(JOBSTEP &CHALT &OFILE)
OVRDBF FILE(JOBSTEP000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(JS13VR &CHALT &OFILE)
OVRDBF FILE(JS13VR000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(ORDER &CHALT &OFILE)
OVRDBF FILE(ORDER000) TOFILE(&OFILE)

CALL PGM(GTFLNM) PARM(PART &CHALT &OFILE)
OVRDBF FILE(PART000) TOFILE(&OFILE)

/* Call the RPG program to do the actual transfer */
CALL PGM(TRANSFER/NEWFERTTL) +
PARM(&CHALT &STRDAT &ENDDAT)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
        MSGDTA('Transfer completed normally') MSGTYPE(*COMP)
GOTO CMDLBL(DONE)

FUNCCHK:    SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
            MSGDTA('Function check trapped in program NEWFERORD') +
            MSGTYPE(*ESCAPE)
            GOTO CMDLBL(DONE)
NOTEX:      SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
            MSGDTA('The order file, jobstep file, or part file does not exist') +
            MSGTYPE(*ESCAPE)
            GOTO CMDLBL(DONE)

RELAY:      RCVMSG          MSGQ(*PGMQ)          MSGTYPE(*LAST)
MSGDTA(&MSG)
            SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) +
            MSGDTA(&MSG) MSGTYPE(*ESCAPE)

DONE:       ENDPGM
```

Source Code for GTFLNM

```
PGM PARM(&DTATYP &ALTNO &FILNAM)
/* Declare Variables */
DCL VAR(&DATTYP) TYPE(*CHAR) LEN(9)
DCL VAR(&ALTNO) TYPE(*CHAR) LEN(3)
DCL VAR(&FILNAM) TYPE(*CHAR) LEN(10)

MONMSG NSGID(CPF9999) EXEC(GOTO CMDLBL (FUNCCHK))
CRTDTAARA DTAARA(QTEMP/GTFNM) TYPE(*CHAR) LEN(10)
MONMSG MSGID(CPF1023)
CALL PGM(GTFNM) PARM(&DTATYP &ALTNO)
RTVDTAARA DTAARA(GTFNM) RTNVAR(&FILNAM)
DLTDTAARA DTAARA(GTFNM)
IF COND(&FILENAM *EQ '*NOALT') THEN (DO)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA   +
('Alternative ' *BACT *ALTNO *BCAT 'does not exist') MSGTYPE(*ESCAPE)
CHGVAR VAR(&FILENAM) VALUE('       ')
ENDDO
ELSE CMD(IF COND(&FILENAM *EQ '*NOREC') THEN (DO))
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA   +
('Record type' *BACT *DTATYP *BCAT 'does not exist') MSGTYPE(*ESCAPE)
CHGVAR VAR(&FILENAM) VALUE('       ')
ENDDO
GOTO CMDLBL(DONE)
FUNCCHK:  +
        SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA  +
        ('Function check trapped in program EDTFDB') MSGTYPE(*ESCAPE)
DONE:  +
ENDPGM
```

Appendix E

File Structures & Field Assignments

Table 3

File Structure and Field Assignments for ORDERXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
ORDERID	CHAR(20)	:ORDID	ORDNO	CHAR(7)	MOMAST
DESCR	CHAR(30)	:ODESCR	JOBNO	CHAR(6)	MOMAST
PARTID	CHAR(20)	:PARTID	FITEM	CHAR(15)	MOMAST
PROCPLANID	CHAR(20)	:ORDID	ORDNO	CHAR(7)	MOMAST
ORDSIZE	INT(9)	:RORDSZ>:ORDSIZ	ORQTY	DEC(10,3)	MOMAST
LOADSIZE	INT(9)	:RORDSZ>:ORDSIZ	ORQTY	DEC(10,3)	MOMAST
EXCESSCD	CHAR(1)	:EXCSCD	'A'	-------	USER
RELDATE	CHAR(6)	:CRLDAT>:RELDAT	SSTDT	DEC(6,0)	MOMAST
RELTIME	CHAR(4)	:RELTIM	'0000'	-------	USER
DUEDATE	CHAR(6)	:CDUEDT>:DUEDAT	ODUDT	DEC(6,0)	MOMAST
DUETIME	CHAR(4)	:DUETIM	'2359'	-------	USER
STATUSCD	CHAR(1)	:STATCD	'N'	-------	USER
PRIORITY	SMALLINT(4)	:PRIO	*ZERO	-------	USER
SCHEDFG	CHAR(1)	:SCHDFG	'Y'	-------	USER

Table 4

File Structure and Field Assignments for PARTXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
PARTID	CHAR(20)	:PARTID	FITEM	CHAR(15)	MOMAST
FAMILY	CHAR(20)	:FAMILY	*BL	-------	USER
SUBFAMILY	CHAR(20)	:SUBFAM	*BL	-------	USER
DESCR	CHAR(30)	:PDESCR	FDESC	CHAR(30)	MOMAST
PROCPLANID	CHAR(20)	:PPID	*BL	-------	USER
STRTMATLID	CHAR(20)	:STMATL	*BL	-------	USER
ENDMATLID	CHAR(20)	:ENMATL	*BL	-------	USER
PTTABLE	CHAR(8)	:PTTAB	*BL	-------	USER

Table 5

File Structure and Field Assignments for JOBSTEPXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
PROCPLANID	CHAR(20)	:PROCPN	ORDNO	CHAR(7)	MOROUT
TYPE	SMALLINT(4)	:TYPE	'13'	-------	USER
JSID	CHAR(8)	:JSID	OPSEQ	CHAR(4)	MOROUT
DESCR	CHAR(30)	:PPDESC	WKCTR	CHAR(5)	MOROUT
NEXTJSID	CHAR(8)	:NEXTJS	OPSEQ	CHAR(4)	MOROUT
SELECTRL	SMALLINT(4)	:SELRL	*ZERO	-------	USER
ALTJSID	CHAR(8)	:ALTJS	*BLANKS	-------	USER
ALOCRL	SMALLINT(4)	:ALOCRL	'2'	-------	USER
STEPTMRL	SMALLINT(4)	:STEPRL	'1'	-------	USER
STEPTIME	FLOAT(17)	:STEPTM	SELEC	-------	
FREECHCKFG	CHAR(1)	:FREECK	'Y'	--------	USER
HOLDTEMPFG	CHAR(1)	:HOLDTM	'N'	-------	USER
RESSCHDFG	CHAR((1)	:RESCHD	'Y'	-------	USER
RESACTN1	CHAR(1)	:RACT1	'S'	-------	USER
RESNMBR1	SMALLINT(4)	:RNBR1	'1'	-------	USER
RESID1	CHAR(8)	:RESID1>:PPDESC	WKCTR	CHAR(5)	MOROUT
RESACTN2	CHAR(1)	:ACTN	*BLANKS	-------	USER
RESNMBR2	SMALLINT(4)	:NMBR	*ZERO	-------	USER
RESID2	CHAR(8)	:RESID	*BLANKS	-------	USER

Table 5 - Continued

RESACTN3	CHAR(1)	:ACTN	*BLANKS	-------	USER
RESNMBR3	SMALLINT(4)	:NMBR	*ZERO	-------	USER
RESID3	CHAR(8)	:RESID	*BLANKS	-------	USER
RESACTN4	CHAR(1)	:ACTN	*BLANKS	-------	USER
RESNMBR4	SMALLINT(4)	:NMBR	*ZERO	-------	USER
RESID4	CHAR(8)	:RESID	*BLANKS	-------	USER
RESACTN5	CHAR(1)	:ACTN	*BLANKS	-------	USER
RESNMBR5	SMALLINT(4)	:NMBR	*ZERO	-------	USER
RESID5	CHAR(8)	:RESID	*BLANKS	-------	USER
RESACTN6	CHAR(1)	:ACTN	*BLANKS	-------	USER
RESNMBR6	SMALLINT(4)	:NMBR	*ZERO	-------	USER
RESID6	CHAR(8)	:RESID	*BLANKS	-------	USER

Table 6

File Structure and Field Assignments for JS13VRXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
PROCPLANID	CHAR(20)	:PROCPN	ORDNO	CHAR(7)	MOROUT
JSID	CHAR(8)	:JSID	OPSEQ	CHAR(4)	MOROUT
RGID	CHAR(8)	:RESID1>:PPDESC	WKCTR	CHAR(5)	MOROUT
WHENRL	SMALLINT(4)	:WHENRL	'1'	-------	USER
BASEDCD	CHAR(1)	:BASECD	'P'	-------	USER
START	SMALLINT(4)	:START	'1'	-------	USER
LENGTH	SMALLINT(4)	:LENGTH	'40	-------	USER
TABLEID	CHAR(8)	:TABID	*BLANKS	-------	USER
RSETUPID	CHAR(8)	:RSETUP	*BLANKS	-------	USER
STEPTMRL	SMALLINT(4)	:STRL13	'5'	-------	USER
STEPTIME	FLOAT(17)	:STTM13	SELECT*	-------	PROG

Table 7

File Structure and Field Assignments for RESRCXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
RESID	CHAR(8)	:RESID>:RGID	WKCTR	CHAR(5)	WRKCTR
DESCR	CHAR(30)	:RDESCR>:DESCR	WCDSC	CHAR(40)	WRKCTR
RESTYPE	CHAR(8)	:RESTYP	*BL	-------	USER
SHIFTID1	CHAR(8)	:SHIFT1	FIRST OR *BL	-------	USER
SHIFTID2	CHAR(8)	:SHIFT2	SECOND OR *BL	-------	USER
SHIFTID3	CHAR(8)	:SHIFT3	THIRD OR *BL	-------	USER
SHIFTID4	CHAR(8)	:SHIFT4	*BL	-------	USER
SELRL	SMALLINT(4)	:SELRL(2,0)	11	-------	USER
SEQRL	SMALLINT(4)	:SEQRL(2,0)	*ZERO	-------	USER
ALLOCCD	CHAR(1)	:ALOCCD	'P'	-------	USER
MUSTCOMPFG	CHAR(1)	:MSTCMP	'N'	-------	USER
MAXORUN	FLOAT(17)	:MAXOVR(9,3)	*ZERO	-------	USER
SUMFG	CHAR(1)	:RSUMFG	'Y'	-------	USER
SCHEDFG	CHAR(1)	:RSCDFG	'Y'	-------	USER
FINALQFG	CHAR(1)	:FINQFG	'N'	-------	USER
LOADFG	CHAR(1)	:RLODFG	'Y'	-------	USER

Table 8

File Structure and Field Assignments for RESGRPXXX

TO FIELD	TYPE	VARIABLE	FROM FIELD	TYPE	DATABASE
RGID	CHAR(8)	:RGID	WKCTR	CHAR(5)	WRKCTR
DESCR	CHAR(30)	:RDESCR>:DESCR	WCDSC	CHAR(40)	WRKCTR
REALLOCFG	CHAR(1)	:REALOC	'I'	-------	USER
ALLOCRL	SMALLINT(4)	:ALOCRL	*ZERO	-------	USER
SUMFG	CHAR(1)	:SUMFG	'Y'	-------	USER
LOADFG	CHAR(1)	:LOADFG	'Y'	-------	USER
SCHEDFG	CHAR(1)	:SCHDFG	'Y'	-------	USER
SEQNUM	SMALLINT(4)	:SEQNUM	CNTR	-------	USER
RESID1-20		:RES1-20			

BIBLIOGRAPHY

Cassis, Sami. (March, 1994). The Agile Factory. APICS

Huber, F. Robert (1987). Simulate Integrate Innovate. Production.

Lankford, Ray (1994). Here's How To Integrate MRP II With Execution Systems. APICS

Norman B. Van (January, 1991). Asking 'What If?' - Real Answers from Simulation Software. Manufacturing Systems.

Yaeger, Judy (1993). Programming In RPG/400, Duke Communications International