AD-A250 095

CUMENTATION PAGE

Form Approved
OMB No 0704-0188

2. REPORT DATE
1991

3. REPORT TYPE AND DATES COVERED
THESIS/XXXXXXXXXX

4. TITLE AND SUBTITLE
Computer Aided Process Planning (CAPP): The User Interface for The Fabrication Module of the Rapid Design System

5. FUNDING NUMBERS

6. AUTHOR(S)
Cartaya, Christine M., Captain

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending: University of Dayton

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA-91-129

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFIT/CI
Wright-Patterson AFB OH 45433-6583

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distributed Unlimited
ERNEST A. HAYGOOD, Captain, USAF
Executive Officer

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

14. SUBJECT TERMS

15. NUMBER OF PAGES
112

16. PRICE CODE

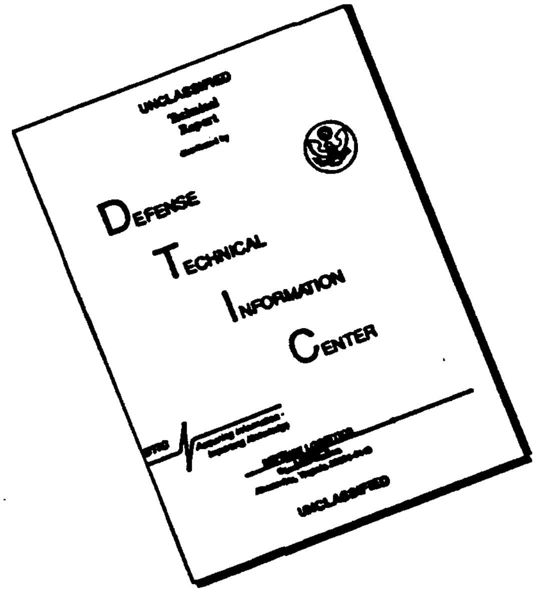
17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

/29

ABSTRACT

COMPUTER AIDED PROCESS PLANNING (CAPP): THE USER INTERFACE FOR THE FABRICATION MODULE OF THE RAPID DESIGN SYSTEM

Name: Cartaya, Christine Marie
University of Dayton, 1991

Advisor: Dr. John P. Eimermacher

The Rapid Design System (RDS) is a feature based design, fabrication, and inspection system. Research and development of the RDS has been sponsored by the United States Air Force. The Fabrication Planning (FP) Module of the RDS is being developed by the University of Dayton. The FP module automatically creates a process plan for machining a metal part. The research addressed herein deals with the User Interface for the review and modification of a process plan.

The Fabrication Planning Module automatically creates a plan using information from the Feature Based Design Environment (FBDE) of the RDS. It integrates this information with the information received from a program/database called MetCAPP™. The process plan is created in a series of steps: feature translation, feature sequencing, setup generation, and operations generation. The purpose of the Planning Window and User Interface is to give a designer, machinists, or process planner the ability to automatically generate and manipulate a complete process plan.

By using the user Interface, the final process plan can be modified in many different ways. The translation of a design feature to a more appropriate MetCAPP™ manufacturing feature can be accomplished. Manufacturing features can be moved between setups, setups can be added or deleted, and machining operations generated by MetCAPP™ can be changed. Finally, when an acceptable plan has been generated, the code to run a Computer Numerically Controlled (CNC) machine is generated.

The research here addresses the process of generating and changing a process plan. Work was done in the area of feature translation, the effect of changing a setup, and the effect of changing the machining operations. At each step in the process, the designer, process planner, or machinist is given the opportunity to accept what has been generated or change it. By allowing changes, the results of both future and prior steps may be affected. Code was written that tracks these affects and regenerates the necessary parts of the plan. This thesis documents the parts of the plan that may be changed, the affects of making such changes, and the code to implement the research results.

ABSTRACT

COMPUTER AIDED PROCESS PLANNING (CAPP): THE USER INTERFACE FOR THE FABRICATION MODULE OF THE RAPID DESIGN SYSTEM

Name: Cartaya, Christine Marie
University of Dayton, 1991

Advisor: Dr. John P. Eimermacher

The Rapid Design System (RDS) is a feature based design, fabrication, and inspection system. Research and development of the RDS has been sponsored by the United States Air Force. The Fabrication Planning (FP) Module of the RDS is being developed by the University of Dayton. The FP module automatically creates a process plan for machining a metal part. The research addressed herein deals with the User Interface for the review and modification of a process plan.

The Fabrication Planning Module automatically creates a plan using information from the Feature Based Design Environment (FBDE) of the RDS. It integrates this information with the information received from a program/database called MetCAPP™. The process plan is created in a series of steps: feature translation, feature sequencing, setup generation, and operations generation. The purpose of the Planning Window and User Interface is to give a designer, machinists, or process planner the ability to automatically generate and manipulate a complete process plan.

By using the user Interface, the final process plan can be modified in many different ways. The translation of a design feature to a more appropriate MetCAPP™ manufacturing feature can be accomplished. Manufacturing features can be moved between setups, setups can be added or deleted, and machining operations generated by MetCAPP™ can be changed. Finally, when an acceptable plan has been generated, the code to run a Computer Numerically Controlled (CNC) machine is generated.

The research here addresses the process of generating and changing a process plan. Work was done in the area of feature translation, the effect of changing a setup, and the effect of changing the machining operations. At each step in the process, the designer, process planner, or machinist is given the opportunity to accept what has been generated or change it. By allowing changes, the results of both future and prior steps may be affected. Code was written that tracks these affects and regenerates the necessary parts of the plan. This thesis documents the parts of the plan that may be changed, the affects of making such changes, and the code to implement the research results.

COMPUTER AIDED PROCESS PLANNING (CAPP) THE USER INTERFACE FOR THE FABRICATION PLANNING MODULE OF THE RAPID DESIGN SYSTEM

Thesis

Submitted to

Graduate Engineering & Research
School of Engineering

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree

Master of Science in Mechanical Engineering

by

Christine Marie Cartaya

UNIVERSITY OF DAYTON

Dayton, Ohio

December 1991

COMPUTER AIDED PROCESS PLANNING (CAPP): THE USER INTERFACE FOR THE FABRICATION PLANNING MODULE OF THE RAPID DESIGN SYSTEM

APPROVED BY:

John Eimermacher, Ph.D.
Advisory Committee, Chairman
Professor, Mechanical and Aerospace Engineering Department

Franklin E. Eastep, Ph.D.
Interim Associate Dean/Director
Graduate Engineering & Research
School of Engineering

Patrick J. Sweeney, Ph.D.
Interim Dean
School of Engineering

ABSTRACT

COMPUTER AIDED PROCESS PLANNING (CAPP): THE USER INTERFACE FOR THE FABRICATION MODULE OF THE RAPID DESIGN SYSTEM

Name: Cartaya, Christine Marie
University of Dayton, 1991

Advisor: Dr. John P. Eimermacher

The Rapid Design System (RDS) is a feature based design, fabrication, and inspection system. Research and development of the RDS has been sponsored by the United States Air Force. The Fabrication Planning (FP) Module of the RDS is being developed by the University of Dayton. The FP module automatically creates a process plan for machining a metal part. The research addressed herein deals with the User Interface for the review and modification of a process plan.

The Fabrication Planning Module automatically creates a plan using information from the Feature Based Design Environment (FBDE) of the RDS. It integrates this information with the information received from a program/database called MetCAPP™. The process plan is created in a series of steps: feature translation, feature sequencing, setup generation, and operations generation. The purpose of the Planning Window and User Interface is to give a

designer, machinists, or process planner the ability to automatically generate and manipulate a complete process plan.

By using the user Interface, the final process plan can be modified in many different ways. The translation of a design feature to a more appropriate MetCAPP™ manufacturing feature can be accomplished. Manufacturing features can be moved between setups, setups can be added or deleted, and machining operations generated by MetCAPP™ can be changed. Finally, when an acceptable plan has been generated, the code to run a Computer Numerically Controlled (CNC) machine is generated.

The research here addresses the process of generating and changing a process plan. Work was done in the area of feature translation, the effect of changing a setup, and the effect of changing the machining operations. At each step in the process, the designer, process planner, or machinist is given the opportunity to accept what has been generated or change it. By allowing changes, the results of both future and prior steps may be affected. Code was written that tracks these affects and regenerates the necessary parts of the plan. This thesis documents the parts of the plan that may be changed, the affects of making such changes, and the code to implement the research results.

ACKNOWLEDGEMENTS

I would like to express my appreciation to the people at the Manufacturing Research section of the Materials Laboratory at Wright Patterson Air Force Base for their support of this project. Especially to Charles Wright for patiently teaching me LISP.

Most importantly, I would like to thank my husband, Ulises, for all his love and support throughout the this research.

VITA

November 14, 1964	Born - Miami, Florida
1986	B.S., University of Miami, Coral Gables, Florida
1986 - 1990	Contracting Officer, Ogden Air Logistics Center, Hill Air Force Base, Utah
1991	M.S., University of Dayton, Dayton, Ohio

FIELDS OF STUDY

Major Field:	Integrated Manufacturing, University of Dayton

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

In traditional manufacturing planning, a machinist or process planner first examines the part specifications or drawings of the part. The planner then develops a plan based on past experience with similar parts and his knowledge of the resources available. Physical differences between parts must be considered, e.g. a similar but slightly larger or smaller part may not fit on the same machine. The capacity of each piece of equipment and the timing of material to the equipment must be taken into consideration. Also, the capabilities of new machines and equipment must be taken into consideration to avoid duplication of effort between machines. Any two process planners, even with similar background, may develop different process plans. This means that even standard plans could be inconsistent and inefficient. To combat this, companies attempt to standardize.

Computer Aided Process Planning (CAPP) is the use of specialized computer systems to aid in process planning. CAPP comes in two forms, variant and generative. Variant systems utilize previously developed process plans which are stored in the computer. Variant systems not only store completed process plans but also the elements which go into their development. Elements such as machine capacities and material specifications, as well as design attributes are coded and stored in the system. Some codes are developed based on the attributes stored, although various types of coding systems are used. The codes used in variant systems classify parts into part families. As new parts are generated a code is created for that part. The system then looks for a previously stored process plan with a similar code. The plan presented can then be used as is or modified. The new plan can then be stored under a new code.

Generative systems, on the other hand, have no previous plans stored. They create a process plan every time a part number is entered into the system. These systems are based mostly on logic rules, design characteristics, formulas, and algorithms. All elements of the manufacturing process (machine types available, machine specification, capacity, material specification, etc.) are programmed into the computer. Parts are coded based on the above elements. As a process plan is needed, the computer searches all the possible combinations based on the code entered and develops a plan.

There are two major components to most generative systems. The first is a geometry based coding scheme to translate part specifications and drawings into computer understandable data. This scheme must be done with great detail in order for the computer to manipulate the data. The coding scheme must also include data on the tools and machines used. This is necessary to insure that parts are put through these machines correctly and in the most efficient manner. The second component is the software. It must compare geometry and specification with the manufacturing capabilities to develop the process plan. Frequently the software includes the ability to print out the plans and provide the codes needed for CNC machines.

Some of the advantages of using computers are reduced labor cost of preparing a process plan, reduced training costs of new planners, and mitigation of the impact of employee turnover. Plan consistency will also improve and CAPP can be a useful tool in improving manufacturing operations. Variant systems, while simpler to create and easier to introduce to a facility, have limitations. As new machines or manufacturing processes are added, the computer database must be manually updated. An expert process planner is also needed to help create and maintain the system. It sometimes requires the referencing of external manuals or charts to modify an existing plan for use.

The advantage of generative process planning is the quick response to change. Since a new plan is generated every time, any change to the manufacturing process plan can be included quickly. With equipment capabilities known to the computer, duplication is reduced. It is also fully automatic which means less human intervention with less chance of error.

Artificial Intelligence (AI) programs or expert systems are sometimes used to develop generative CAPP software. AI programs analyze part geometries based on symbolic representations. They produce a logical sequence of operations to manufacture a part.

taking into account appropriate machine parameters. An alteration of the knowledge base (not the core program) provides an easy way to keep up with changes. They provide the "brains" to a generative system.

Two basic approaches for using AI in process planning are interactive and batch mode. An expert system that works in batch mode processes several jobs submitted in a batch. These systems require or allow little interaction with the process planner. Most process planning systems today run in batch mode. The advantage of batch mode is that it requires less time to run than to interact with the system. The user interface for a batch system only consists of a means to start the planning, display results, and if possible provide an explanation[2]. The disadvantage of batch mode is that the process planner has no capability to modify the plan except by hand. In order for the system to gain knowledge and improve its ability to create plans, changes have to be made in the base line program.

In an interactive system the process planner consults the system for advice (similar to advice from a human expert). The advantage is that through this interaction both the system and the process planner can learn to make better plans. The user interface would consist of several menus or forms with a series of questions or inputs for the process plan to provide. The major disadvantage is that it leads to long planning sessions.[2]

However, no system is capable of providing a "perfect" or "optimal" process plan. Part of the problem is that what is optimal today may not be optimal tomorrow. The process planner needs the capability to either change the parameters of manufacturing knowledge in the system before the process plan is developed or change the process plan itself. This need has been recognized since the first CAPP systems were developed. The variant system developed by Computer Aided Manufacturing International-Inc. (CAM-I) was an interactive system which required no specialized training by the operator.

A generative feature based design system known as Quick Turnaround Cell (QTC) ran in batch mode[2]. The user interface consisted of a design window to create the features and a process planning window. Once activated, the process planning system automatically generates a process plan without input from the user. When complete, process planning documentation is displayed and the CNC code is generated. This system generates process plans relatively quickly but has one major disadvantage. Without user input, any changes to the knowledge in the process planning system has to be changed in the baseline code.

The RDS avoids this problem. It provides both the capability to allow user input to the plan and the ability to automatically generate a process plan through the EAM and the User Interface. Since the EAM can learn from previous design, this flexibility gives the RDS the advantage of being able to improve process planning capability.

The Rapid Design System

The Rapid Design System (RDS) is a United States Air Force in-house research project within the Wright Laboratory at Wright-Patterson Air Force Base, Ohio. The research objective is development of a next generation Computer Aided Design/Manufacturing/Inspection System for machined piece parts.[4] The RDS utilizes a memory driven, feature based design system integrated with an intelligent fabrication and inspection system process planning system. This integration allows the RDS to automatically evaluate a design and generate the fabrication and inspection plans.

The RDS consists of an Episodal Associative Memory (EAM) module, a Feature Based Design Environment (FBDE) module, a Fabrication Planner (FP) module, and an Inspection Planning and Evaluation (IPEM) module.

The EAM uses neural net technology to cluster designs. Along with the ability to store and retrieve designs based on geometric similarity, it has the ability to "learn". As new designs are added, the memory reclusters the designs using the information gathered from the new designs. With these designs clustered in the memory, the EAM can then "learn" what type of designs are good and what are not. It can then prompt the designer, through development of new constraints, when a part has been designed incorrectly.

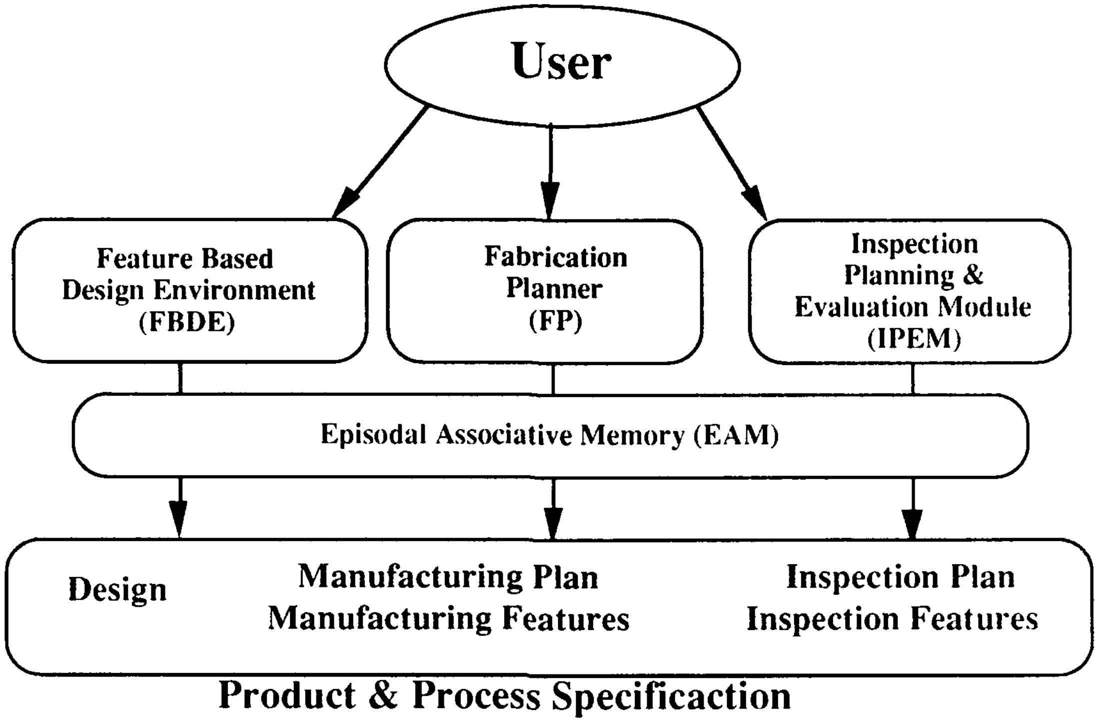


Figure 1. Overall View of the RDS

Traditional Computer Aided Design systems use lines and arcs to describe a part. The RDS through the FBDE uses features. Features such as holes, pockets, and edge cuts, allow the designer to specify relationships among features. These relationships include not only distances between features but also the fact that one feature may be attached to another, such as a hole in the bottom of a pocket. Features allow the FBDE to capture more information about the part than just the dimensions. Information concerning the process, i.e. need to manufacture and inspect the part, are also included.

The FP module produces a manufacturing process plan for machining prismatic parts. A plan is created using the information from the FBDE. The plan is created in a series of steps. Each of these steps, feature translation, feature sequencing, setup generation, and operations generation, can be generated automatically and can be modified. When an acceptable plan has been completed, the FP generates the Automatically Programmed Tool (APT) code to run a Computer Numerical Control (CNC) machine.

The FP module of the RDS system uses a combination of generative and variant process planning. The RDS uses variant process planning in that it can store and retrieve previous parts and their process plans. Parts are stored in a database and can be retrieved by part name or by similar design. However, the RDS uses mostly generative process planning. It has the ability to generate a complete process plan from a FBDE design without any input from the process planner. All elements of the manufacturing process (i.e. available machine types, machine specification, material specifications, feature sequencing, and feature interactions) are programmed into the system. As a process plan is needed, the system uses the design data from the FBDE and machining knowledge and develops a plan. The integration of the FBDE and FP modules allows the designer to check for producibility of a design in the preliminary phases. This can mean less rework and fewer engineering change orders. This also means that rapid prototyping can be accomplished cheaply.

The RDS receives its machining information from a system/database called MetCAPP™. MetCAPP™ is a stand alone CAPP program which generates machining information for a single feature. MetCAPP™ algorithms are based on feature dimensions, machine selection, and material and designed to be used interactively to create a feature by feature process plan for a part. MetCAPP™ makes no consideration for feature sequencing, feature interactions, or the design of the part as a whole. This means that optimization is accomplished done by the designer, planner, or machinist. The RDS uses MetCAPP™ to obtain the machining operations for each feature. These operations include the tool data, spindle speed and feed, and the estimated time to machine the part. Once these operations are obtained, the RDS manipulates the information to create a complete machining process plan.

Statement of the Problem

The initial planning window and user interface for the RDS had two major limitations. The first limitation was that they consisted of only three functions. The first function was the ability to translate design features into manufacturing features with no input from the user. The second function was to group the features into setups based on approach faces only, again, with no input from the user. The third function was to get the machining operations for each feature by calling MetCAPP™. The current user interface and planning window allows the user to interact with the system at each point of the planning process. It also will adjust the outcome to suit current needs.

The second limitation was that planning was very inflexible. The RDS will simply back up to the beginning of the plan after doing a change. If data is changed near the end of the planning process that effects another part of the process above it, the whole plan had to be regenerated (reference Figure 2). It is interesting to note that in recalculating the plan, the change could be erased.

In order to create a workable user interface, two conditions had to be met. First, a planning process had to be developed that was smart enough to allow changes at any step in the process without having to regenerate the plan. Second, this planning process had to automatically regenerate any needed parts of the plan without destroying any changes previously made. These changes could then be stored as part of the design.

Work on the User Interface was divided into two areas. The first was the planning window. This is the visual representation of the manufacturing and design information needed to create a process plan. The second is the user interface itself. The user interface is the functionality behind the planning window. It consists of the algorithms and code used to produce the process plan.

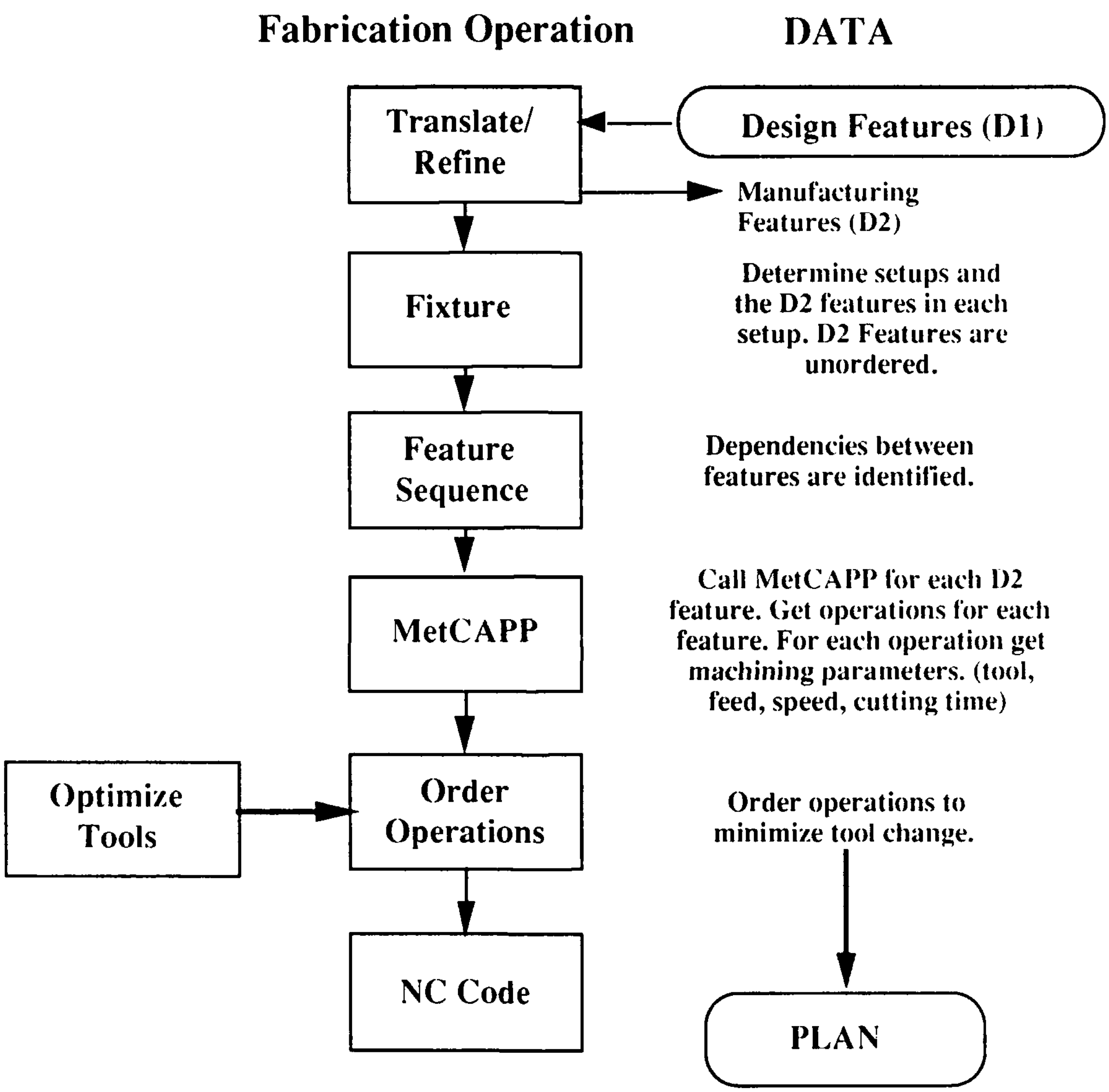


Figure 2. Fabrication Planning Flow

CHAPTER II

PLANNING WINDOW

In developing the planning window, data was collected on the type of information the user needed to interact and develop a complete machining process plan. This information consists of several items. First, information is needed about the part itself. This type of information includes dimensions, part number, material, number and design-with features, and feature sequencing. Second, information is needed about the way in which the part will be manufactured. This information consists of setup fixturing, setup sequencing, and the features to be cut in each setup. Using these two types of information, the machines and processes available to manufacture the part can be determined. This information includes machine availability, operator availability, tool availability, and machining operation details. Machining operation details are operation descriptions (rough cut, drill, etc.), spindle speed, spindle feed, tool identification, operation order, cutting time, and APT/NC code generation.

Once the data is collected, it is grouped by functionality. The main groups of functionality are part identification information (Part Number, Revision No., etc.), feature translation (design to manufacturing), setup data (machine, quantity, etc.), fixturing, machining operations, tool data, and APT/NC code. Once the data is grouped, windows, panes, and icons were developed to display the information. The code for these windows, panes and icons are in Appendix B. Development of the functions and methods for actually creating and editing the plan is part of the user interface. This was developed in conjunction with the planning window.

The Manufacturing Window

The planning window is actually two separate windows. The first window is the manufacturing window (Figure 3). In this window a user, e.g. process planner, has several choices. The user can input the part identification information or generate a completed process plan without any input. The data needed here is received from the FBDE of the RDS and from MetCAPP™. This data includes the identification information, the "design-to-manufacturing" feature translation, setup information, and the fixturing information. Machining operations and tool data are received from MetCAPP™. The user can also generate APT/NC code, modify the "design-to-manufacturing" feature translation, switch to the setups window (discussed below), or store the complete design and manufacturing plan.

As an aid to the user, the ability to draw the wire frame representation, draw the solid representation, and change the graphical view of the design are included. The manufacturing window also includes an area which gives the user a means to view and edit various aspects of the manufacturing features or the machining plan

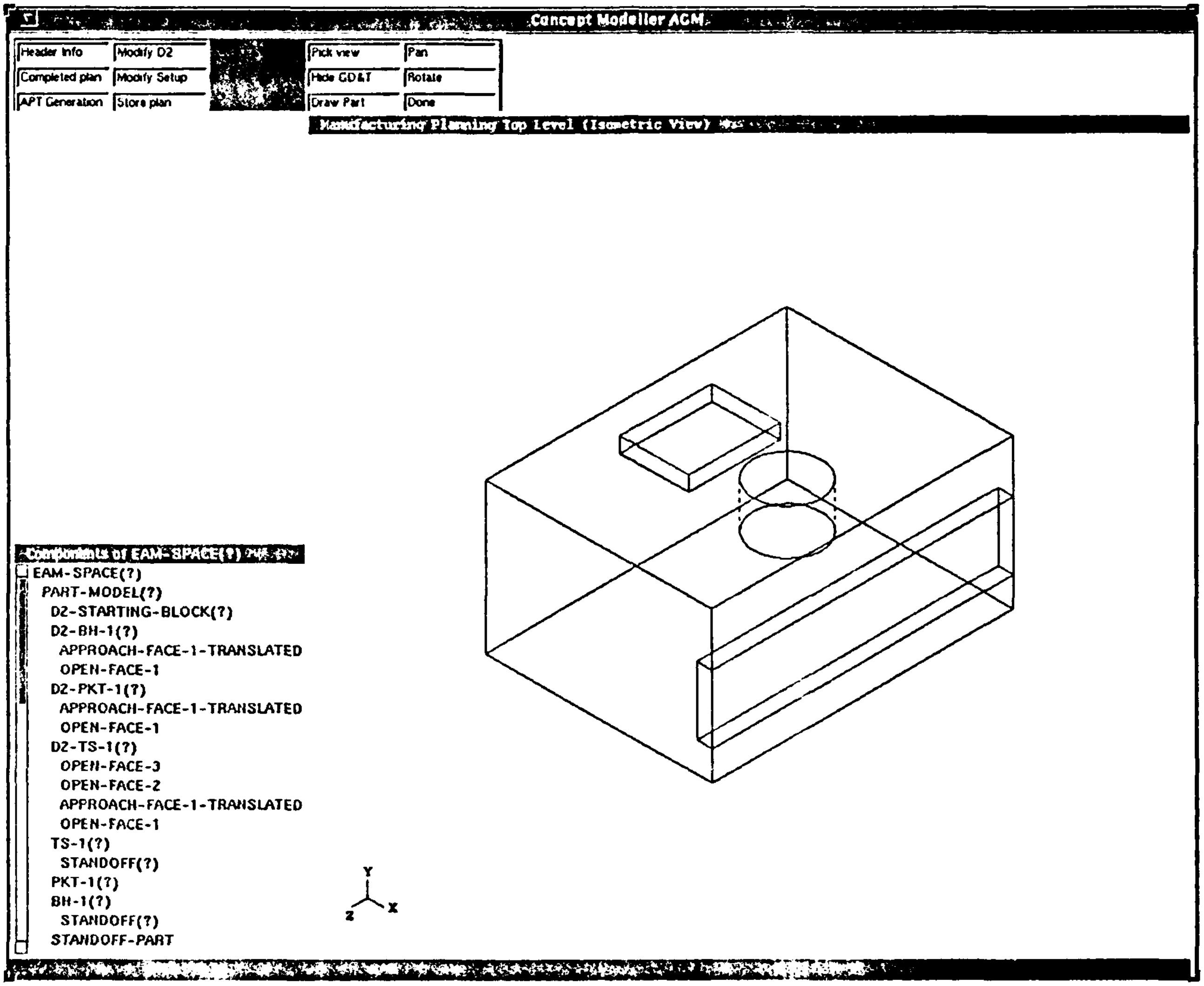


Figure 3. Manufacturing Planning Top Level Window

itself. The ability to generate a completed process plan without guidance from an expert is important. It allows the designer or process planner to check for the manufacturability of the design. If there are problems in the design which prevent the part from being manufactured, the completed plan will show these problems.

Setup Window

The second window is the setup window (Figure 4). It is actually a subpart of the manufacturing window. This window is accessed by clicking on the modify setup icon in the manufacturing window. In this window, the user has nine options. He can add features to a setup, delete features from a setup, add a new setup, delete a setup, generate the manufacturing operations from MetCAPP™, move back to the manufacturing window, view a list of the current setups and the features in each setup, view the surface areas available for the different types of fixturing, or change the part quantity or machine for a specified setup. These abilities allow the user to modify the setups and the individual machining operations to create an optimal process plan.

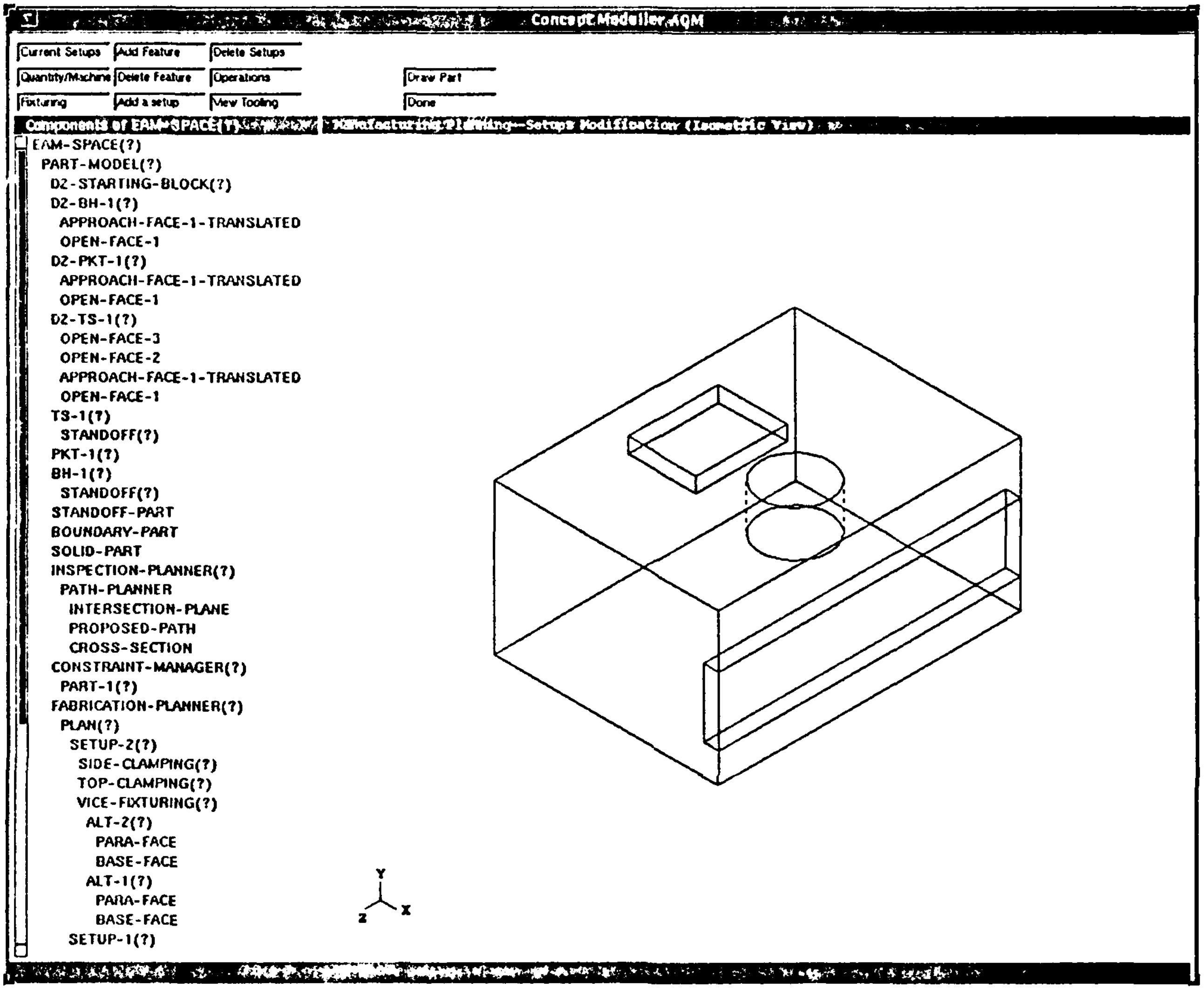


Figure 4. Setups Modification Window

CHAPTER III

THE USER INTERFACE

In developing the user interface, a new flow chart was developed (Figure 5) to highlight that each step of a process plan can be changed and dependant steps are recalculated in any order. The only limitations are the translation of design features into manufacturing features and the generation of APT/NC code. Since feature translation is the basis of the manufacturing planning, it will effect all other steps in the process. Therefore, translation must always be done first. Any changes to the feature translation will cause the rest of the plan to be regenerated. The APT/NC code is the final process to be completed. It depends on all of the previous planning steps. Any change to the previous steps creates a situation where new APT/NC code must be generated.

Feature Translation

In order to create a complete manufacturing process plan, one must first translate the design features into manufacturing features. Design features are used to represent the part graphically. A *manufacturing* feature contains manufacturing information about a part. The manufacturing information includes part dimensions as well as the machining operations needed to manufacture the part. These manufacturing features are the basis for the rest of the machining planning.

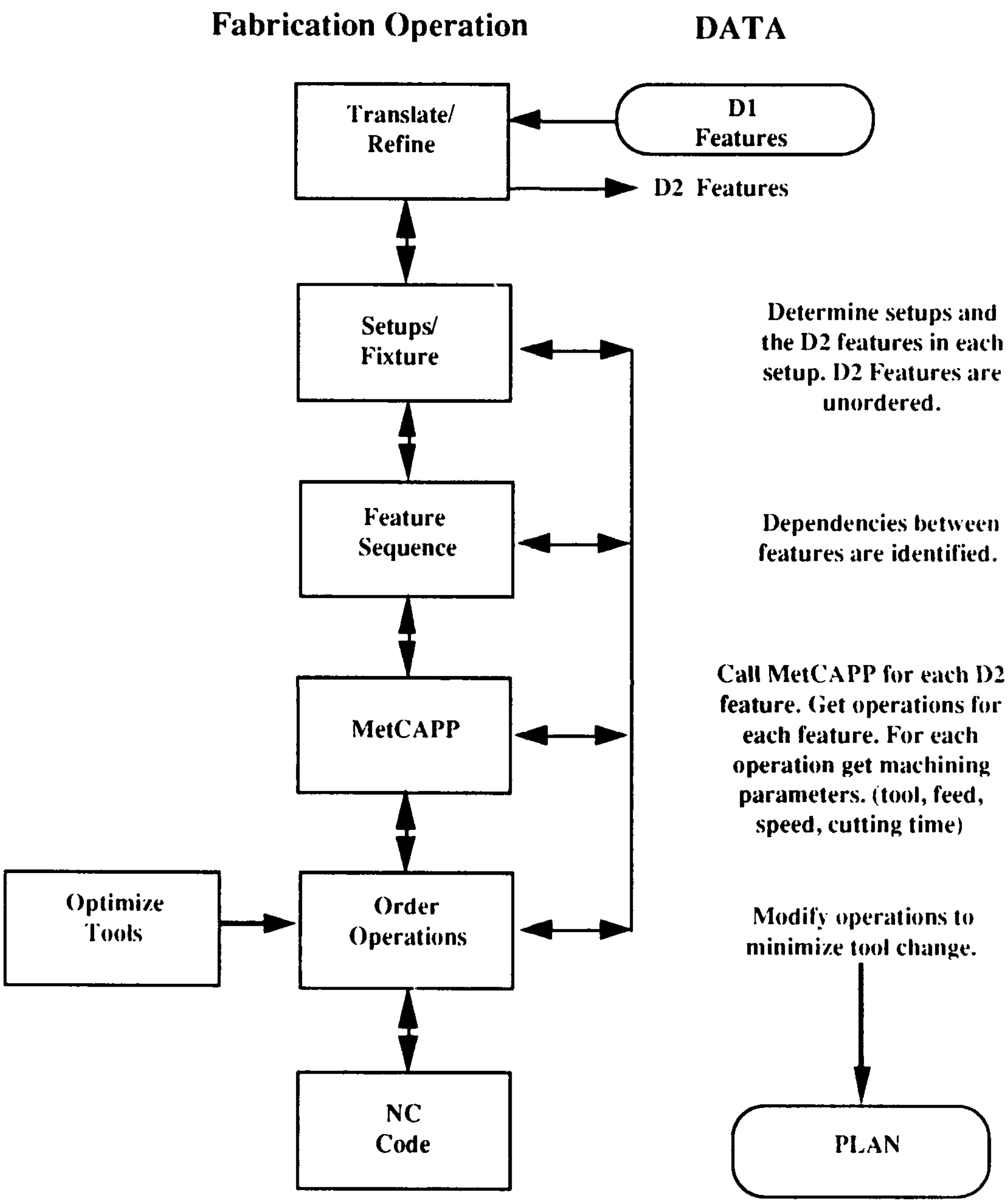


Figure 5. Revised Fabrication Planning Flow

The RDS has separate design and manufacturing features. The manufacturing features in the RDS are the same features used by MetCAPP™ (Figure 6).

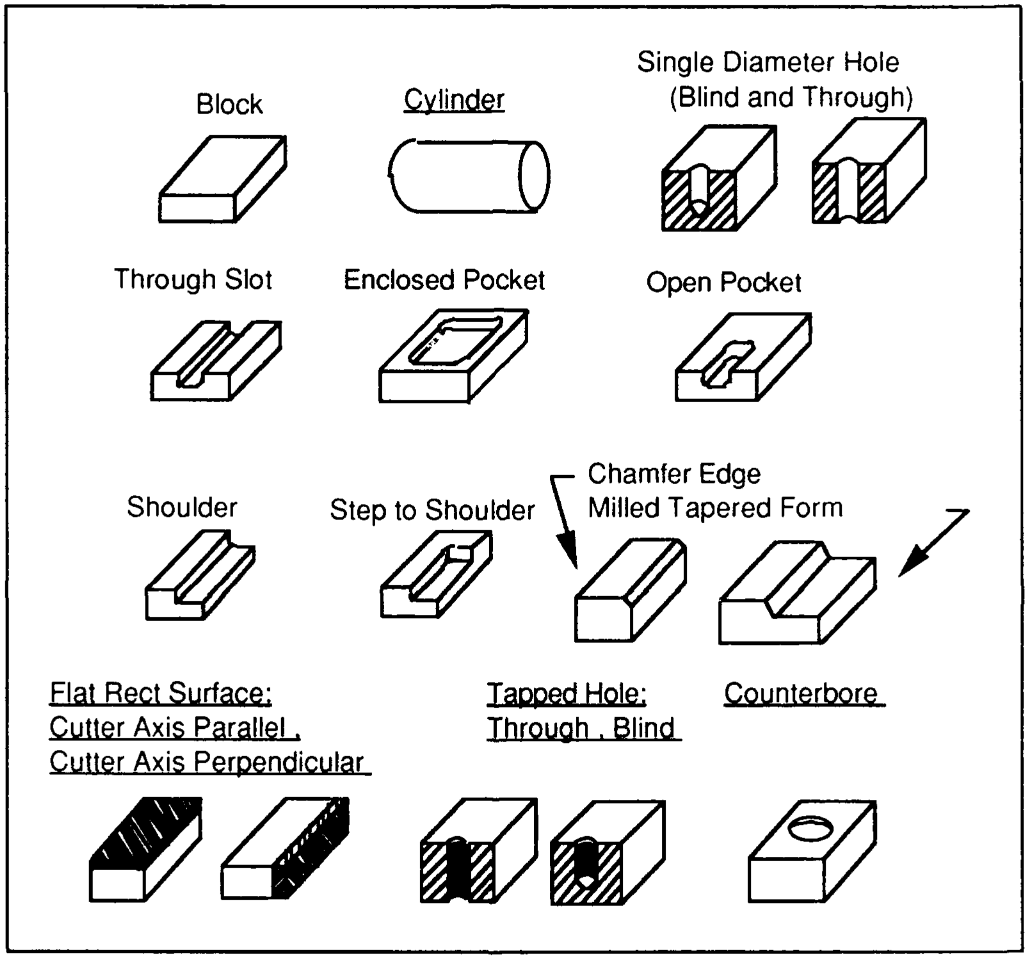


Figure 6. Manufacturing Features

MetCAPP™ is used to obtain machining information for a specific feature. This information includes the machining operation, the tool, the spindle speed and feed, the number of passes the tool will make for each operation, and the time required to cut each operation. This feature translation is done automatically when entering the manufacturing top level window. However, some of the design features do not have a one to one correlation with the MetCAPP™ features. Specifically, there are six untranslated features. These are a hole (either blind or through) with a diameter greater than 2 inches, a triangular pocket, a right triangular pocket, a biased pocket, and a quadrilateral. To use these features, they must be translated as a current MetCAPP™ feature. However, each of these features may be machined.

Each of these features will need to be milled. In order to determine which type of MetCAPP™ features is appropriate for each design feature, a number of tests were run. First, a pocket with the following dimensions was chosen.

Length: 4" Width: 4" Depth: 4"
Corner Radius: 0.1" Fillet Radius: 0.0313"
Angle between floor and wall: 90°
Maximum allowable cutter diameter: 2"

Then, MetCAPP™ was run using these dimensions for a pocket, an open step, a step to a shoulder, a through slot, an open pocket, a flat rectangular surface cutter axis parallel, and a flat rectangular surface cutter axis perpendicular. The operations returned for these features are similar but not identical. The results are in Table 1 and Tables 2 - 7 Appendix A.

Table 1. MetCAPP™ Results for Pocket

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM*	Feed IPM**	Est Time
Center Drill	DLS-001	1	0.097	0.097	5348	37.433	0.003
Drill	Non Insert Drill	1	3.9419	3.9419	6909	35.927	0.190
Plunge End Mill	MLS-0288	1	4.0	4.0	891	3.742	0.988
Slot-end-mill	MLS-0288	2	1.2733	3.82	290	7.98	0.408
Rough-end-mill-floor-wall	MLS-0272	4	1.91	3.82	1141.0	16.77	0.392
Semi-fin-end-mill-wall	MLS-0265	2	1.985	3.97	1454.0	28.79	0.439
Fin-end-mill-floor	MLS-0217	5	0.03	0.03	1273.0	26.73	0.275
Fin-end-mill-wall	Milling Cutter	1	4.0	4.0	19100	30.94	0.293

* IPM = Inches per minute ** RPM = Revolutions per minute

These particular MetCAPP™ features were chosen for trial runs for specific reasons. The first five features were chosen because they are all types of pockets. The difference is the location of the features. The other two features, a flat rectangular surface cutter axis parallel and a flat rectangular surface cutter axis perpendicular, were chosen because they are the basis for all other MetCAPP™ features. Each of these features requires similar input values and return similar operations. The differences arise mainly from the location of the feature.

The first MetCAPP™ test was for a hole (either blind or through) with a diameter of 4 inches. This type of hole can not be translated as a hole because it will be milled not drilled. The hole was modeled as each of the previous mentioned MetCAPP™ features except for a corner radius of 2.0". Results are shown in Table 9 Appendix A. From these results, one can see that a hole can be translated into any of these features and machining operations can be obtained for each. In the case of a blind hole, the machining operations may be used as is. In the case of a through hole, the operations obtained for roughing and finishing the floor of the hole would have to be deleted.

The next MetCAPP™ test was run for a biased pocket. This pocket was modeled as each of the previous mentioned MetCAPP™ features except for an angle between floor and wall of 100^{o}. The results for a pocket are shown in Table 8 in Appendix A. These results show that a biased pocket can be translated into any of these features and machining operations obtained for each.

MetCAPP™ was then run for a triangular pocket, a right triangular pocket, and a quadrilateral pocket (a pocket with four unequal sides and angle between the floor and the wall not equal to 90o). Each of these odd shaped pockets can be modeled as the previously mentioned MetCAPP™ features with certain considerations. A triangular pocket has a smaller inside area than a rectangular pocket with the same height, depth, and width. When using the MetCAPP™ features to represent triangular pockets, care must be taken to use the maximum cutter diameter option. This will return tools with diameters less than or equal to the specified diameter. This option should also be used with quadrilaterals. If the angle between the sides of the pocket are not 90°, then a tool that would fit in a 90° pocket may not fit in a quadrilateral.

The selection of which manufacturing feature to translate the design feature to is defaulted or made by the user during review and edit of the plan. Modification of feature translation is provided in order to give the user maximum flexibility.

Developing Algorithms for Modifying the Process Plan

In creating a process plan the steps in Figure 5 are used. In the RDS system each of these steps are done in any order. However, the modification of any step has a distinct effect on the other steps. None of these steps effect the operations received from MetCAPP™ for each individual feature. The original machining operations received from MetCAPP™ are preserved. In this way the process planner will always have a record of them. The effect of changing each of the planning steps must be looked at separately.

After the translations are complete, setups are generated. The generation of setups includes four areas. These are placing features in a setup, assigning a machine to the setup, assigning the quantity of parts to be done in each setup, and choosing the fixturing for the setup. As each of these areas are modified, the results of the

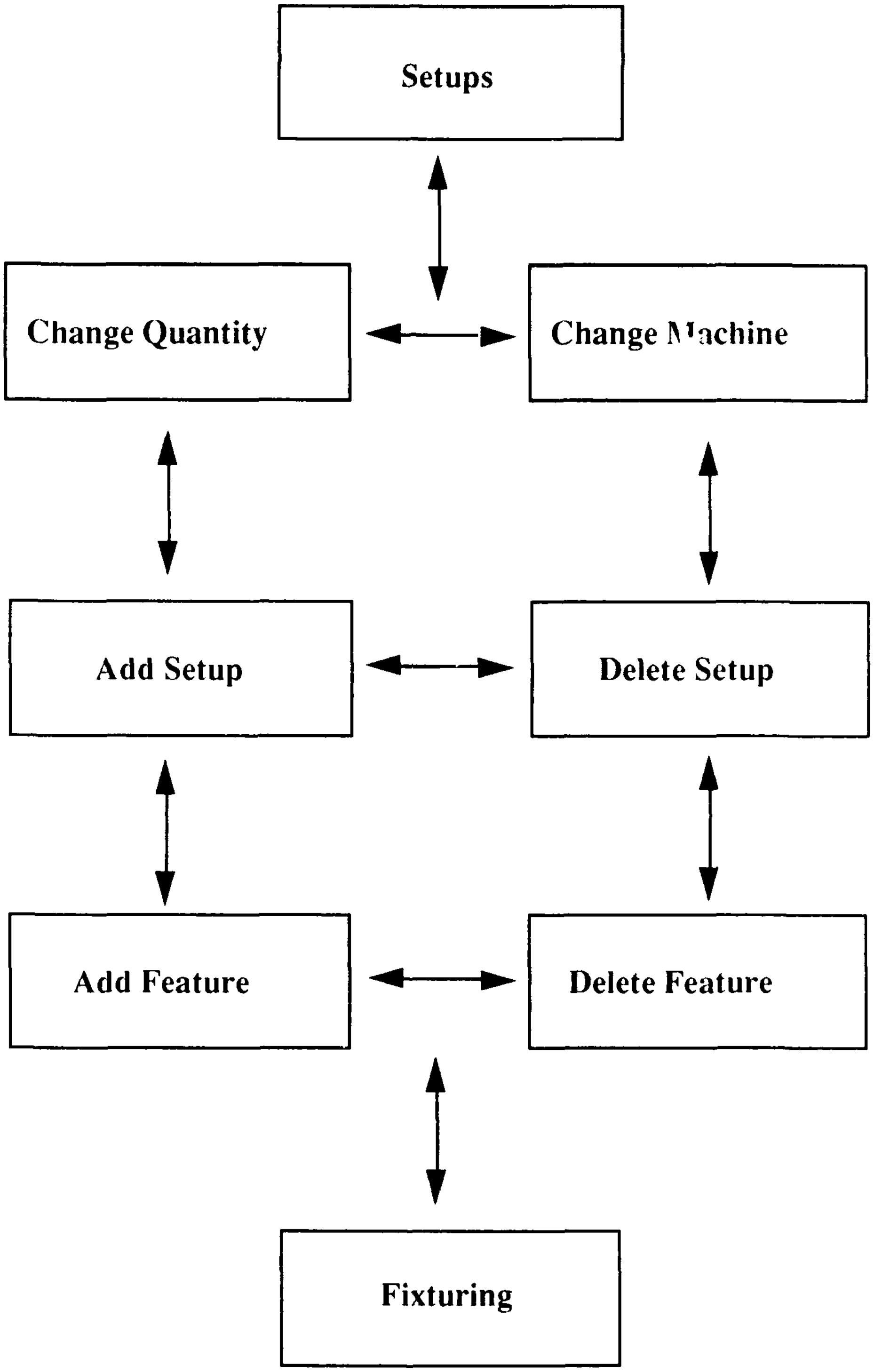


Figure 7. Setups Flow

process plan are changed. Figure 7 shows the overall flow of the process.

The RDS automatically sequences the features within a setup. This sequencing is based on feature interactions and available surface area. The changing of this sequencing can effect the fixturing, operations, and the final APT/NC code generated. If a feature is moved from setup A to setup B, the surface area available is increased in setup A and decreased in setup B. This may change the type of fixturing used for each setup. If the surface area is too small, setup B may now be unstable. If the feature interacts with another feature, moving it to another setup could make the part unmachineable. A hole in the bottom of a pocket may need to be drilled after the pocket has been milled out. If these two features are in setup A and setup B respectively, setup A would have to be machined before setup B. The operations generated for the setup will also change. The operations for the feature will be deleted from setup A and integrated into the operations for setup B.

The quantity of parts machined in a setup is an important factor in choosing the machine for a setup. If the user changes the quantity of parts done in one setup, then the machines capable of handling that setup are recalculated. This recalculation includes changing the machine sent to MetCAPP™ for each feature. The user can also change the machine used for a particular setup. To do so, the user is given two lists of machines. The first is a list of machines capable of handling the setup for a given quantity of parts. The second is a list of machines capable of handling the setup based solely on the size of the part. This allows maximum flexibility for the user. The results of this change will effect the tool selection, fixturing type, or APT/NC code for that particular setup.

Fixturing information is generated for top clamping, side clamping, and alternative vice fixturing. The surfaces available for each type of fixturing is displayed graphically. When choosing the fixturing for each setup, consideration should be made for stability of the part. Once fixturing is decided, the profile of the setup is complete.

Once the setups are established, the machining operations are generated. The machining operations for each feature in a setup are based on MetCAPP™ recommendations. The effect of modifying these operations must now be considered (Figure 8). The user can now modify the machining operations by deleting selected ones, reordering them, modifying the tools, speeds, or feeds. If an operation is deleted, then the final dimensions and "look" of the feature could be affected. The APT/NC code generated depends on the machining operations. If a plunging or roughing operation is deleted and the plan generates the APT/NC code, tools could be broken or injures could occur. Therefore, the only operations that can be deleted are semi-finish and finish operations. These operations only effect the final tolerance and finish of the features.

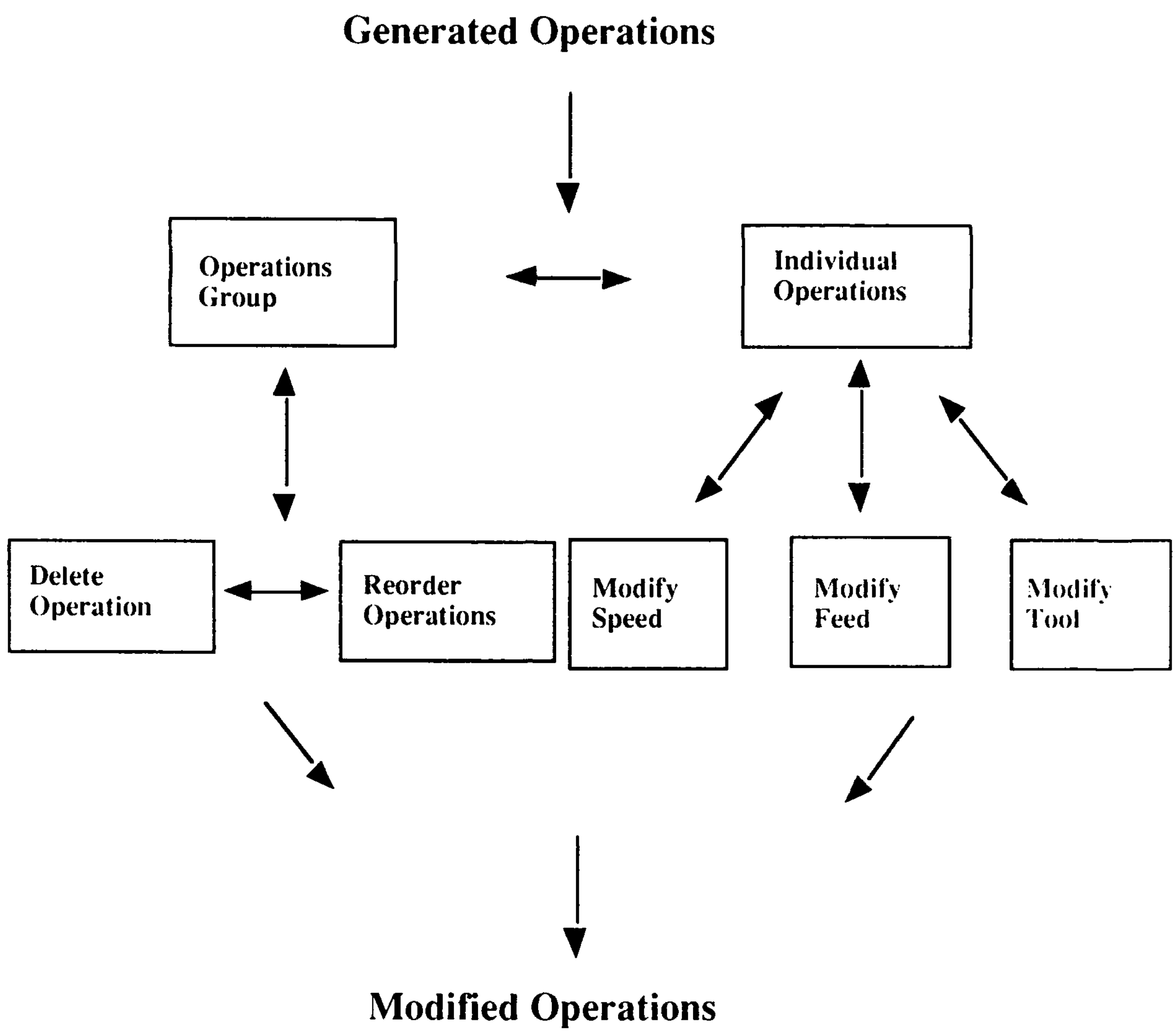


Figure 8. Modifying the Operations

Initial ordering of machining operations is accomplished automatically by the RDS. When reordering the operations, some safety considerations must be made. Roughing operations for a particular feature must always come before finishing operations. However, the finishing operation for one feature may or may not come before the roughing operation of another feature. Care must be taken when editing the order. If roughing and finishing operations are ordered incorrectly, tools could be broken and injuries could occur.

When modifying the speeds or feeds for an operation one important consideration should be noted. The interface to MetCAPP™ consists only of extracting the operations and tool data. This means that changing the speed will not automatically recalculate the feed or vice versa its done in MetCAPP™. Speed and feed modifications must therefore be done with care.

A tool may be changed for a particular operation. This could be done to minimize the number of tools used or because of tool availability. If the tool is modified, the information for the new tool is generated and replaces the information for the old tool.

By allowing the user to interact and modify the process plan as needed, the system becomes more user friendly. Each step in the process plan has distinct affects on the other steps. These must be considered carefully when developing an optimum process plan.

CHAPTER IV

USING THE USER INTERFACE

The code developed here is written in LISP and involved the use of the Concept Modeller™ (CM). The Concept Modeller™ is a parametric design system developed by Wisdom Systems. It is a Lisp based system which provides a tool kit of object oriented functions. It provides functions for defining features, methods, and user interface functions. A complete listing of the code can be found in Appendix B. Partial listings of the code will be illustrated as needed.

The Layouts, Forms, and Icons for the Planning Window

The planning window consists of two layouts, the manufacturing top level layout and the setups modification layout. Each of these layouts is a Concept Modeller™ layout which consists of several panes (Figure 9).

```
([illegible]-configuration 'ui-layout-man-plan '((TREE-PANE [illegible]
[illegible] 62[illegible]19 0.25 1.0 WHITE STEEL-BLUE)
     (GRAPHICS DISPLAY-PANE-FLAVOR 0.25 0.0843373493975903[illegible] 1.0 [illegible]
     (MESSAGE-PANE MESSAGE-PANE-FLAVOR 0.418848167539267 0.0 1.[illegible] [illegible]
WHITE STEEL-BLUE)
     (INSPECTOR SCROLLING-INSPECT-PANE-FLAVOR 0.0 0.0843373493975903[illegible] 0.2[illegible]
[illegible].54[illegible]90[illegible]33[illegible]62822[illegible]19 WHITE STEEL-BLUE)
     (FBDE-CONTROL-PANE FORM-FLAVOR 0.0 0.0 0.418848167539267 0.0843[illegible]
WHITE STEEL-BLUE)))

([illegible]-layout '[illegible]-layout-man-plan '(ui-layout-man-plan (
          ([illegible]-CONTROL-PANE MAN-PLAN))))
```

Figure 9. Sample Code for a Layout

A pane is a designated area of the layout. There are different types of panes available for use. The types used here are the graphics pane, message pane, tree pane, inspection pane, and the form pane. The first four panes are Concept Modeller™ defined panes and require no further definition than to name them.

The form pane requires a specified form. The purpose of of which is to display the icons. There are many different types of icons available in the Concept Modeller™. The ones used in here are property icons, multiple choice icons, editor icons, and command icons. A property icon is used to display and modify the property of a specified object or feature. A multiple choice icon is a group of icons where one of the icons can be chosen for use. An editor icon displays a string of data for display or editing. A command icon is used to fire a predefined function.

The manufacturing top level layout consists of all five of the above panes. The form used for its form pane has eleven command icons. These icons are labeled Completed Plan, Modify D2, Modify Setups, APT Generation, Header Info, Draw Part, Rotate, Pan, Pick View, Store Plan, Hide GD&T (Geometric Dimensioning and Tolerancing), and Done. The functionality behind the first five will be discussed under the user interface section. Draw Part allows the user to draw the wire frame, wire frame with hidden lines removed, or the solid part. Rotate, Pan, Pick View, and Hide GD&T effect the orientation of the part on the screen.

The setups modification layout consists of a graphics, tree, message, and form pane. The form attached to the form pane has the same function as for the manufacturing top level form but, consists of eleven command icons. These icons are Current Setups, Fixturing, Quantity/Machine, Add Feature, Delete Feature, Add Setup, Delete Setup, Operations, View Tooling, Draw Part, and Done. The Current Setups icon displays the number of setups and the features in each setup. The View Fixturing icon provides a list of fixturing types and graphically displays the surface area available for each type of fixturing per setup.

Modifying the Process Plan

The user interface allows the user to create a complete process plan with or without any input. In order to do this several options are available. To create a process plan automatically, i.e. without any human input, the user selects the Completed Plan icon in the Manufacturing Top Level layout. This icon uses several functions which call MetCAPP™ by setup, sequences the MetCAPP™ operations, and displays the complete plan by setup.

In order to call MetCAPP™, each feature must be associated to a machine. Machine selection is based on the size of the starting block. Once the machine is chosen, the dimensions and material are obtained from the manufacturing feature. Using this information, the operations, speeds, feeds, tool, and estimated cutting time are generated from MetCAPP™. These operations are then sequenced by setup. Once this is complete the final plan is displayed. The plan can then be saved to a file for printing, along with the tool data.

In general, modifications to the plan are needed for optimization. This can be done either before generating the completed plan or after. One of the first modifications that is needed is to enter the header information. This is done by selecting the Header Info icon (Figure 10). The header information is needed to identify the plan for storage and retrieval. It consists of the part number, the job control number, the JOCAS number, the name of the planner, the name of the lead technician who will be making the part, the priority of the part run, the scheduler, the issue date of the process plan, the due date, and the customer. The engineer's and planner's name are taken from the login name since they are the users of the system. Each piece of information is a property of the plan. They are entered and edited by property icons.

If the user wishes to change the "design-to-manufacturing" translation, the icon Modify D2 is provided. (Figure 11) It displays the list of design features and their corresponding manufacturing features. If there is no direct translation for a design feature then NIL is displayed. To change the translation, the user selects the feature pair. The system then provides a list of manufacturing features that the chosen design feature can be translated into. Once a manufacturing feature is selected, the user is returned to the "design-to-manufacturing" feature list display.

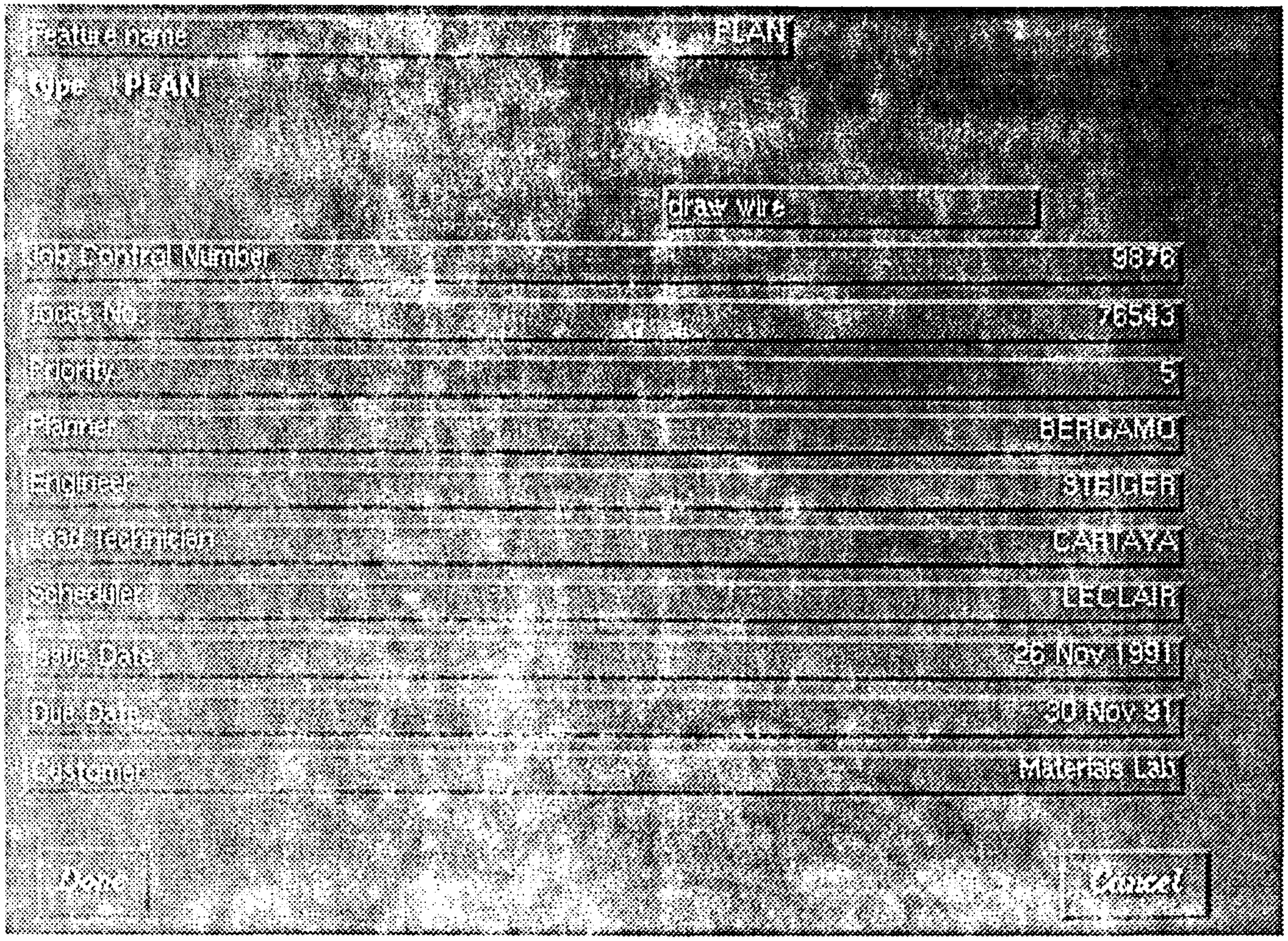


Figure 10. Header Information Screen

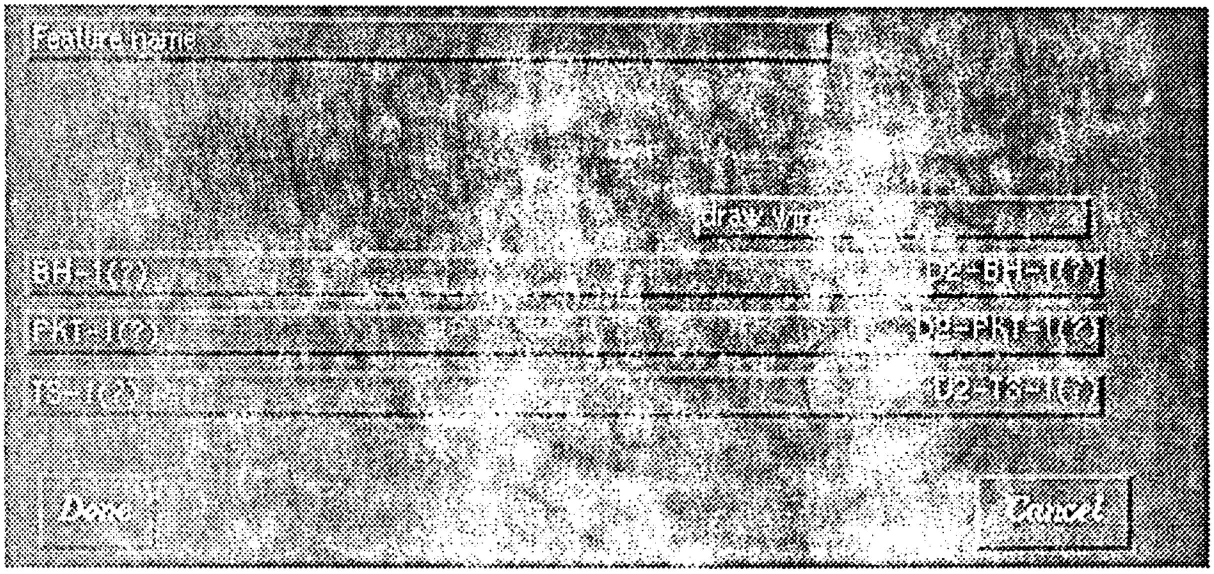


Figure 11. Modify D2 Screen

After completing the desired changes, the user is given the choice of Done or Cancel. If the user chooses Cancel, the "design-to-manufacturing" feature translation remains unchanged. If the user selects Done, the translation is changed according to the users inputs. The functions used here are written for each design feature type. For each feature type, the function deletes the current manufacturing feature, if it exists. It then translates the design feature to the user specified manufacturing feature. When translating very different types of features, such as a hole into a pocket, special mapping of some the dimensions to accommodate the translation. In the instance of a hole to a pocket, the diameter of the hole is mapped to the length and width of the pocket. The radius of the hole is then mapped to the corner radius of the pocket. These dimension changes are done automatically according to the type of feature being translated.

Both of the previously mentioned edits are top level changes. What if the user wishes to change part of the process plan itself? This is done in the Setups Modification layout. To enter that layout, the user selects the Modify Setups icon. In this layout, the user has several options. In order to view the list of setups and the features in each setup, he can select the Current Setups icon. This icon calls a function, view-setups, which finds the features for each setup and displays them (Figure 12).

```
(defun view-setups ()
  (pop-up-message (format nil "Here is a list of setups and the features in each
setup~%~a" (setups-string))
                  :button-list '(Continue))
  )

(defun setups-string (&aux setup-list)
  (setf setup-list nil)
  (dolist (it (select :type 'setup-face) setup-list)
        (setf setup-list (append setup-list
                                 (list (format nil "~a~%" (get-print-name it))) (features-
string (the a2-features (:from it)))))
         )
  )

(defun features-string (features &aux features-list)
  (setf features-list nil)
  (dolist (it features features-list)
        (setf features-list (append features-list (list (format nil "~a~%" it))))
        )
  )
```

Figure 12. Sample Code for Viewing the Current Setup

If the user does not like the setups, he can change them. To add a feature to a setup, the user selects the Add Feature icon. The RDS then displays the current features not in the setup. The user chooses the feature to add to the setup. If the feature chosen is correct, the system proceeds to create a new setup with the requested features. The previous setups are not destroyed. This allows the user to compare setups to choose the optimum one for his needs.

If a feature is to be deleted from a setup, the Delete Feature icon is selected. The system displays the current features in the setup and the user chooses one to delete. The system then creates a new setup with the revised list of features. Again, the previous setup is not destroyed.

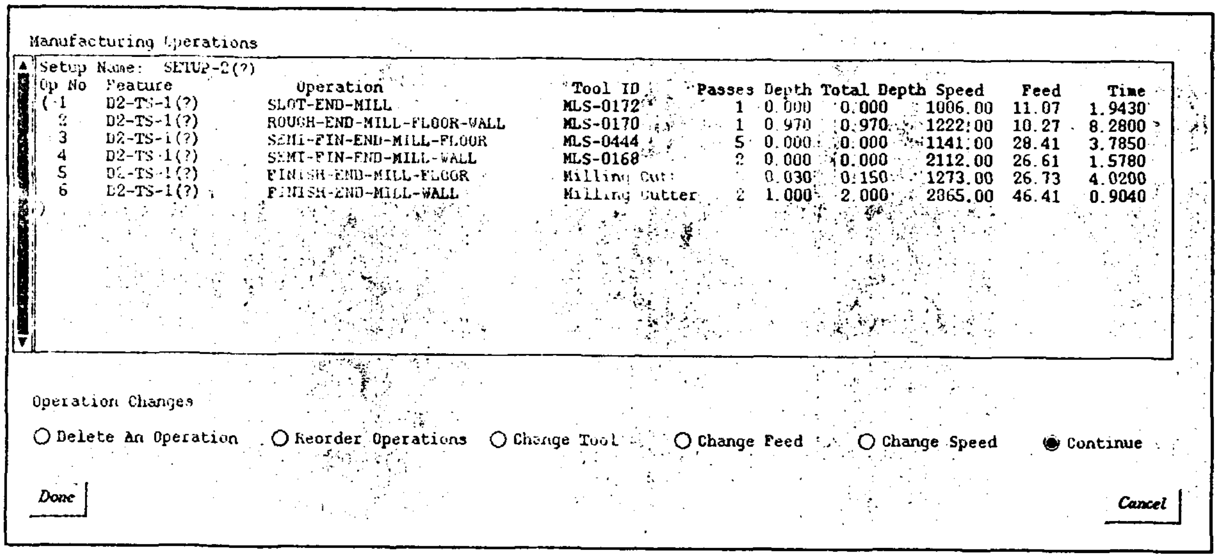
Add Feature and Delete Feature only change the composition of a setup one feature at a time. If the user wishes to move several features between setups, a new setup can be created. This is done by selecting the Add Setup icon. The system then asks for the number of features to be included in the setup. Once that number is chosen, the system cycles through the list of available manufacturing features, allowing the user to select one feature at a time. If the final list of chosen features are correct, the system creates a new setup with the selected features, leaving the current setups intact. This allows the user to do "what if" scenarios. When creating the new setup, the system checks to make sure that there is adequate surface area to fixture the setup. If there is not, then the features are divided into the least number of setups that are needed to adequately fixture the part.

Once the setups are created, the machine associated with that setup can be changed. The machine is chosen based on two criteria, part size and setup quantity. The setup quantity is the quantity of parts to be run in one setup. In order to change either the setup quantity or the machine, the user selects the Quantity/Machine icon. This icon generates a form with two property icons, quantity and machine. The user can select either one to modify. If the user chooses to modify the quantity, that modification could change the machine selection. Two different tables are used for machine selection. One is based on a quantity greater than or equal to seven. As the quantity changes, the machine selection also changes.

The user may choose to change the machine manually. In this case the user is given the choice of two machine lists. The first is a list of machines capable of machining the part based on the quantity of parts in a setup. This list is ordered by preference of the current user of the system, 4950th Test Wing, a unit of the Air Force which does manufacturing for aircraft modifications. The second is a list of machines capable of machining the part based on size of the part only. These two lists are used to allow the user the flexibility to choose the machine that best fits the current situation. Once the machine is chosen, the machine property is changed for that particular setup as well as for each feature in a setup.

With the setups created, the surface area available for fixturing can be displayed. Fixturing area is done for vice fixturing, top clamping, and side clamping devices. In order to view the surface area, the user chooses the View Fixturing icon. The system then displays a list of fixturing options to be viewed. These include side clamping, top clamping, and two alternatives for vice clamping with parallel or base face. The parallel face is the area that the vice clamp touches. The base face is the face that the part sits on in that setup when fixtured using vice clamps. Once the fixturing option is chosen, the surface area is displayed in different colors in the graphics pane.

Once a particular setup is chosen, the operations to machine the setup can be generated. This is done by choosing the Operations icon. The Operations icon calls a function which requests the operations for all the features in that setup. A call to MetCAPP™ is made for each feature. The resulting operations are then sequenced



Manufacturing Operations

Setup Name: SETUP-2(?)

Op No	Feature	Operation	Tool ID	Passes	Depth	Total Depth	Speed	Feed	Time
1	D2-TS-1(?)	SLOT-END-MILL	MLS-0172	1	0.000	0.000	1006.00	11.07	1.9430
2	D2-TS-1(?)	ROUGH-END-MILL-FLOOR-WALL	MLS-0170	1	0.970	0.970	1222.00	10.27	8.2800
3	D2-TS-1(?)	SEMI-FIN-END-MILL-FLOOR	MLS-0444	5	0.000	0.000	1141.00	28.41	3.7850
4	D2-TS-1(?)	SEMI-FIN-END-MILL-WALL	MLS-0168	2	0.000	0.000	2112.00	26.61	1.5780
5	D2-TS-1(?)	FINISH-END-MILL-FLOOR	Milling Cut		0.030	0.150	1273.00	26.73	4.0200
6	D2-TS-1(?)	FINISH-END-MILL-WALL	Milling Cutter	2	1.000	2.000	2365.00	46.41	0.9040

Operation Changes

○ Delete An Operation ○ Reorder Operations ○ Change Tool ○ Change Feed ○ Change Speed ● Continue

Done Cancel

Figure 13. Manufacturing Operations by Setup

and displayed on screen (Figure 13). Once these operations are displayed, the user has six options displayed in a multiple choice icon. Only one option can be chosen at a time. This is to reduce confusion when modifying the operations. The user can delete an operation, reorder the operations, modify a tool, modify a speed, modify a feed value, or continue. If the user chooses to continue, he in effect accepts the plan without modification. In order to delete an operation, the user selects the delete operation option. Then the Done icon is selected. The system then displays a list of numbers corresponding to the number of operations. The planner chooses a number and the corresponding operation is displayed. If that is the operation to be deleted, the system deletes it. If not, the system returns to the operations form.

When reordering the operations, the user is again given a list of numbers corresponding to the number of operations. However, this list is in a more editable form. It enables the user to reorder the numbers as desired. Once the reordering is completed, the new order of operations is displayed. If this is acceptable, the operations are changed within the setup itself. This reordering does not effect the MetCAPP™ order of operations within a feature. The original operations order is preserved and can be regenerated if needed in the future.

In order to modify a tool, the operation that contains that tool must first be selected. The RDS displays a list of numbers identical to the ones used in deleting an operation. However, this time the tool in a selected operation is then displayed. The user can then edit the tool as desired. There is one problem with editing the tool. If you change tools and select a tool that is not in the MetCAPP™ tool database, you will get no information back about it. This could be a problem when trying to identify or obtain the correct tool for an operation from a tool crib. Once the tool is edited, it is changed in the setup operation property and in the property of each feature.

Editing the speed and feed values of individual machining operations is similar to editing the tools. The difference is that the values of speed and feed are changed in the feature properties only; because in the setup the property contains only the list of features, operations, and tools. Once all modifications to the operations have been done, the planner can choose Continue to leave the operations form.

The tooling information is provided for two reasons. The first reason is to give the user the needed information to select the correct tool for each operation. The second is to give the machinist the needed information to retrieve the tool from the tool crib. The tooling information generated can be displayed on the screen and is saved to a file for printing when the plan is complete. In order to display the tooling information, the Tool Data icon is chosen. This icon calls a function to check what type of tool is used and then displays the information based on that type. The information is obtained by a call to MetCAPP™. There are 22 different types of tools available in the MetCAPP™ tooling database. This calls for 22 different formats for displaying the information (Figure 14).

```
(defun non-ins-mil-cut (tool-data)
  (format nil "Drawing Number:                   ~a
Cutting Diameter:             ~a
Diameter Designation:         ~a
Effective Flute Length:       ~a
Overall Length:               ~a
Cutter Body Type:             ~a
Cutting Angle:                ~a
Coolant Feeding:              ~a
Number of Flutes:             ~a
Cutter End Type:              ~a
Nose Style:                   ~a
Nose Radius Size:             ~a
Nose Flat Angle:              ~a
Hand of Cut:                  ~a
Helix Direction:              ~a
Helix Angle:                  ~a
Shank or Drive Type:          ~a
Shank/Pilot Diameter:         ~a
Tool Material:                ~a
Tool Material Class:          ~a
Tool Material Grade:          ~a
Tool Material Construction:   ~a
Land Type - Relief:           ~a
Land Width:                   ~a
Radial Rake Angle:            ~a
Primary Clearance Angle:      ~a
Secondary Clearance Angle:    ~a
Neck Diameter:                ~a
Neck Length:                  ~a
Application Code:             ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
          (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data))
  )
```

Figure 14. Sample Code for Non Insertable Milling Cutter

The format is chosen based on the type of tool. The information is displayed for each different tool used in a setup. The information can be done by setup or for the whole part. Each tool is displayed one at a time (Figure 15).

Here is your tool data
(Tool ID: DLS-009
Drawing Number: DLS-1
Cutter Body Diameter: 25.5
Diameter Designation: 25.5
Effective Axial Cutting Length: 25.5
Minor/Pilot Diameter: 0.0
Minor/Pilot Dia Designation:
Effective Length: 0.0
Overall Length: 150.0
Cutter Body Type: EP
Coolant Feeding: N
Number of Flutes: 2
Margin Width: 0.0
Cutter Point Type: EA
Point Grind Included Angle: 90.0
Web Thickness: 0.0
Hand of Cut: RH
Helix Angle: 30.0
Shank Type: SS
Shank Diameter: 25.5
Tool Material: M7
Tool Material Class: A
Tool Material Grade: M7
Tool Material Construction: S
)

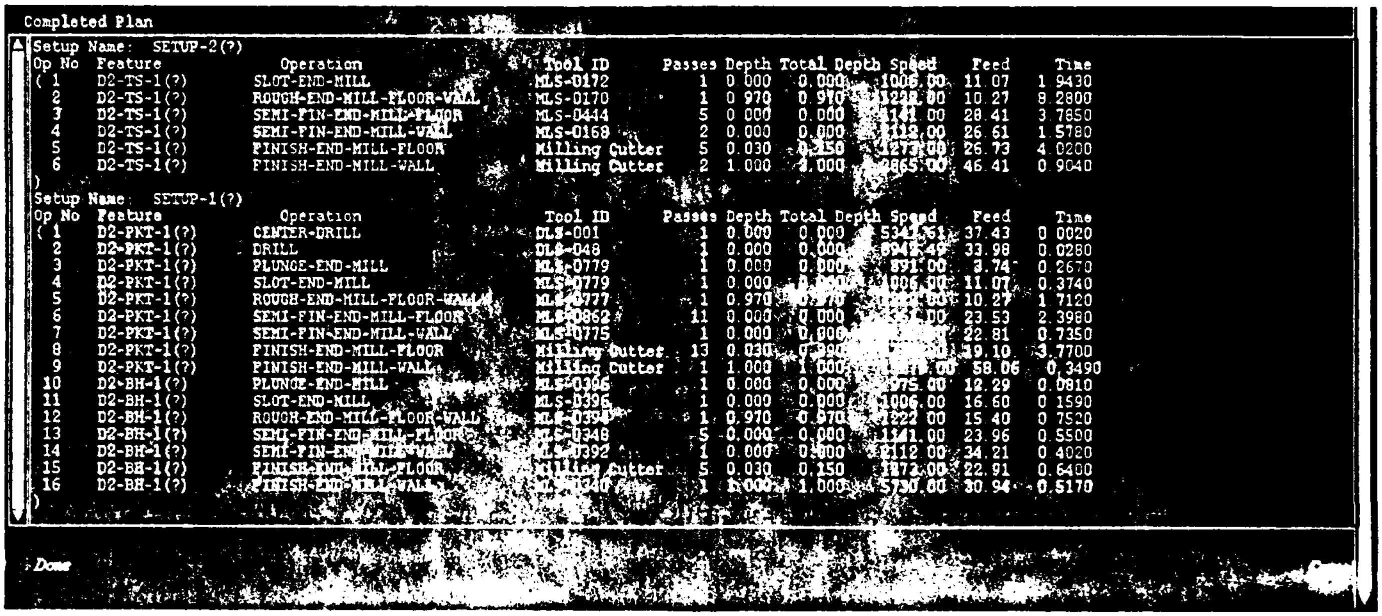
CONTINUE

Figure 15. Tooling Information Screen

Once all the modifications to the setups have been done, the user can exit the Setup Modification window and return to the Manufacturing Top Level window, by selecting the Done icon. But before the user can exit, a check is made to ensure that each feature is used in one and only one setup. If a feature is used in multiple setups or not used at all a warning message is displayed and the user is left in the Setup Modification window. Extra setups can be deleted by choosing the Delete Setup icon. The user simply chooses the setup and it is deleted. By providing this check the user is forced to ensure that no feature is duplicated or left out.

When the final process plan is completed, the user has several options. First, the user can review the plan on the screen, (Figure 16) again using the Competed Plan icon. Second, the user can print out the plan. In order to print out the plan, the user must have chosen the Completed Plan icon and stored the plan to file. The planner can then simply print that file (Figure 17). Third, the user can generate the APT/NC code for an individual setup or the complete plan. To generate the APT/NC code, the planner must choose the APT generation icon. The system then generates the APT/NC code and saves it to a file. Lastly, the user can store the design and the process plan in a database for future use.

This last option is important. The Episodal Associative Memory portion of the RDS allows the user to bring up designs



Completed Plan

Setup Name: SETUP-2(?)

Op No	Feature	Operation	Tool ID	Passes	Depth	Total Depth	Speed	Feed	Time
(1	D2-TS-1(?)	SLOT-END-MILL	MLS-0172	1	0.000	0.000	1006.00	11.07	1.9430
2	D2-TS-1(?)	ROUGH-END-MILL-FLOOR-WALL	MLS-0170	1	0.970	0.970	1222.00	10.27	8.2800
3	D2-TS-1(?)	SEMI-FIN-END-MILL-FLOOR	MLS-0444	5	0.000	0.000	1141.00	28.41	3.7850
4	D2-TS-1(?)	SEMI-FIN-END-MILL-WALL	MLS-0168	2	0.000	0.000	2112.00	26.61	1.5780
5	D2-TS-1(?)	FINISH-END-MILL-FLOOR	Milling Cutter	5	0.030	0.150	1273.00	26.73	4.0200
6	D2-TS-1(?)	FINISH-END-MILL-WALL	Milling Cutter	2	1.000	2.000	2865.00	46.41	0.9040

)

Setup Name: SETUP-1(?)

Op No	Feature	Operation	Tool ID	Passes	Depth	Total Depth	Speed	Feed	Time
(1	D2-PKT-1(?)	CENTER-DRILL	DLS-001	1	0.000	0.000	5341.61	37.43	0.0020
2	D2-PKT-1(?)	DRILL	DLS-048	1	0.000	0.000	[illegible]	33.98	0.0280
3	D2-PKT-1(?)	PLUNGE-END-MILL	MLS-0779	1	0.000	0.000	891.00	3.74	0.2670
4	D2-PKT-1(?)	SLOT-END-MILL	MLS-0779	1	0.000	0.000	1006.00	11.07	0.3740
5	D2-PKT-1(?)	ROUGH-END-MILL-FLOOR-WALL	MLS-0777	1	0.970	0.970	[illegible]	10.27	1.7120
6	D2-PKT-1(?)	SEMI-FIN-END-MILL-FLOOR	MLS-0862	11	0.000	0.000	[illegible]	23.53	2.3980
7	D2-PKT-1(?)	SEMI-FIN-END-MILL-WALL	MLS-0775	1	0.000	0.000	[illegible]	22.81	0.7350
8	D2-PKT-1(?)	FINISH-END-MILL-FLOOR	Milling Cutter	13	0.030	0.390	[illegible]	19.10	3.7700
9	D2-PKT-1(?)	FINISH-END-MILL-WALL	Milling Cutter	1	1.000	1.000	[illegible]	58.06	0.3490
10	D2-BH-1(?)	PLUNGE-END-MILL	MLS-0396	1	0.000	0.000	975.00	12.29	0.0810
11	D2-BH-1(?)	SLOT-END-MILL	MLS-0396	1	0.000	0.000	1006.00	16.60	0.1590
12	D2-BH-1(?)	ROUGH-END-MILL-FLOOR-WALL	MLS-0394	1	0.970	0.970	1222.00	15.40	0.7520
13	D2-BH-1(?)	SEMI-FIN-END-MILL-FLOOR	MLS-0348	5	0.000	0.000	1141.00	23.96	0.5500
14	D2-BH-1(?)	SEMI-FIN-END-MILL-WALL	MLS-0392	1	0.000	0.000	2112.00	34.21	0.4020
15	D2-BH-1(?)	FINISH-END-MILL-FLOOR	Milling Cutter	5	0.030	0.150	1272.00	22.91	0.6400
16	D2-BH-1(?)	FINISH-END-MILL-WALL	MLS-0340	1	1.000	1.000	5730.00	30.94	0.5170

)

Done

Figure 16. Complete Process Plan

JON NIL	RDS MANUFACTURING ORDER								Page of
RDS Planner: NIL	Programmer: WHO-KNOWS			Lead Tech: NIL					
Project Name/Drawing Number: PART-NO		Material: T-7075			Sequence No:				

Machine/ Oper No/Date	Description of Work	Tool ID	No. of Passes	Depth of Cut	Total Depth	Speed RPM	Feed IPM	Est Time	Act Time
PRATT-WHITNEY 1 /27 Nov 1991	SLOT-END-MILL D2-TS-1(?)	MLS-0172	1	0.000	0.000	1006.00	11.07	1.9430	
PRATT-WHITNEY 2 /27 Nov 1991	ROUGH-END-MILL-FLOOR-WALL D2-TS-1(?)	MLS-0170	1	0.970	0.970	1222.00	10.27	8.2800	
PRATT-WHITNEY 3 /27 Nov 1991	SEMI-FIN-END-MILL-FLOOR D2-TS-1(?)	MLS-0444	5	0.000	0.000	1141.00	28.41	3.7850	
PRATT-WHITNEY 4 /27 Nov 1991	SEMI-FIN-END-MILL-WALL D2-TS-1(?)	MLS-0168	2	0.000	0.000	2112.00	26.61	1.5780	
PRATT-WHITNEY 5 /27 Nov 1991	FINISH-END-MILL-FLOOR D2-TS-1(?)	Milling Cutter	5	0.030	0.150	1273.00	26.73	4.0200	
PRATT-WHITNEY 6 /27 Nov 1991	FINISH-END-MILL-WALL D2-TS-1(?)	Milling Cutter	2	1.000	2.000	2865.00	46.41	0.9040	
PRATT-WHITNEY 1 /27 Nov 1991	CENTER-DRILL D2-PKT-1(?)	DLS-001	1	0.000	0.000	5347.61	37.43	0.0020	
PRATT-WHITNEY 2 /27 Nov 1991	DRILL D2-PKT-1(?)	DLS-048	1	0.000	0.000	8942.49	33.98	0.0280	
PRATT-WHITNEY 3 /27 Nov 1991	PLUNGE-END-MILL D2-PKT-1(?)	MLS-0779	1	0.030	0.000	891.00	3.74	0.2670	
PRATT-WHITNEY 4 /27 Nov 1991	SLOT-END-MILL D2-PKT-1(?)	MLS-0779	1	0.000	0.000	1006.00	11.07	0.3740	
PRATT-WHITNEY 5 /27 Nov 1991	ROUGH-END-MILL-FLOOR-WALL D2-PKT-1(?)	MLS-0777	1	0.970	0.970	1222.00	10.27	1.7120	
PRATT-WHITNEY 6 /27 Nov 1991	SEMI-FIN-END-MILL-FLOOR D2-PKT-1(?)	MLS-0862	11	0.000	0.000	3361.00	23.53	2.3980	
PRATT-WHITNEY 7 /27 Nov 1991	SEMI-FIN-END-MILL-WALL D2-PKT-1(?)	MLS-0775	1	0.000	0.000	[illegible]112.00	22.81	0.7350	
PRATT-WHITNEY 8 /27 Nov 1991	FINISH-END-MILL-FLOOR D2-PKT-1(?)	Milling Cutter	13	0.030	0.390	47[illegible]5.00	19.10	3.7700	
PRATT-WHITNEY 9 /27 Nov 1991	FINISH-END-MILL-WALL D2-PKT-1(?)	Milling Cutter	1	1.000	1.000	15[illegible]79.00	58.06	0.3490	
PRATT-WHITNEY 10 /27 Nov 1991	PLUNGE-END-MILL D2-BH-1(?)	MLS-0396	1	0.000	0.000	975.00	12.29	0.0810	
PRATT-WHITNEY 11 /27 Nov 1991	SLOT-END-MILL D2-BH-1(?)	MLS-0396	1	0.000	0.000	1006.00	16.60	0.1590	
PRATT-WHITNEY 12 /27 Nov 1991	ROUGH-END-MILL-FLOOR-WALL D2-BH-1(?)	MLS-0394	1	0.970	0.970	1222.00	15.40	0.7520	
PRATT-WHITNEY 13 /27 Nov 1991	SEMI-FIN-END-MILL-FLOOR D2-BH-1(?)	MLS-0348	5	0.000	0.000	1141.00	23.96	0.5500	
PRATT-WHITNEY 14 /27 Nov 1991	SEMI-FIN-END-MILL-WALL D2-BH-1(?)	MLS-0392	1	0.000	0.000	2112.00	34.21	0.4020	
PRATT-WHITNEY 15 /27 Nov 1991	FINISH-END-MILL-FLOOR D2-BH-1(?)	Milling Cutter	5	0.030	0.150	1273.00	22.91	0.6400	
PRATT-WHITNEY 16 /27 Nov 1991	FINISH-END-MILL-WALL D2-BH-1(?)	MLS-0340	1	1.000	1.000	5730.00	30.94	0.5170	

Figure 17. Printed Process Plan

similar to the one currently in the system based on similar geometry. Having the manufacturing information stored as part of that design gives the user the ability to use the previous information as part of the new design.

Each of these modifications can be made in any order. The demand driven nature of the Concept Modeller™ means that if information is needed from one property to generate another property, the first property will be generated in order to satisfy the needs of the second property. In terms of the user interface, this means that when viewing or modifying one part of the plan, all other needed parts of the plan are generated. Any changes to these parts are automatically displayed.

CHAPTER V

TRACKING THE PLANS AND USERS

Tracking the Plan

In order to track the initial and modified versions of the process plan, the operations are stored in three ways. First, the initial operations received from MetCAPP™ for each feature is stored by feature under the property Operations. This group of unsequenced operations is left unaltered. These operations are then sequenced by setup. Second, the sequenced operations are stored under each individual setup in the property called operations-sequence-list. If the operations is modified in any way, the operations-sequence-list is modified. Finally, the modified operations are also stored under each feature in the property NC-ops-list. This is the property which the APT/NC operations methods use to generate the APT/NC code.

By storing the plan in this way, manufacturing knowledge is now a part of the design. The system knows not only the initial plan it generated, but also the modified plan the user created. If later a part is run again, there is now a choice of using the initial generated plan, the modified plan, or creating a new plan depending on the circumstances. Having several iterations of the plan available also allows the EAM to begin to reason about process planning. As plans are modified, the EAM will be able to use the modifications to generate better plans in the future.

Tracking the Users

For the RDS, as for many other systems, it is important to both control access to the system and maintain a log of who is using the system. This is done by requiring each user to have a login ID. This ID is then stored as a global variable in the system. Once the user logs in and generates a process plan, his ID and name are stored as part of the plan.

In order to protect designs and process plans, the amount of access to the system a user has depends on the permissions that are associated with the ID. Users classified for read, modify, and write permissions. Read permission gives you access to view a completed design and process plan. It does not allow you to generate, modify, or store plans. Write permission gives you full control over generating, modifying, and store a process plan. Modify permission does the same as write except you can not store the plan. It allows the user to play "what if"scenarios without permanently changing a process plan. This permission scheme was developed in this way in order to allow junior level process planners to use the system without being able to overwrite a plan deemed acceptable by the senior level process planners.

CHAPTER VI

CONCLUSION

In developing the user interface for the RDS, two major problems were evident. First, the interface had to be user friendly. Many process planners and machinists are unfamiliar with computer aided process planning or even computers at all. But some may be quite computer literate. By providing a menu driven interface that allows direct access through user command, both types of users are satisfied.

Second, the interface had to be flexible. No CAPP program, whether variant or generative, can produce a process plan without any human interaction. At least, not without becoming quickly outdated. This means that, for the RDS to avoid this problem, the option to change had to be present at every point in the planning process. But allowing this degree of flexibility causes its own problems. Since the planning process is not always done in a serial manner, careful accounting had to be done in order to keep track of how a modification to one part of the plan effected the overall plan. Checks were built in that eliminated many of these problems, however, this is an issue which needs to be monitored carefully.

By allowing the user to create a plan either interactively or automatically, the benefits of both worlds are realized. Automatic generation allows the user to generate plans quickly. It can also aid the designer in ensuring that the designs created are manufacturable. By using the interactive mode, knowledge gained from the modifications can be captured by the EAM. This will allow the RDS to improve its process planning capabilities.

CHAPTER VII

RECOMMENDATIONS

As the RDS continues to expand, many parts of the user interface will have to be changed. More work should be done in feature translation. Each of the design feature needs to have a direct translation to a manufacturing feature. But the option to change that translation needs to be maintained. This will allow the user to tailor the responses from MetCAPP™ to his present needs.

Work should be done is regard to the EAM. The manufacturing knowledge is not being fully captured in the RDS. Some work has been done on using the EAM to do feature sequencing[11]. However, the EAM is not being fully utilized with regard to manufacturing. While some manufacturing constraints have been coded into the system, they are primarily the constraints imposed by MetCAPP™. Using the EAM to train and capture manufacturing constraints as they are developed through use of the RDS would be a great advantage. Future users of the system would then have the benefit of past experience (both the problems and successes) of previous planners.

The EAM should also be used to cluster on parameters other then just design features. The EAM could retrieve designs based on manufacturing features, feature interactions, setups, or entire plans. Since manufacturing and design features differ, allowing the

EAM to retrieve on manufacturing features may provide a different set of similar parts then retrieving on design alone. Process planners may also want to retrieve designs based on similar process plans and the problems encountered. The EAM could be queried based on any number of user defined parameters.

The flexibility of the user interface could also be enhanced. Right now there are several input parameters to a process plan that are defaulted. These include such things as material specification, tolerancing, and other inputs to MetCAPP™. Material specification is currently defaulted to T-7075 aluminum. This can be changed by the user but not easily. There are also up to 16 different parameters which can be fed into MetCAPP™. Currently, only the dimensions, material, machine, and tolerance are used. Other parameters, such as setup rigidity, thin wall conditions, and required cutter radius are defaulted. These parameters could be included in the user interface to allow the user to retrieve operations which best suit the current situation.

Tolerancing information is not currently being used to its fullest potential. Default tolerancing of single features is being passed to MetCAPP™. The user can change that tolerance when creating or modifying a feature. Geometric Dimensioning and Tolerancing (GD&T) information is also available through the use of GD&T features. This gives you positional tolerancing and tolerancing between features. But this type of information is not used by MetCAPP™, therefore it is not considered in creating the process plan. Incorporating this information into the process plan would increase the potential usefulness of the RDS.

Flexibility in storing the plans could also be increased. A new way to store plans created by the junior process planners is needed. Currently, only the plans created by the senior level process planners are stored. This means that if a junior planner creates an acceptable plan, the plan must be recreated by a senior planner in order to be stored. One way of doing this may be to create a temporary storage location for plans created by junior planners. If these plans are approved for usage, a senior level planner could move them over to the permanent database which the EAM accesses.

APPENDIX A

Table 2. MetCAPP™ Results for Open Step

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Rough-end-mill-floor-wall	MLS-0272	8	1.985	3.970	768	13.824	3.472
Fin-end-mill-floor	MLS-0224	3	0.03	0.03	764	13.752	1.308
Fin-end-mill-wall	MLS-0224	1	4.0	4.0	1432	33.509	0.179

Table 3. MetCAPP™ Results for Step to a Shoulder

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Rough-end-mill-floor-wall	MLS-0272	8	1.985	3.97	781	14.527	4.208
Semi-fin-end-mill-wall	MLS-0265	2	1.985	3.97	1898	52.385	0.304
Fin-end-mill-floor	MLS-0217	4	0.03	0.03	1358	24.444	1.300
Fin-end-mill-wall	Milling Cutter	1	4.0	4.0	3820	44.312	0.181

Table 4. MetCAPP™ Results for Through Slot.

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Slot-end-mill	MLS-0288	2	1.985	3.97	670	8.04	1.492
Rough-end-mill-floor-wall	MLS-0272	4	1.985	3.97	775	14.183	1.692
Fin-end-mill-floor	MLS-0224	3	0.03	0.03	955	28.65	0.627
Fin-end-mill-wall	MLS-0224	2	4.0	4.0	1432	33.509	0.358

Table 5. MetCAPP™ Results for Open Pocket

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Center Drill	DLS-001	1	0.097	0.097	5348	37.433	0.003
Drill	Non Insert Drill	1	4.0	4.0	6909	35.927	0.192
Slot End Mill	MLS-0288	2	1.985	3.97	670	8.040	0.950
Rough-end-mill-floor-wall	MLS-0272	4	1.985	3.97	833	16.743	0.912
Semi-fin-end-mill-wall	MLS-0265	2	1.985	3.97	1898	52.385	0.410
Fin-end-mill-floor	MLS-0217	5	0.03	0.03	1698	50.94	0.390
Fin-end-mill-wall	Milling Cutter	1	4.0	4.0	3820	44.312	0.254

Table 6. MetCAPP™ Results for Cutter Axis Parallel

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Rough-end-mill-floor-wall	MLS-0272	8	.990	2.970	1141.0	16.77	3.472
Fin-end-mill-wall	MLS-0224	1	3.0	6.0	19100	30.94	0.179

Table 7. MetCAPP™ Results for Cutter Axis Perpendicular

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Rough-end-mill-floor	MLS-0272	4	.990	2.970	1141.0	16.77	2.616
Fin-end-mill-floor	MLS-0224	3	3.0	6.0	19100	30.94	1.308

Table 8. MetCAPP™ Results for Biased Pocket

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Center Drill	DLS-001	1	0.097	0.097	5348	37.433	0.003
Drill	Non-insert-drill	1	3.9419	3.9419	6909	35.927	0.190
Plunge-end-mill	MLS-0288	1	4.0	4.0	891	3.742	0.988
Slot-end-mill-wall	MLS-0288	2	1.2733	3.82	290	7.98	0.408
Rough-end-mill	MLS-0272	4	1.91	3.82	1141.0	16.77	0.392
Semi-fin-end-mill-wall	MLS-0265	2	1.985	3.97	1454.0	28.79	0.430
Fin-end-mill-floor	MLS-0217	5	0.03	0.03	1273.0	26.73	0.275
Fin-end-mill-wall	Milling Cutter	1	4.0	4.0	19100	30.94	0.293
Rough-conical-mill-form	Conical Mill	2	2.0	4.0	710.0	24.708	0.158
Finish-conical-form-mill	Conical Mill	1	4.0	4.0	1222	41.059	0.195

Table 9. MetCAPP™ Results for Hole with Diameter Greater Than 2 Inches

Operation	Tool	No Passes	Cut Depth	Total Depth	Speed RPM	Feed IPM	Est Time
Plunge-end-mill	MLS-0288	1	4.0	4.0	668	8.016	0.499
Slot-end-mill	MLS-0288	2	1.985	3.97	670	8.04	0.482
Rough-end-mill-floor-wall	MLS-0272	4	1.985	3.97	775	14.183	0.548
Fin-end-mill-floor	MLS-0224	3	0.03	0.03	955	28.65	0.204
Fin-end-mill-wall	MLS-0224	1	4.0	4.0	1432	33.509	0.239

APPENDIX B

THE CODE

Object Definitions

Fabrication Planner

```
[illegible] fab-planner-type
  :[illegible] (masking non-graphic mutable-part feature-mixin [illegible])
  :[illegible]
                display? nil
                display-masked-parts? nil
                d1-d2-feature-pair    nil
                )
  :[illegible]-form-definition (d1-d2-feature-pair)
  )
```

Plan

```
; [illegible] from object, not non-graphic-object
[illegible] plan
  :[illegible] (masking non-graphic mutable-part object feature-[illegible])
  :[illegible]
                 sequence          nil
                 quantity          [illegible]
                 machine           ([illegible]-use-machines (true))
                 approach-[illegible]       nil
                 job-control-number  nil
                 [illegible]         nil
                [illegible]          nil
                planner              nil
                engineer             nil
                 [illegible]         nil
                  [illegible]        nil
                 [illegible]-date    ([illegible]-date)
                [illegible]          nil
                 [illegible]         nil
                display? nil
                display-masked-parts? nil
                feature-list nil
                starting-object nil
                setup-list-feature (gen-setups-sequence 'feature-[illegible])
                setups
                )
  :[illegible]-form-definition (('cwru-property
                              label "Job Control Number"
```

```
left-action '(change-jcn (the part-model (active-
planner plan))
 name 'joo-control-number
 top  '*y*
 width 450)
('cwru-property
 label "Jocas No."
 left-action '(change-jocas-no (the part-model (active-
planner plan))
 name 'jocas-no
 top  '*y*
 width 450)
('cwru-property
 label "Priority"
 left-action '(change-priority (the part-model (active-
planner plan))
 name 'priority
 top  '*y*
 width 450)
('cwru-property
 label "Planner"
 left-action '(change-planner (the part-model (active-
planner plan))
 name 'planner
 top  '*y*
 width 450)
('cwru-property
 label "Engineer"
 left-action '(change-engineer (the part-model (active-
planner plan))
 name 'engineer
 top  '*y*
 width 450)
('cwru-property
 label "Lead Technician"
 left-action '(change-lead-technician (the part-[illegible]
[illegible] planner plan))
 name 'lead-technician
 top  '*y*
 width 450)
('cwru-property
 label "Scheduler"
 left-action '(change-scheduler (the part-model (active-
planner plan))
 name 'scheduler
 top  '*y*
 width 450)
('cwru-property
 label "Issue Date"
 left-action '(change-issue-date (the part-[illegible]
[illegible] planner plan))
 name 'issue-date
 top  '*y*
 width 450)
('cwru-property
 label "Due Date"
 left-action '(change-date (the part-model (active-
planner plan))
 name 'due-date
 top  '*y*
 width 450)
('cwru-property
 label "Customer"
```

```
                     left-action '(change-customer (the part-[illegible]
[illegible] plan))
                     name 'customer
                     top  '*y*
                     width 450))
 )
```

Setup

```
([illegible] feature setup-face
  :inherit-from  (masking non-graphic mutable-part object feature-[illegible])
  :properties (
               display? nil
               display-masked-parts? nil
               direction nil
               starting-object nil
               feature-list-geom (mapcar '(lambda (a) (the geom [illegible]
[illegible])
               fixturing-list (fixturing-setup-by-feature-pointers [illegible]
[illegible] feature-list-geom)
                feature-face      nil
                operations        nil
                operations-sequence-list (colon-sequence-the-[illegible]
                a2-features       nil
                sequenced         nil
                intersecting      nil
                machine           (first ^machine)
                [illegible]       nil
                [illegible]       nil
               quantity          ^quantity
                tool-change-point '(-3 -3 6)
                tool-list        (colon-get-tool-data (the))
               )
  :[illegible]-form-definition (('cwru-property
                     label "Quantity"
                     name 'quantity
                     left-action '(colon-change-quantity [illegible]
[illegible]))
                     top '*y*)
                    ('cwru-property
                     label "machine"
                     left-action '(change-machine)
                     name 'machine
                     top  '*y*))
 .
```

The Manufacturing Top Level Window

Plan Manufacturing

```
[illegible] (colon-plan-manufacturing command-icon) [illegible]
"[illegible] manufacturing planning environment and generate [illegible]"
[illegible] ([illegible] pick-icon-args))
[illegible] (name '[illegible]-layout-man-plan)
```

```
([illegible]-pane-label (get-pane-named 'graphics) "Manufacturing Planning Top Level")
(colon-[illegible]-context (the part-model) 'fbde-design)
([illegible]-context (the part-model) 'fbde-goal)
([illegible]-context (the part-model) 'man-planning)
([illegible] (select :type 'geometric-feature-mixin))
        ([illegible]-add-d2-feature it))
(when (string= (pop-up-message "Create new plan?" :button-list '("Yes" "No"))
              "Yes")
      ([illegible]-remove-all-plans (the eam-space part-model fabrication-planner))
      (colon-add-plan (the eam-space part-model fabrication-planner))
      (generate-all-setups))
(colon-clear-and-redraw-wire (the part-model))
(mapcar #'draw-part (select :type 'open-face-cyl))
(mapcar #'draw-part (select :type 'open-face-rect))
(mapcar #'draw-part (select :type 'triangular-face))
```

Manufacturing Layout

```
[illegible] 'ui-layout-man-plan '(
    ([illegible]-PANE-FLAVOR 0.0 0.5490533562822719 0.25 1.0 [illegible])
    ([illegible]-PANE-FLAVOR 0.25 0.08433734939759036 1.0 [illegible] YELLOW [illegible])
    ([illegible] MESSAGE-PANE-FLAVOR 0.41884816753926 0.0 1.0 0.08[illegible]
    [illegible])
    ([illegible]NG-INSPECT-PANE-FLAVOR 0.0 0.08433734939759036 0.25
  [illegible] WHITE STEEL-BLUE)
    ([illegible]-PANE FORM-FLAVOR 0.0 0.0 0.41884816753926 0.08[illegible]
  [illegible])))

 [illegible] 'ui-layout-man-plan '(ui-layout-man-plan (
        [illegible]-PANE MAN-PLAN))))
```

Manufacturing Plan Form

```
[illegible] 'FORM-OBJECT name 'MAN-PLAN file "/users/ros/ros/ui-[illegible]"
          left 0 top 0
          edit-settings (copy-list '((GRID-SIZE . 10) (DISPLAY-GRID [illegible]) ([illegible] .
[illegible]))
          [illegible]-action 'FORM-OPERATIONS
          [illegible]-action nil
          [illegible]-action nil
          [illegible]-ALIST (QUOTE ((:LIGHT-[illegible] . [illegible])
                                      (:DARK-[illegible] . [illegible])
                                      (:BACKGROUND . CYAN)
                                      (:PURPOSE :LIGHT-[illegible] : [illegible]
[illegible])))
          [illegible]-PURPOSE-ALIST (QUOTE ((:LABEL :normal :normal :NORMAL) ([illegible]
[illegible] :NORMAL)))

            [illegible] '(
                ('[illegible]-COMMAND LABEL "Done"
                                  LEFT-ACTION (QUOTE colon-[illegible])
                                  MIDDLE-ACTION NIL
                                  MOUSE-DOCUMENTATION-STRING " [illegible]
  [illegible]"
                                  NAME (QUOTE DONE)
                                  BORDER-[illegible]AW-[illegible]
  [illegible]
```

```
                                   BORDER-HILITE-METHOD          'multi-corner-
rectangle
                                   TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 3))
                    ('CWRU-COMMAND LABEL "APT Generation"
                                   LEFT-ACTION '(lac (select :type 'setup-f[illegible]
(the operations-sequence-list)))
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible] of APT code"
                                   NAME 'APT-generation
                                   BORDER-DRAW-METHOD            '[illegible]
[illegible]
                                   BORDER-HILITE-METHOD          'multi-corner-
rectangle
                                   TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 1))
                    ('CWRU-COMMAND LABEL "Completed plan"
                                   LEFT-ACTION '(view-plan (the part-model
[illegible]-planner plan))
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible] plan"
                                   NAME 'view-plan
                                   TOP (ROW 2) LEFT (CWRU-FEATURE[illegible]
                    ('CWRU-COMMAND LABEL "Header Info"
                                   LEFT-ACTION '(color-mod-[illegible]
[illegible]
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                   NAME 'header-info
                                   TOP (ROW 1) LEFT (CWRU-FEATURE[illegible] 1))
                    ('CWRU-COMMAND LABEL "Modify D2"
                                   LEFT-ACTION '(color-mod-feature (the part-[illegible]
[illegible]-planner))
                                   MOUSE-DOCUMENTATION-STRING "modify existing
[illegible]"
                                   NAME 'D2-features
                                   TOP (ROW 1) LEFT (CWRU-FEATURES-[illegible]
                    ('CWRU-COMMAND LABEL "Modify Setup"
                                   LEFT-ACTION '(modify-setup)
                                   MOUSE-DOCUMENTATION-STRING "[illegible]"
                                   NAME 'modify-setup
                                   TOP (ROW 2) LEFT (CWRU-FEA[illegible]
                    ('CWRU-COMMAND LABEL "Pick view"
                                   LEFT-ACTION 'SET-VIEW
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                   NAME (QUOTE SET[illegible])
                                   TOP (ROW 1) LEFT (CWRU-FEATURES-[illegible]
                    ('CWRU-COMMAND LABEL "Store plan"
                                LEFT-ACTION '(convert-dl-primary)
                                icon-style 'command
                                MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                NAME 'store
                                TOP (ROW 3) LEFT (CWRU-FEATURES-[illegible]
                    ('CWRU-COMMAND LABEL "Hide GD&T"
                                   LEFT-ACTION (QUOTE (TOGGLE-[illegible]
[illegible]
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                   NAME (QUOTE TOGGLE-GD&T)
                                   TOP (ROW 2) LEFT (CWRU-FEATURES-[illegible]
                    ('CWRU-COMMAND LABEL "Draw Part"
                                   LEFT-ACTION (QUOTE CLEAR-AND-[illegible]
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
```

```
                                   NAME (QUOTE [illegible])
                                   TOP (ROW 3) LEFT (CWRU-FEA[illegible]
               ('CWRU-COMMAND LABEL "Pan"
                                   LEFT-ACTION (QUOTE (PAN [illegible]
[illegible])))
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                   NAME (QUOTE PAN)
                                   TOP (ROW 1) LEFT (CWRU-FEA[illegible]
               ('CWRU-COMMAND LABEL "Rotate"
                                   LEFT-ACTION (QUOTE (ROTATE-[illegible]
[illegible]
                                   MOUSE-DOCUMENTATION-STRING "[illegible]
[illegible]"
                                   NAME (QUOTE ROTATE)
                                   TOP (ROW 2) LEFT (CWRU-[illegible]
               ))
```

Modifying the "Design-to-Manufacturing" Feature Translation (Modify D2)

```
;..............................................................................
.......
; [illegible]
;..............................................................................
.......

[illegible] (mach-feature pocket-feature) ([illegible]
  [illegible] '(pocket 'step-to-corner 'through-[illegible]
    [illegible] feature-choice "Select a Manufactur[illegible]"
    [illegible] feature)
    [illegible]
      [illegible] fabrication-planner d[illegible]-feature-p[illegible]
        [illegible] fabrication-planner d[illegible]-feature-p[illegible]
          [illegible] (the feature [illegible])) [illegible]

[illegible]
```

```
                (colon-make-d2-name self)
                :mixin '(d2-perp-aligned-through-slot-feature)
                :init-list (list
                           (cons 'd1-feature (the)))
                )
        )
        ((equal feature 'open-step)
         (if (the d2-feature) (kill-part (the d2-feature)))
         (interactive-unasn-variable (the) :attribute-name 'd2-feature)
         (add-part (the cam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-open-step-feature)
                :init-list (list
                           (cons 'd1-feature (the))
                            (fillet-radius . (the [illegible]
[illegible]
                )
        )
        (t (all feature (the d2-feature) feature))
        )



;..........................................................................
.......
; [illegible] methods for perp-aligned-through-slot-feature
;..........................................................................
.......

[illegible] (perp-aligned-through-slot-feature [illegible]
[illegible] 'pocket 'step-to-corner 'through-[illegible]
[illegible] feature-name "[illegible] Value [illegible]"
[illegible] d2-value feature)

[illegible] planner [illegible] d1-feature [illegible]
[illegible] planner d1-d2-feature-pair [illegible]
[illegible] (the) feature pair) [illegible]


[illegible] d2-feature perp-aligned-through-[illegible]
[illegible] d2-feature)
      [illegible] ((equal feature 'through-slot)
              (if (the d2-feature) (kill-part (the d2-feature)))
              [illegible]-variable (the) :attribute-name '[illegible]
              [illegible] (the))
        [illegible] 'step-[illegible]
              [illegible] (kill-part (the d2-feature))
              [illegible]-variable (the) :attribute-name '[illegible]
              (add-part (the cam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-step-to-corner-feature)
                :init-list (list
                           (cons 'd1-feature (the))
                           '(shoulder-corner-radius . [illegible]
[illegible] d1-feature)))))
                )

        [illegible] 'pocket)
          [illegible] (kill-part (the d2-feature))
                [illegible] (the) :attribute-name '[illegible]
          (add-part (the cam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-pocket-feature)
```

```
                :init-list (list
                            (cons 'd1-feature (the)))
                )
          )
          ((equal feature 'open-step)
           (if (the d2-feature) (kill-part (the d2-feature)))
           (interactive-smash-variable (the) :attribute-name 'd2-feature)
           (add-part (the eam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-open-step-feature)
                :init-list (list
                            (cons 'd1-feature (the))
                            '(fillet-radius . (the bottom-[illegible]
[illegible] d1-feature))))
                )
          )
          [illegible] (the d2-feature)) feature)
          )

;************************************************************************************
*******
; [illegible] step-to-a-shoulder-feature
;************************************************************************************
*******

[illegible] d-d2-feature step-to-a-shoulder-feature [illegible]
  [illegible] (list 'pocket 'step-to-corner 'thread-[illegible]
  [illegible] (menu-choose feature-choice "Select a Manufactur[illegible]"
    [illegible] :input-value feature)
  [illegible]
  [illegible] part-model fabrication-planner d1-d2-feature-pairs)
    [illegible] part-model fabrication-planner d1-d2-feature-pairs
        [illegible] (acons (the) feature pairs)) (assoc (the [illegible]


[illegible] d-d2-feature step-to-a-shoulder-feature [illegible]
  [illegible] feature)
        [illegible] feature 'step-to-corner)
           [illegible] (the d2-feature) (kill-part (the d2-feature))
           [illegible] variable (the) :attribute-name '[illegible]
           [illegible] d2-feature (the)))
          [illegible] feature 'pocket)
           [illegible] d2-feature) (kill-part (the d2-feature)))
           [illegible] variable (the) :attribute-name '[illegible]
           [illegible] (the eam-space part-model)
                [illegible] make-d2-name self)
                :mixin '(d2-step-to-corner-feature)
                :init-list (list
                            (cons 'd1-feature (the)))
                )
          )
          ((equal feature 'thread-slot)
           (if (the d2-feature) (kill-part (the d2-feature)))
           (interactive-smash-variable (the) :attribute-name '[illegible]
           (add-part (the eam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-perp-aligned-thread-[illegible]
                :init-list (list
                            (cons 'd1-feature (the)))
                )
          )
```

```
              ((equal feature 'open-step)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name [illegible])
               (add-part (the eam-space part-model)
                         (color-make-d2-name self)
                         :mixin '(d2-open-step-feature)
                         :init-list (list
                                     (cons 'd1-feature (the))
                                     '(fillet-radius . (the bottom-[illegible]
[illegible])))
                         )
               )
              ((equal feature (the d2-feature)) [illegible])
              )
         )
   )

;*****************************************************************************
*******
; [illegible] methods for open-step feature
;*****************************************************************************
*******

[illegible] (make-d2-feature open-step-feature) [illegible]
  [illegible] (list 'pocket 'step-to-corner 'through-[illegible]
  [illegible] feature-choice "Select a [illegible]"
  [illegible] feature)
  [illegible]
  [illegible] (the part-model [illegible]-planner d1-d2-feature-[illegible])
  [illegible] (the part-model [illegible]-planner d1-d2-feature-pair
        [illegible] (the) feature pair)) [illegible]
  )


[illegible] (make-new-d2-feature open-step-feature) [illegible]
  [illegible] d1-feature)
        [illegible] (equal feature 'open-step)
              [illegible] (kill-part [illegible])
              [illegible] [illegible]-variable (the) [illegible]
              [illegible])
             ((equal feature 'step-to-corner)
              (if (the d2-feature) (kill-part (the d2-feature)))
              [illegible]-smash-variable (the) :attribute-name [illegible]
              (add-part (the eam-space part-model)
                        (color-make-d2-name self)
                        :mixin '(d2-step-to-corner-feature)
                        :init-list (list
                                    (cons 'd1-feature (the))
                                     (shoulder-corner-radius . (the [illegible]
[illegible] (the d1-feature)))))
                        )
              )
             ((equal feature 'through-slot)
              (if (the d2-feature) (kill-part (the d2-feature)))
              [illegible]-smash-variable (the) :attribute-name [illegible]
              (add-part (the eam-space part-model)
                        (color-make-d2-name self)
                        :mixin '(d2-perp-align[illegible])
                        :init-list (list
                                    (cons 'd1-feature (the)))
                        )
              )
             ((equal feature 'pocket)
              (if (the d2-feature) (kill-part (the d2-feature)))
```

```
(interactive-smash-variable (the) :attribute-name 'd2-feature)
(add-part (the cam-space part-model)
          (colon-make-d2-name self)
          :mixin '(d2-open-step-feature)
          :init-list (list
                      (cons 'd1-feature (the))
                      (cons 'corner-fillet-radius '0.25))
          )
 )
((equal feature (the d2-feature)) feature)
)


;*************************************************************************
*******
; [illegible] method for perp-blind-hole feature
;*************************************************************************
*******

([illegible] (make-d2-feature perp-blind-hole-feature) [illegible]
  (setf feature-choice (list 'through-hole 'blind-hole 'pocket [illegible])
  (setf feature (menu-choose feature-choice "Select a Manufacturing [illegible]")
  ([illegible] instance :input-value feature)
  ([illegible])
  ([illegible] part-model fabrication-planner d1-d2-feature [illegible])
   ([illegible] fabrication-planner d1-d2-feature-pair
        [illegible] (the) feature [illegible])


([illegible] make-d2-feature perp-blind-hole-feature [illegible]
  ([illegible] d2-feature)
     ([illegible] feature 'blind-hole)
         ([illegible] (the d2-feature) (kill-part (the d2-feature)))
         ([illegible] (the) :attribute-name [illegible])
           ([illegible] d2-feature (the)))
        ([illegible] feature 'step-to-corner)
         ([illegible] (the d2-feature) (kill-part (the d2-feature)))
         ([illegible] (the) :attribute-name [illegible])
         ([illegible])
                    [illegible]
              :mixin '([illegible])
              :init-list (list
                          (cons 'd1-feature [illegible])
                           (width . (the radius [illegible]))
                           (depth . (* 2 (the radius [illegible])))
[illegible]
                          (cons 'fillet-radius '0.25))
              )
 )
        ([illegible] feature 'through-hole)
         ([illegible] (the d2-feature) (kill-part (the d2-feature)))
          ([illegible] (the) :attribute-name [illegible])
          ([illegible])
              [illegible] make-d2-name [illegible]
              :mixin '(d2-perp-through-feature)
              :init-list (list
                          (cons 'd1-feature (the))
                          (cons 'hole-type [illegible]))
              )
 )
        ([illegible] feature 'pocket)
         ([illegible] (the d2-feature) (kill-part (the d2-feature)))
```

```
        (interactive-smash-variable (the) :attribute-name 'd2-feature)
        (add-part (the eam-space part-model)
                (colon-make-d2-name self)
                :mixin '(d2-pocket-feature)
                :init-list (list
                        (cons 'd1-feature (the))
                         '(width . (* 2 (the radius [illegible]
feature))))
                         '(depth . (* 2 (the radius (from (the d1-
feature))))
                         '(corner-radius . (the radius (from (the d1-
feature))))
                        (cons 'fillet-radius '0.25))
                )
        )
        (equal feature (the d2-feature)) feature)
        )


;.........................................................................
.......
; [illegible] perp-through-hole feature
;.........................................................................
.......

[illegible] (make-d2-feature perp-through-hole-feature) [illegible]
 [illegible] (list 'through-hole 'blind-hole 'pocket [illegible]
 [illegible] feature-choice "Select a Manufacturing [illegible]"
 [illegible] input-value feature)
 [illegible]
 [illegible] part-model (part-model-d2-feature) [illegible]
 [illegible]
 [illegible] (radius (the) feature path) [illegible]


[illegible] (colon-new-d2-feature perp-through-hole-feature) [illegible]
 [illegible] d2-feature)
        [illegible] ((equal feature 'through-hole)
                (let ((the d2-feature) (kill-part (the d2-feature)))
                (interactive-smash-variable (the) :attribute-name 'd2-[illegible]
                ([illegible]-add-d2-feature (the)))
                (equal feature '[illegible])
                (let ((the d2-feature) (kill-part (the d2-feature)))
                (interactive-smash-variable (the) :attribute-name '[illegible]
                (add-part (the eam-space part-model)
                        ([illegible])
                        :mixin '([illegible])
                        :init-list (list
                                (cons 'd1-feature (the))
                                 (width . (the radius [illegible]
                                 (depth . (* 2 (the radius [illegible]
[illegible]
                                (cons 'fillet-radius '0.25))
                        )
                )
                (equal feature 'pocket)
                 (let ((the d2-feature) (kill-part (the d2-feature)))
                (interactive-smash-variable (the) :attribute-name 'd2-[illegible]
                (add-part (the eam-space part-model)
                        (colon-make-d2-name self)
                        :mixin '(d2-pocket-feature)
                        :init-list (list
```

```
                    (cons 'd1-feature [illegible])
                     (width . (the radius [illegible]
                     (depth . (* 2 (the radius [illegible]
[illegible])))

                     (center-radius . (the radius [illegible]
[illegible]

                    (cons 'fillet-radius [illegible]
          )
        )
        ([illegible] feature '[illegible]-hole)
         ([illegible] (the d2-feature) (kill-part (the d2-feature)))
         ([illegible]-variable (the) [illegible]-name '[illegible]-feature)
         ([illegible]-part (the cam-space part-model)
                 (colon-make-d2-name self)
                 :mixin '(d2-perp-hole-feature)
                 :init-list (list
                             (cons 'd1-feature (the))
                             (cons 'hole-type '[illegible])
                 )
        )
        ([illegible] feature (the d1-feature) feature)
        )


;..............................................................................
.......
; [illegible] perp-[illegible]-feature
;..............................................................................
.......

     [illegible] feature perp-[illegible]-feature [illegible]
     [illegible] "[illegible]" [illegible]
)



;..............................................................................
.......
; [illegible] right-triangular-extrusion feature
;..............................................................................
.......

 [illegible]-feature right-triangular-extrusion-[illegible]
     [illegible] ([illegible] 'step-to-corner [illegible]
     [illegible] feature-choice "Select a Variant [illegible]"
 [illegible]-value feature)
 [illegible]
     [illegible]
     [illegible]
     [illegible] feature [illegible]



 [illegible] feature right-triangular-[illegible]
 [illegible]
   [illegible]-feature)
        ([illegible] feature 'step-to-corner)
         ([illegible] (the d1-feature) (kill-part (the d2-feature)))
         ([illegible]-variable (the) :attribute-name '[illegible]-feature)
         ([illegible]-part (the cam-space part-model)
                 ([illegible]-make-d2-name self)
                 :mixin '([illegible]-feature)
                 :init-list (list
```

```
                                   (cons 'a1-feature (the))
                                    (shoulder-corner-radius . [illegible]
[illegible] (the a1-feature))))
                        )

                 (equal feature 'pocket)
                  (if (the a2-feature) (kill-part (the a2-feature)))
                  (interactive-smash-variable (the) :attribute-name [illegible])
                  (add-part (the eam-space part-model)
                        (colon-make-d2-name self)
                        :mixin '(d2-pocket-feature)
                        :init-list (list
                                   (cons 'a1-feature (the)))
                        )
                 )
                 (equal feature 'through-slot)
                  (if (the a2-feature) (kill-part (the a2-feature)))
                  (interactive-smash-variable (the) :attribute-name [illegible])
                  (add-part (the eam-space part-model)
                        (colon-make-d2-name self)
                        :mixin '(d2-perp-aligned-through-[illegible])
                        :init-list (list
                                   (cons 'a1-feature (the)))
                        )
                 )
                 (equal feature 'open-step)
                  (if (the a2-feature) (kill-part (the a2-feature)))
                  (interactive-smash-variable (the) :attribute-name [illegible])
                  (add-part (the eam-space part-model)
                        (colon-make-d2-name self)
                        :mixin '([illegible]-open-step-feature)
                        :init-list (list
                                   [illegible]
                                    [illegible]
[illegible]
                        )

                 (equal feature (the a2-feature) feature)
                 )


;.........................................................................................
.......
; [illegible] methods for triangular-extrusion feature
;.........................................................................................
.......


     [illegible]
     [illegible] 'pocket 'step-[illegible]
     [illegible] feature [illegible]
     [illegible] slip-fit feature)

     [illegible] fabrication-planner a1-a2-feature-[illegible]
     [illegible] part-model fabrication-planner a1-a2-feature-pairs
          [illegible] (acons (the) feature pairs) (assoc (the [illegible]


     [illegible] a1-feature triangular-extrusion [illegible]
     [illegible]
          [illegible] (equal feature 'step-to-corner)
                  (if (the a2-feature) (kill-part (the a2-feature)))
                  (interactive-smash-variable (the) :attribute-name [illegible])
```

```
(add-part (the eam-space part-model)
          (colon-make-d2-name self)
          :mixin '(d2-step-to-corner-feature)
          :init-list (list
                     (cons 'd1-feature (the))
                      (shoulder-corner-fa[illegible]
[illegible] (the d1-feature))))
          )
;
(equal feature 'pocket)
 [illegible] (the d1-feature) (kill-part (the d2-feature))
 (interactive-smash-variable (the) :attribute-name '[illegible]
 (add-part (the eam-space part-model)
          (colon-make-d2-name self)
          :mixin '(d2-pocket-feature)
          :init-list (list
                     (cons 'd1-feature (the))))
          )

[illegible]
[illegible]
[illegible]
 (add-part (the eam-space part-model)
          (colon-make-d2-name self)
          :mixin '(d2-perp-aligned-[illegible]
          :init-list (list
                     (cons 'd1-feature (the)))
          )

(equal feature '[illegible]per-step)
 [illegible] (the d1-feature) (kill-part (the d2-feature))
 [illegible]
 [illegible]
          [illegible]
          :mixin '[illegible]
          :[illegible]
                     ([illegible] 'd1-feature [illegible]
                      ([illegible]let-rad[illegible]
[illegible]

 [illegible] feature (the d1-feature) feature)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;
;  [illegible]
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;

[illegible] d1-feature perp-aligned-quadrilateral-[illegible]
 [illegible] (list 'pocket 'step-to-corner 'through-[illegible]
 [illegible]-choose feature-choice "Select a Manufactur[illegible]"
 [illegible] :input-value feature)
 [illegible]
 [illegible] fabrication-planner d1-d2-feature-[illegible]
 [illegible] fabrication-planner d1-d2-feature-[illegible]
      [illegible] (the) feature pair) (setq (the [illegible]
```

```
[illegible] (defmethod [illegible] d-new-d2-feature perp-aligned-quad [illegible]
feature)
  (case (the d2-feature)
        (cond ((equal feature 'step-to-corner)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name 'd2-feature)
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-step-to-corner-feature)
                         :init-list (list
                                     (cons 'd1-feature (the)))
                         )
               )
              ((equal feature 'pocket)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name 'd2-feature)
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-pocket-feature)
                         :init-list (list
                                     (cons 'd1-feature (the)))
                         )
               )
              ((equal feature 'through-slot)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name 'd2-feature)
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-perp-aligned-through-slot-feature)
                         :init-list (list
                                     (cons 'd1-feature (the)))
                         )
               )
              ((equal feature 'open-step)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name 'd2-feature)
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-open-step-feature)
                         :init-list (list
                                     (cons 'd1-feature (the))
                                     [illegible]
  [illegible]
                         )
               )
              ((equal feature (the d2-feature)) feature)
              ))

;..........................................................................
.......
; [illegible] closed-pocket feature
;..........................................................................
.......

(defmethod (: make-d2-feature closed-pocket) (instance)
  (let ((feature-choice (list 'pocket 'step-to-corner 'through-slot [illegible]
        (feature (menu-choose feature-choice "Select a Manufacturing Feature [illegible]"
    (change-value instance :input-value feature)
    (setf pair [illegible])
    (setf pair (the part-model fabrication-planner d1-d2-feature-pair))
    (change (the part-model fabrication-planner d1-d2-feature-pair)
            ([illegible] (car (acons (the) feature pair)) (assoc (the) [illegible]
```

```
[illegible] (colon-new-d2-feature biased-pocket) (feature)
 [illegible] d2-feature)
        [illegible] ((equal feature 'step-to-corner)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name '[illegible])
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-step-to-corner-feature)
                         :init-list (list
                                    (cons 'd1-feature (the))
                                     (shoulder-corner-radius . (the corner-[illegible]
[illegible] (the d1-feature)))))
                         )
               ;
              ((equal feature 'pocket)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name '[illegible])
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-pocket-feature)
                         :init-list (list
                                    (cons 'd1-feature (the)))
                         )
               ;
              ((equal feature 'through-slot)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name '[illegible])
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-perp-aligned-through-[illegible])
                         :init-list (list
                                    (cons 'd1-feature (the)))
                         )
               ;
              ((equal feature 'open-step)
               (if (the d2-feature) (kill-part (the d2-feature)))
               (interactive-smash-variable (the) :attribute-name '[illegible])
               (add-part (the eam-space part-model)
                         (colon-make-d2-name self)
                         :mixin '(d2-open-step-feature)
                         :init-list (list
                                    (cons 'd1-feature (the))
                                     [illegible]
[illegible]
                         )

              ([illegible] feature (the d2-feature)) feature)
              )
         )

;..............................................................................
.......
; [illegible] feature
;..............................................................................
.......

[illegible]
 [illegible] "Sorry, you cannot change [illegible]" [illegible]
'[illegible])
 )
```

```
(defwhopper-method (colon-new-d2-feature fillet-feature) (feature)
  )

;**************************************************************************
*******
; modifying methods for rib feature
;**************************************************************************
*******

(defwhopper-method (mod-d2-feature rib-feature) (instance)
  (pop-up-message (format nil "Sorry, you cannot change this feature") :buttons
'(Continue))
  )


(defwhopper-method (colon-new-d2-feature rib-feature) (feature)
  )


;**************************************************************************
*******
; modifying methods for boss feature
;**************************************************************************
*******

(defwhopper-method (mod-d2-feature boss-feature) (instance)
  (pop-up-message (format nil "Sorry, you cannot change this feature") :buttons
'(Continue))
  )


(defwhopper-method (colon-new-d2-feature boss-feature) (feature)


;**************************************************************************
*******
; modifying methods for biased-rib feature
;**************************************************************************
*******

(defwhopper-method (mod-d2-feature biased-rib) (instance)
  (pop-up-message (format nil "Sorry, you cannot change this feature") :buttons
'(Continue))


(defwhopper-method (colon-new-d2-feature biased-rib) (feature)


;**************************************************************************
*******
; modifying methods for skew rib feature
;**************************************************************************
*******

(defwhopper-method (mod-d2-feature skew-rib) (instance)
  (pop-up-message (format nil "Sorry, you cannot change this feature") :buttons
'(Continue))
  )
```

```
(define-part-method (colon-new-d2-feature skew-rib) (feature)
  )

(define-part-method (new-d2-features-form fab-planner-type) ()
  (setf x 0)
  (setq pair nil)
  (do for it in (select :type 'geometric-feature-mixin :test (not (equal (the [illegible]
[illegible] "starting block"))))
           (setq pair (acons it (the d2-feature (:from it)) pair)))
  (change (the d1-d2-feature-pair) pair)
  (do list (it pair)
          (if (null (car it)) (setf d2 nil)
            (setf d2 (get-print-name (cdr it))))
            )
          (setf new-instance (create 'owrap-property
                                     label (format nil "~a" (get-[illegible]
                                     contents (format nil "~a" d2)
                                     name (get-print-name (car it))
                                     top (row x)
                                     left-action 'which-icon
                                     width 400))
          (decorate-icon (find-form (the form)) new-instance)
          (incf x 1)
          )
  )

([illegible] (attach-new-d2-features fab-planner-type) (&aux [illegible]
  ([illegible]-form (the))
  (setf [illegible] extent (max-extent (find-form (the form))))
  ([illegible] (the form)
                 :[illegible] (the)
                 :width (+ 40 (second extent))
                 :height (+ 40 (first extent))
                 :[illegible]
                 :[illegible]
                 :[illegible]-function '[illegible]-feature
                 :exit-function 'colon-exit-feature)

([illegible]-feature fab-planner-type) ()
  [illegible] (primary-domain (the))))
  [illegible]-features (the))
  [illegible]

([illegible]-feature fab-planner-type)([illegible])
  ([illegible] feature (the))
  ([illegible] (the [illegible]-d2-feature-pair))
          ([illegible]-icon (find-form (the form)) (get-icon-[illegible]
[illegible] (car it))))
          )
  )
  )

(define-part-method (colon-exit-feature fab-planner-type)(&rest [illegible]
[illegible]
  ([illegible] link-icon-args))
  (change (the [illegible]) t)
```

```
(setf pair nil)
(dolist (it (the d1-d2-feature-pair))
        ([illegible]-new-d2-feature (car it) (cdr it))
        (setf pair (acons (car it) (the d2-feature (:from (car it))) pair))
        (unregister-icon (find-form (the form)) (get-icon-instance (the form) [illegible]
part-name (car it))))
        )
(change [illegible]-d2-feature-pair pair)
([illegible]-wire (the part-model))
)
```

Header Information

```
([illegible]-part-method (color-cancel-feature plan)(&rest link-icon-args)
  ([illegible] link-icon-args))
  ([illegible]-feature (the))
)

[illegible]
[illegible]
[illegible]
)

([illegible]-part-method (change-jocas-no plan) (&aux old-jocas new-[illegible]
  [illegible] (the jocas-no)) (setf old-jocas "") (setf old-jocas (the [illegible]
  [illegible] new-[illegible] (pop-up-typein-read "Input Jocas Number" :[illegible]
  [illegible] (get-icon-instance (the form) 'jocas-no) :input-[illegible]
)

[illegible] (&aux [illegible]
[illegible] ""[illegible]
[illegible]
[illegible] "Input [illegible]" [illegible]
[illegible] (the form) [illegible]
)

([illegible] (change-priority plan) (&aux old-priority new-[illegible]
  [illegible] (setf old-priority "") (setf old-[illegible]
  [illegible] (pop-up-typein-read "Input Priority" [illegible]
  [illegible] (the form) 'priority) [illegible]
)

([illegible] (change-planner plan) (&aux old-planner new-[illegible]
  [illegible] (setf old-planner "") (setf old-planner [illegible]
  [illegible] new-planner (pop-up-typein-read "Input Planner" :init-[illegible]
  [illegible] (get-icon-instance (the form) 'planner) :input-[illegible]
)

(define-part-method (change-engineer plan) (&aux old-engineer new-engineer)
  [illegible] (the engineer)) (setf old-engineer "") (setf old-engineer [illegible]
  (setf new-engineer (pop-up-typein-read "Input Engineer" :init-[illegible]
```

```
  (change-value (get-icon-instance (the form) 'engineer) :input-value new-engineer)
  )


(define-part-method (change-lead-technician plan) (&aux old-lead-tech new-lead-tech)
  (if (null (the lead-technician)) (setf old-lead-tech "") (setf old-lead-tech (the
lead-technician)))
  (setf new-lead-tech (pop-up-typein-read "Input Lead Technician" :init-string old-
lead-tech))
  (change-value (get-icon-instance (the form) 'lead-technician) :input-value new-lead-
tech)
  )


(define-part-method (change-scheduler plan) (&aux old-scheduler new-scheduler)
  (if (null (the scheduler)) (setf old-scheduler "") (setf old-scheduler (the
scheduler)))
  (setf new-scheduler (pop-up-typein-read "Input Scheduler" :init-string old-
scheduler))
  (change-value (get-icon-instance (the form) 'scheduler) :input-value new-scheduler)
  )


(define-part-method (change-issue-date plan) (&aux old-issue-date new-issue-date)
  (if (null (the issue-date)) (setf old-issue-date "") (setf old-issue-date (the
issue-date)))
  (setf new-issue-date (pop-up-typein-read "Input Issue Date" :init-string old-issue-
date))
  (change-value (get-icon-instance (the form) 'issue-date) :input-value new-issue-
date)
  )


(define-part-method (change-date plan) (&aux old-due-date new-date)
  (if (null (the due-date)) (setf old-due-date "") (setf old-due-date (the due-date)))
  (setf new-date (pop-up-typein-read "Input Job Due Date" :init-string old-due-date))
  (change-value (get-icon-instance (the form) 'due-date) :input-value new-date)
  )


(define-part-method (change-customer plan) (&aux old-customer new-customer)
  (if (null (the customer)) (setf old-customer "") (setf old-customer (the customer)))
  (setf new-customer (pop-up-typein-read "Input Customer Name" :init-string old-
customer))
  (change-value (get-icon-instance (the form) 'customer) :input-value new-customer)
  )
```

Modify Setup

```
(defun modify-setup ()
  "enter the setup and fixturing environment"
  (select-layout :name 'ui-layout-man-setups)
  (set-pane-label (get-pane-named 'graphics) "Setups Modification")
  (_____-clear-and-redraw-wire (the part-model))
  )
```

Completed Plan

```
(defun finished (&rest junk-icon-args)
  (declare (ignore junk-icon-args))
  (when (equal (menu-choose '(Yes No) "Do you want to save this plan to file?")
        'Yes)
        (plan-printout (the part-model fabrication-planner plan))
        (tool-printout (the part-model fabrication-planner plan))
        (evaluation)
        t)
)

(define-part-method (view-plan plan) ()
  (accept-values 'view-plan :left 0 :top 40
                  :defaults-alist (list (cons 'final-plan (completed-plan (the))))
                 :exit-function 'finished            )
  )

(ws::create 'FORM-OBJECT ws::name 'VIEW-PLAN
  ws::logical name ':HOME ws::file (concat curpath  "man-view-plan.form")
  ws::left 0 top 0
  ws::edit-settings (list (cons 'ws::grid-size 10) (cons 'ws::display-grid [illegible]
'ws::mode 'GRID))
  ws::left-action 'FORM-OPERATIONS
  ws::middle-action 'FORM-REFRESH
  ws::right-action 'FORM-OPERATIONS
  ws::font-purpose-alist (copy-tree 'NIL)
  ws::color-purpose-alist (copy-tree 'NIL)
  ws::mouse-documentation-string "Mouse-Left & Mouse-Right: Form Operations Menu;
Mouse-Middle: Refresh Form."
  ws::icons '(('EDITOR-ICON
              CONTENTS (make-adjustable-array-of-strings (list "Please Wait..."))
              EDITOR-WIDTH 1004
              EDITOR-HEIGHT 489
              LABEL "Completed Plan"
              LABEL-LEFT 12
              BORDER-POINTS (QUOTE (16 21 1020 21 1020 510 16 510 16 21))
              HILITE-BORDER-POINTS (QUOTE (17 22 1019 22 1019 509 17 509 17 22))
              NAME (QUOTE final-plan)
              TOP 10
              WIDTH 1020
              HEIGHT 510)
              )
  ws::scroll-bar-info (copy-tree 'NIL))



(define-part-method (completed-plan plan) (&aux setups prints)
  (setf setups (select :type 'setup-face))
  (setf prints nil)
  (dolist (it setups prints)
        (setf prints (append prints (alt-ops-header it))))
  )
```

Done

```
(defmethod (color-return-to-fbdo command-icon) (&rest junk-icon-args)
  "exit current layout and return to feature based design environment (fbde)"
  (declare (ignore junk-icon-args))
  (select-layout :name 'cwru-feature-layout)
  (set-pane-label (get-pane-named 'display-pane) "Feature Based [illegible]")
  (color-enable-context    (the part-model) 'fbde-design)
  (color-enable-context    (the part-model) 'fbde-gdst)
```

```
  (colon-disable-context  (the part-model) 'man-planning)
  (colon-clear-and-redraw-wire (the part-model))
)
```

Modifying the Setups

Setups Modification Layout

```
(add-configuration 'ui-layout-man-setups  '(
     (TREE-PANE TREE-PANE-FLAVOR 0.0 0.08375209380234507 0.2623376623376623 1.0 WHITE
STEEL-BLUE)
     (GRAPHICS DISPLAY-PANE-FLAVOR 0.2623376623376623 0.08375209380234507 1.0 1.0
YELLOW BLACK)
     (MESSAGE-PANE MESSAGE-PANE-FLAVOR 0.4181818181818181816 0.0 1.0 0.08375209380234507
BLACK DARK-TURQUOISE)
     (SETUPS-PANE FORM-FLAVOR 0.0 0.0 0.4181818181818181816 0.08375209380234507
DARK-TURQUOISE)))

(create-layout 'ui-layout-man-setups '(ui-layout-man-setups (
        (SETUPS-PANE MAN-SETUPS))))
```

Form for Setups Modification Layout

```
(ws::create 'FORM-OBJECT name 'MAN-SETUPS file "/users/rds/rds/ui-form-man-
setups.lisp"
          left 0 top 0
          edit-settings (copy-list '((GRID-SIZE . 10) (DISPLAY-GRID) (MODE .
GRID)))
          left-action 'FORM-OPERATIONS
          middle-action nil
          right-action  nil
          COLOR-PURPOSE-ALIST (QUOTE ((:LIGHT-SLOPE . LIGHT-STEEL-BLUE)
                                      (:DARK-SLOPE . CORNFLOWER-BLUE)
                                      (:BACKGROUND . DARK-TURQUOISE)
                                      (:PURPOSE :LIGHT-SLOPE :DARK-SLOPE :BORDER
:BACKGROUND :DEFAULT)))
          FONT-PURPOSE-ALIST (QUOTE ((:LABEL :normal :normal :NORMAL) (:VALUE
:normal :normal :NORMAL)))

          icons '(
               ('CWRU-COMMAND LABEL "Current Setups"
                              LEFT-ACTION '(view-setups)
                              MOUSE-DOCUMENTATION-STRING "view current setups"
                              NAME 'current-setups
                              TOP (ROW 1) LEFT (CWRU-FEATURES-COLUMN 1))
               ('CWRU-COMMAND LABEL "Done"
                              LEFT-ACTION (QUOTE colon-return-to-man-plan)
                              MIDDLE-ACTION NIL
                              MOUSE-DOCUMENTATION-STRING "click left to return to
manufacturing planning"
                              NAME (QUOTE DONE)
                              BORDER-DRAW-METHOD          'MOTIF-BORDER-
RECTANGLE
                              BORDER-HILITE-METHOD        'MOTIF-BORDER-
RECTANGLE
                              TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 1))
               ('CWRU-COMMAND LABEL "Quantity/Machine"
```

```
                                LEFT-ACTION '(change-quantitys)
                                MOUSE-DOCUMENTATION-STRING "Change the quant[illegible]
machine in a setups"
                                NAME 'quantity-machine
                                BORDER-DRAW-METHOD            'MOTIF-BORDER-
RECTANGLE
                                BORDER-HILITE-METHOD          'motif-border-
rectangle
                                TOP (ROW 2) LEFT (CWRU-FEATURES-COLUMN 1))
                ('CWRU-COMMAND LABEL "Add Feature"
                                LEFT-ACTION '(add-features)
                                MOUSE-DOCUMENTATION-STRING "Add a feature t[illegible]
setup"
                                NAME 'add-feature
                                TOP (ROW 1) LEFT (CWRU-FEATURES-COLUMN 2))
                ('CWRU-COMMAND LABEL "Delete Feature"
                                LEFT-ACTION '(delete-feature)
                                MOUSE-DOCUMENTATION-STRING "modify existing [illegible]
features"
                                NAME 'delete-feature
                                TOP (ROW 2) LEFT (CWRU-FEATURES-COLUMN 2))
                ('CWRU-COMMAND LABEL "Add a setup"
                                LEFT-ACTION '(add-new-setup (the part-model
fabrication-planner plan))
                                MOUSE-DOCUMENTATION-STRING "Add Setups"
                                NAME 'add-setup
                                TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 2))
                ('CWRU-COMMAND LABEL "Delete Setups"
                                LEFT-ACTION '(colon-delete-a-setup (the part[illegible]
fabrication-planner plan))
                                MOUSE-DOCUMENTATION-STRING "Delete a Setup"
                                NAME 'delete-setup
                                TOP (ROW 1) LEFT (CWRU-FEATURES-COLUMN [illegible]))
                ('CWRU-COMMAND LABEL "Fixturing"
                                LEFT-ACTION '(view-setup-fixturing (select-a-
setup))
                                MOUSE-DOCUMENTATION-STRING "view fixturing
information"
                                NAME 'fixturing
                                TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 1))
                ('CWRU-COMMAND label "Operations"
                                LEFT-ACTION '(VIEW-OPERATIONS)
                                MOUSE-DOCUMENTATION-STRING "Call MetCAPP[illegible]
optimize operations"
                                NAME 'VIEW-OPERATIONS
                                TOP (ROW 2) LEFT (CWRU-FEATURES-COLUMN 3))
               ('CWRU-COMMAND LABEL "View Tooling"
                              LEFT-ACTION '(view-tooling)
                              MOUSE-DOCUMENTATION-STRING "View the MetCAPP[illegible]
information"
                              NAME 'VIEW-TOOLING
                              TOP (ROW 3) LEFT (CWRU-FEATURES-COLUMN 3))
                ('CWRU-COMMAND LABEL "Draw Part"
                                LEFT-ACTION (QUOTE CLEAR-AND-REDRAW)
                                MOUSE-DOCUMENTATION-STRING "redraw the [illegible]
representation"
                                NAME (QUOTE REDRAW)
                                TOP (ROW 2) LEFT (CWRU-FEATURES-COLUMN 3))
                ))
```

Deleting a Setups

```
(define-part-method (colon-delete-a-setup plan) (&aux (which-setup (select-a-setup)))
  (if (eq (pop-up-message (format nil "Do you really want to delete ~a?" (get-pr [illegible]-
name which-setup))
                          :button-list (list 'No! 'Yes))
        'Yes)
      (kill-part which-setup))
)
```

Adding a Setup

```
(define-part-method (add-new-setup plan) ()
  (let* ((features (select :type 'manufacturing-feature-mixin :test (not (equal (the
d2-feature-type) "starting block"))))
         (number (pop-up-typein-read "How many features do you want in this setup."))
         (choosing-features '())
         (choosing-new-feature '())
         (features-chosen (dotimes (x (read-from-string number) choosing-features)
                                 (setf choosing-new-feature (menu-choose features
"Select a feature to add to the new setup"))
                                 (setf features (remove choosing-new-feature features))
                                 (setf choosing-features (append (list choosing-new-
feature) choosing-features))
                                 ))
         (answer (pop-up-message (format nil "Are these the features you want in the
setup?~%~a" (features-string features-chosen))
                           :button-list (list 'Yes 'No))))
    (if (equal answer 'Yes) (gen-setups (the) (the part-model d2-starting-block)
features-chosen))
    )
  )
```

Adding a Feature

```
(define-part-method (colon-add-a-feature setup-face) (&aux features features-new
feature-chosen final-features answer)
  (setf features (select :type 'manufacturing-feature-mixin :test (not (equal (the d2-
feature-type) "starting block"))))
  (setf features-new nil)
  (dolist (it features)
         (if (not (member it (the d2-features))) (setf features-new (append features-
new (list it))))
         )
  (setf feature-chosen
       (menu-choose features-new (format nil "Select a Feature [illegible]" [illegible]
print-name (the)) (features-string (the d2-features)))))
  (setf final-features (append (the d2-features) (list feature-chosen)))
  (setf answer
       (pop-up-message (format nil "Are these the features you want [illegible]" [illegible]
print-name (the)) (features-string final-features))
                      :button-list (list 'Yes 'No)))
  (when (and (equal answer 'Yes) (not (member feature-chosen (the d2-features))))
       (change (the d2-features) final-features)
       (gen-setups (the superior) (the part-model d2-starting-block) (the d2-
features))
       )
  )
```

Deleting A Feature

```
(define-part-method (colon-delete-a-feature setup-face) ()
  (let* ((features (the d2-features))
        (feature-chosen (menu-choose (the d2-features) (format nil "Select a feature
to delete from ~a" (get-print-name (the)))))
        (final-features  (remove feature-chosen features))
        (answer
         (pop-up-message (format nil "Are these the features you want in ~a?~%~a"
(get-print-name (the)) (features-string final-features))
                         :button-list (list 'Yes 'No))))
    (when (equal answer 'Yes)
         (change (the d2-features) final-features)
         (gen-setups (the superior) (the part-model d2-starting-block) (the d2-
features))
         )
    )
  )
```

View Fixturing

```
(define-part-method (view-setup-fixturing setup-face) ()
  (let* ((fixtures-avail (car (select :use (the))))
        (print-fixtures (mapcar #'get-print-name fixtures-avail))
        (fixture-wanted
         (nth (position (menu-choose print-fixtures  (format nil "Choose the fixture
to view for ~a" (get-print-name (the))))
                         print-fixtures) fixtures-avail)))
       (draw-part fixture-wanted)
       )
  )
```

Quantity/Machine

```
(define-part-method (colon-suggest-machines setup-face) (&aux machines machine-wanted)
  (setq machines (the superior machine))
  (setq machine-wanted (menu-choose machines))
  (change-value (get-icon-instance (the form) 'machine) :input-value machine-wanted))
)

(define-part-method (colon-capable-machine setup-face) (&aux machines machine-wanted)
  (setq machines
        (retrieve
         :table 'msellow
         :data 'name
         :where (list
                   'xmax #'(lambda (x) (> x !width))
                   'zmax #'(lambda (z) (> z !depth)))
         :unique nil))
  (setq machine-wanted (menu-choose machines))
  (change-value (get-icon-instance (the form) 'machine) :input-value machine-wanted))
)

(define-part-method (colon-change-quantity setup-face) (&aux machine-wanted)
  (change-value (get-icon-instance (the form) 'quantity))
  (setf machine-wanted
       (first (retrieve
```

```
                :table (if (< (read-icon-value (the form) 'quantity) 7) 'mse...
'mse[illegible]gn)
                :data 'name
                :where (list
                        'xmax #'(lambda (x) (> x !width))
                        'zmax #'(lambda (z) (> z !depth)))
                :unique? nil))
        )
  (change-value (get-icon-instance (the form) 'machine) :input-value machine-[illegible]
  )


(define-part-method (colon-cancel-feature setup-face)(&rest junk-icon-args)
  (declare (ignore junk-icon-args))
  (colon-restore-feature (the))
  )

(define-part-method (colon-exit-feature setup-face)(&rest junk-icon-args)
  (declare (ignore junk-icon-args))
  (dolist (it (the d2-features))
          (change (the machine (:from it)) !machine)
          )
  .
  )


(defun change-machine (&aux choice)
  (setq choice (menu-choose '(("suggested machine list" :value suggested-machine-[illegible]
                              ("capable machine list" :value capable-machine-[illegible]
                            "Machine List Choice"))
  (cond ((equal choice 'suggested-machine-list) (colon-suggest-machines (the [illegible]
model current-subpart)))
        ((equal choice 'capable-machine-list) (colon-capable-machine (the part-model
current-subpart)))
        )
  )
(defun change-quantitys (&aux this-face)
  (setf this-face (select-a-setup))
  (change (the part-model current-subpart) this-face)
  (colon-mod-feature this-face)
  )
```

View Operations

```
(ws::create 'FORM-OBJECT ws::name 'VIEW-OPS
  ws::logical name ':HOME ws::file (concat ourpath  "man-view-ops.form")
  ws::left 0 top 0
  ws::edit-settings (list (cons 'ws::grid-size 10) (cons 'ws::display-[illegible]
'ws::mode 'GRID))
  ws::left-action 'FORM-OPERATIONS
  ws::middle-action 'FORM-REFRESH
  ws::right-action 'FORM-OPERATIONS
  ws::font-purpose-alist (copy-tree 'NIL)
  ws::color-purpose-alist (copy-tree 'NIL)
  ws::mouse-documentation-string "Mouse-Left & Mouse-Right: Form [illegible]
Mouse-Middle: Refresh Form."
  ws::items '(('EDITOR-ICON
              CONTENTS (make-adjustable-array-of-strings (list " [illegible] ..."
              EDITOR-WIDTH 834
              EDITOR-HEIGHT 219
              LABEL "Manufacturing Operations"
```

```
            LABEL-LEFT 12
            BORDER-POINTS (QUOTE (16 21 850 21 850 240 16 240 16 21))
            HILITE-BORDER-POINTS (QUOTE (17 22 849 22 849 239 17 239 17 22))
            NAME (QUOTE MANUFACTURING)
            TOP 10
            WIDTH 850
            HEIGHT 240)
           ('MULTIPLE-CHOICE-ICON
            CURRENT-CHOICE (QUOTE Continue)
            LABEL "Operation Changes"
            MOUSE-DOCUMENTATION-STRING "Rearrange operations in a setup"
            NAME 'ops-change
            TOP 275
            LEFT 10
            WIDTH 175
            HEIGHT 30
            icons '(('BOX-CHECK-ICON
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Delete An Operation"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'delete-operation
                    TOP 300
                    LEFT 10
                    WIDTH 175)
                    ('BOX-CHECK-ICON
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Reorder Operations"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'reorder-operations
                    TOP 300
                    LEFT 185
                    WIDTH 175)
                    ('BOX-CHECK-ICON
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Change Tool"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'change-tool
                    TOP 300
                    LEFT 345
                    WIDTH 100)
                    ('BOX-CHECK-ICON
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Change Feed"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'change-feed
                    TOP 300
                    LEFT 480
                    WIDTH 100)
```

```
                    ('BOX-CHECK-ICON
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Change Speed"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'change-speed
                    TOP 300
                    LEFT 615
                    WIDTH 100)
                    ('BOX-CHECK-ICON
                    checked? t
                    CHECKED-DRAW-METHOD (QUOTE CHECKED?-CIRCLE-DRAW)
                    LEFT-ACTION (QUOTE TOGGLE-MULTIPLE-CHOICE)
                    MOUSE-DOCUMENTATION-STRING "Click to make this the selection of
the multiple choice"
                    LABEL "Continue"
                    LABEL-LEFT 24
                    LABEL-TOP 5
                    NAME 'continue
                    TOP 300
                    LEFT 750
                    WIDTH 100))
               )
              )
  ws::scrollbar-info (copy-tree 'NIL))



(defun operations-check (setup-face answer)
  (cond ((equalp answer 'reorder-operations) (reorder-operations setup-face))
        ((equalp answer 'delete-operation) (delete-operation setup-face))
        ((equalp answer 'change-tool) (change-tool setup-face))
        ((equalp answer 'change-speed) (change-speed setup-face))
        ((equalp answer 'change-feed) (change-feed setup-face))
        ((equalp answer 'continue))
        ((equalp answer nil)
         (pop-up-message "Sorry, you must choose one of the multiple choices."
:button-list '(Continue)) nil)
        )
  )



(define-part-method (operations-printout setup-face) (print-lists)
  (let ((op-no 1)
        (printout2 nil)
        (correct-operation nil)
        )
    (if (not (stringp (caar (the operations-sequence-list))))
        (dolist (it (the operations-sequence-list) printout2)
                (setf correct-operation (get-correct-operation (car it) it))
                (if (null (nth 5 correct-operation))
                    (setf correct-operation (substitute 0 (nth 5 correct-operation)
correct-operation)))
                (setf printout2
                      (append printout2 (list (format print-lists " ~[illegible]
[illegible] ~,[illegible]~e,3[illegible]~7,2f|~5,2f|~7,4f|
    [illegible]~2a  /~[illegible]~16a

 --------------------------------------------------------------------------
---------------------------------- ~%"
```

```
                                                            (the machine) (nth 1 it) (nth 2 [illegible]
                                                            (nth 2 correct-operation) (nth [illegible]
correct-operation)
                                                            (* (nth 5 correct-operation) ([illegible]
correct-operation))
                                                            (nth 3 correct-operation) ([illegible]
correct-operation)
                                                            (nth 7 correct-operation) [illegible]
(today-date) (get-print-name (nth 0 it))))))
             (incf op-no 1)
             )
       (dolist (it (cdr (the operations-sequence-list)) printout2)
             (setf printout2
                   (append printout2
                          (list
                           (format print-lists "|~15a    |~20a ~a [illegible]
   ~2d  /~11a|~16a             |                |       |       |

 ---------------------------------------------------------------------------------------------
-------------------------------|~%"
                                                            (the machine) (nth 1 it) ([illegible]
                                                            -1 (today-date) (get-print-name [illegible]
0 it))))))
             )
       )
    )
  )


(define-part-method (alt-ops-printout setup-face) (osl)
  (let ((op-no 1)
        (printout2 nil)
        (correct-operation nil)
        )
    (if (not (stringp (caar osl)))
        (dolist (it osl printout2)
              (setf correct-operation (get-correct-operation (car it) it))
              (if (null (nth 5 correct-operation))
                  (setf correct-operation (substitute 0 (nth 5 correct-operation)
correct-operation)))
              (setf printout2
                   (append printout2 (list (format nil "~2d    ~16a ~29a [illegible]
~5,3f  ~6,3f    ~7,2f  ~5,2f  ~7,4f~%"
                                                            op-no (get-print-name (nth 0 it))
(nth 1 it) (nth 2 it)
                                                            (nth 2 correct-operation) ([illegible]
correct-operation)
                                                            (* (nth 5 correct-operation) ([illegible]
correct-operation))
                                                            (nth 3 correct-operation) ([illegible]
correct-operation)
                                                            (nth 7 correct-operation))))
              (incf op-no 1)
              )
       (dolist (it (car osl) printout2)
             (setf printout2
                  (append printout2 (list (format nil "~2d    ~16a ~29a [illegible]"
                                                          -1 (get-print-name (nth 0 it)) ([illegible]
it) (nth 2 it))))))
             )
       )
    )
  )
```

```
(define-part-method (alt-ops-header setup-face) ()
  (list (format nil "Setup Name:  ~a~%Op No  Feature              Operation
Tool ID       Passes Depth Total Depth Speed     Feed      Time~%~a~%"
               (get-print-name (the)) (alt-ops-printout (the) (the operations-sequence-
list))))
  )


(defun view-operations (&aux this-face)
  (setf this-face (select-a-setup))
  (accept-values 'view-ops :left 0 :top 40
                 :defaults-alist (list (cons 'manufacturing (alt-ops-header this-
face)))
                :exit-function #'(lambda (alist) (operations-check this-face
(assoc 'ops-change alist))))
                )
  )
```

Reorder Operations

```
(define-part-method (reorder-operations setup-face) (&aux new-osl-order string-ops-
order new-osl new-ops-order)
  (setf new-osl-order nil)
  (setf string-ops-order nil)
  (setf new-osl nil)
  (dotimes (x (length (the operations-sequence-list)))
          (setf string-ops-order (append string-ops-order (list (format nil " ~a " (+ x 1
))))))
          )
  (setf new-ops-order (string-convert
                      (pop-up-typein-read "Input the new operations order" :initial-value
(format nil "~a" string-ops-order))))
  (dolist (it (the operations-sequence-list))
          (setf new-osl-order (append new-osl-order
                                      (list (append (list (nth (position it (the
operations-sequence-list)) new-ops-order)) it))))
          )
  (setf new-osl-order (sort new-osl-order #'< :key #'car))
  (dolist (it new-osl-order)
          (setf new-osl (append new-osl (list (cdr it))))
          )
  (setf answer (pop-up-message (format nil "Is this the order you want the
operations?~%~a~%" (alt-ops-printout (the) new-osl))
                               :button-list '(Yes No)
                               :characters-wide 125))
  (when (equal answer 'Yes) (change (the operations-sequence-list) new-osl))
    )
  )
```

Delete an Operation

```
(define-part-method (delete-operation setup-face) ()
  (setf (ops-num (menu-choose (ops-length (the operations-sequence-list)) "Which
operation do you wish to delete?"))
        (answer (nth (1- ops-num) (alt-ops-printout (the) (the operations-sequence-
list))))
        (answer-wanted
         (pop-up-message (format nil "Is this the operation you want to delete? ~a"
answer) :button-list '(Yes No)
```

```
                        :characters-wide 125))
        (osl nil)
        (feature nil)
        )
    (when (equal answer-wanted 'Yes)
        (if (semi-or-finish-op (cdr (nth (1- ops-numb) (the operations-sequence-
list))))
            (progn
             (setf feature (car (nth (1- ops-numb) (the operations-sequence-list))))
             (setf osl (get-correct-operation feature (nth (1- ops-numb) (the
operations-sequence-list))))
             (change (the nc-ops-list (:from feature))
                    (remove osl (the nc-ops-list (:from feature)) :count 1))
             )
          (progn
            (pop-up-message "Sorry, you can't delete this operation"
                          :button-list '(continue) :characters-wide 125)
            nil))
        )
    )
  )
```

Change Tool

```
(define-part-method (change-tool setup-face) ()
  (let* ((ops-numb (menu-choose (ops-length (the operations-sequence-list)) "Which
tool do you wish to change?"))
        (feature (car (nth (- ops-numb 1) (the operations-sequence-list))))
        (ops (get-correct-operation feature (nth (- ops-numb 1) (the operations-
sequence-list))))
        (which-tool (nth 1 ops))
        (answer
         (pop-up-message "If you continue, you may not get tool information if the
tool is not in the original tool list"
                        :button-list '(Continue End)))
        (new-tool nil)
        (new-ops nil)
        )
    (when(equal answer 'Continue)
        (setf new-tool (pop-up-typein-read "Type in the new tool" :initial-string [illegible]
[illegible] "~a" which-tool)))
        (setf new-ops (substitute new-tool which-tool ops))
        (change (the nc-ops-list (:from feature))
              (substitute new-ops ops (the nc-ops-list (:from feature))))
        )
    )
  )
```

Change Speed

```
(define-part-method (change-speed setup-face) ()
  (let* ((ops-numb (menu-choose (ops-length (the operations-sequence-list)) "Which
speed do you wish to change?"))
        (feature (car (nth (- ops-numb 1) (the operations-sequence-list))))
        (ops (get-correct-operation feature (nth (- ops-numb 1) (the operations-
sequence-list))))
        (which-speed (nth 3 ops))
        (new-speed (pop-up-typein-read "Type in the new speed" :initial-string [illegible]
[illegible] "~a" which-speed)))
        (new-ops (substitute (read-from-string new-speed) which-speed ops))
        )
    (change (the nc-ops-list (:from feature))
          (substitute new-ops ops (the nc-ops-list (:from feature))))
    )
```

```
)
```

Change Feed

```
(defmethod-part-method (change-feed setup-face) ()
  (let* ((ops-numb (menu-choose (ops-length (the operations-sequence-list)) "[illegible]
feed do you wish to change?"))
         (feature (car (nth (- ops-numb 1) (the operations-sequence-list))))
         (ops (get-correct-operation feature (nth (- ops-numb 1) (the operation-
sequence-list))))
         (which-feed (nth 4 ops))
         (new-feed (pop-up-typein-read "Type in the new feed" :init-string (format nil
"~a" which-feed)))
         (new-ops (substitute (read-from-string new-feed) which-feed ops))
         )
    (change (the nc-ops-list (:from feature))
          (substitute new-ops ops (the nc-ops-list (:from feature))))
    )
  )
```

Tooling

```
(defun view-tooling ()
  (let* ((more-than-one (menu-choose (list 'Yes 'No) "Do you want the tools for [illegible]
the setups?"))
         (setup-wanted nil)
         (tool-data '()))
    (when (equal more-than-one 'Yes) (setf setup-wanted (select :type [illegible]
          (setf setup-wanted (select-a-setup)))
    [illegible]
                 [illegible]
               (setf tool-data (append tool-data (the tool-list (:from [illegible]
               )
      (setf tool-data (append tool-data (the tool-list (:from setup-wanted))))
    (if (listp tool-data)
          (pop-up-message (format nil "Here is your tool data~%~a" [illegible]
:button-list '(Continue) :characters-wide 80)
          )
    )
  )

(defun list-of-tool (tool-data &aux final-list)
  (setf final-list nil)
  (if (listp tool-data)
      (cond ((equalp (get-tool-format (car tool-data)) 0)
             (setf final-list (append final-list (list (format nil "[illegible]:
~a~%~a~%" (car tool-data) (non-ins-mil-cut (cdr tool-data)))))))
            ((equalp (get-tool-format (car tool-data)) 1)
             (setf final-list (append final-list (list (format nil "Tool ID:
~a~%~a~%" (car tool-data) (non-ins-drill (cdr tool-data)))))))
            ((equalp (get-tool-format (car tool-data)) 2)
             (setf final-list (append final-list (list (format nil "Tool ID:
~a~%~a~%" (car tool-data) (taps (cdr tool-data)))))))
            ((equalp (get-tool-format (car tool-data)) 3)
             (setf final-list (append final-list (list (format nil "[illegible]:
~a~%~a~%" (car tool-data) (non-ins-reamers (cdr tool-data)))))))
            ((equalp (get-tool-format (car tool-data)) 4)
             (setf final-list (append final-list (list (format nil "[illegible]:
[illegible]" (car tool-data) (non-ins-boring (car tool-data)))))))
            ((equalp (get-tool-format (car tool-data)) [illegible]
             (setf final-list (append final-list
```

```
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (non-ins-forw-counter-bore (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 6)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (non-ins-rev-counter-bore (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 7)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (non-ins-forw-counter-sink (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 8)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (non-ins-back-counter-sink (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 9)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (inserts (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 10)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (spade-drill-blades (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 11)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (ins-mil-body (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 12)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (ins-drill-body (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 13)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (ins-reamer-body (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 14)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (ins-bores (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 15)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (turn-bore-ins-holder (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 16)
               (setf final-list (append final-list (list (format nil  "Tool ID:
~a~%~a~%" (car tool-data) (comb-ins-mill-ins (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 17)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (comb-ins-drill-ins (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 18)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (comb-spade-drill-blade (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 19)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (comb-ins-reamer-ins (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 20)
               (setf final-list (append final-list
                                   (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-
data) (comb-ins-coring-ins (cdr tool-data)))))))
              ((equalp (get-tool-format (car tool-data)) 21)
               (setf final-list
                     (append final-list
                             (list (format nil  "Tool ID:  ~a~%~a~%" (car tool-data)
(comb-turn-bore-ins-holder-ins (cdr tool-data)))))))
              (t (setf final-list (append final-list (list (format nil  "~a~%" tool-
data))))))
     (setf final-list (append final-list (list (format nil "~a~%" tool-data)))))
```

```
)


(defun non-ins-mil-cut (tool-data)
  (format nil "Drawing Number:                    ~a
Cutting Diameter:              ~a
Diameter Designation:          ~a
Effective Flute Length:        ~a
Overall Length:                ~a
Cutter Body Type:              ~a
Cutting Angle:                 ~a
Coolant Feeding:               ~a
Number of Flutes:              ~a
Cutter End Type:               ~a
Nose Style:                    ~a
Nose Radius Size:              ~a
Nose Flat Angle:               ~a
Hand of Cut:                   ~a
Helix Direction:               ~a
Helix Angle:                   ~a
Shank or Drive Type:           ~a
Shank/Pilot Diameter:          ~a
Tool Material:                 ~a
Tool Material Class:           ~a
Tool Material Grade:           ~a
Tool Material Construction:    ~a
Land Type - Relief:            ~a
Land Width:                    ~a
Radial Rake Angle:             ~a
Primary Clearance Angle:       ~a
Secondary Clearance Angle:     ~a
Neck Diameter:                 ~a
Neck Length:                   ~a
Application Code:              ~a"
         (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
         (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
         (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
         (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
         (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data))
  )


(defun non-ins-drill (tool-data)
  (format nil "Drawing Number:                    ~a
Cutter Body  Diameter:              ~a
Diameter Designation:               ~a
Effective Axial Cutting Length:     ~a
Minor/Pilot Diameter:               ~a
Minor/Pilot Dia Designation:        ~a
Effective Length:                   ~a
Overall Length:                     ~a
Cutter Body Type:                   ~a
Coolant Feeding:                    ~a
Number of Flutes:                   ~a
Margin Width:                       ~a
Cutter Point Type:                  ~a
Point Grind Included Angle:         ~a
Web Thickness:                      ~a
```

```
Hand of Cut:                    ~a
Helix Angle:                    ~a
Shank Type:                     ~a
Shank Diameter:                 ~a
Tool Material:                  ~a
Tool Material Class:            ~a
Tool Material Grade:            ~a
Tool Material Construction:     ~a"
         (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
         (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
         (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
         (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data))
  )



(defun taps (tool-data)
  (format nil "Drawing Number:                    ~a
Manufacturers Designation:          ~a
Product Name:                       ~a
Basic Type:                         ~a
Tap Size:                           ~a
Tap Size in decimal:                ~a
Thread Pitch:                       ~a
Thread Limit:                       ~a
Pitch Diameter:                     ~a
Pitch Diameter Relief:              ~a
Outside Diameter Relief Type:       ~a
Outside Diameter Relief at Heel:    ~a
Concentric Margin:                  ~a
Number of Flutes:                   ~a
Chamfer Lead:                       ~a
Thread Form:                        ~a
Hand of Cut:                        ~a
Effective Cutting Length:           ~a
Overall Length:                     ~a
Flute Type:                         ~a
Shank Type:                         ~a
Shank Diameter:                     ~a
Vendor Code Designation:            ~a
Tool Material Class:                ~a
Tool Material Grade:                ~a
Manuf. Coating Designation:         ~a
Coolant Feeding:                    ~a
Helicoil Tap Number:                ~a
Helix Angle:                        ~a
Rake Angle:                         ~a
Hook Angle:                         ~a
Land Width:                         ~a
Core Diameter:                       ~a
Back Taper:                         ~a"
         (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
         (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
         (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
         (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
```

```
        (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
        (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) (nth 33 tool-data))
  )


(defun non-ins-reamers (tool-data)
  (format nil "Drawing Number:                 ~a
Min. Cutting Diameter:          ~a
Max. Cutting Diameter:          ~a
Diameter Designation:           ~a
Effective Cutting Length:       ~a
Overall Length:                 ~a
Cutter Body Type:               ~a
Cutting Angle:                  ~a
Coolant Feeding:                ~a
Number of Flutes:               ~a
Cutter End Type:                ~a
Nose Chamfer Angle:             ~a
Nose Radius/Chamfer Length:     ~a
Shaved Lead Length:             ~a
Shaved Lead Angle:              ~a
Hand of Cut:                    ~a
Helix Type:                     ~a
Helix Angle:                    ~a
Shank Type:                     ~a
Shank Diameter:                 ~a
Tool Material:                  ~a
Tool Material Class:            ~a
Tool Material Grade:            ~a
Tool Material Construction:     ~a
Pilot Body Diameter:            ~a
Pilot Diameter Designation:     ~a
Pilot Locating Length:          ~a
Type of Center Grind:           ~a
Margin Width:                   ~a
Margin Relief Angle:            ~a
Radial Rake Angle:              ~a
Core Diameter:                   ~a"
        (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
        (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
        (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
        (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
        (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
        (nth 30 tool-data) (nth 31 tool-data))
  )


(defun non-ins-boring (tool-data)
  (format nil "Drawing Number:                 ~a
Manufacturer's Designation:    ~a
Body Type:                     ~a
End Type:                      ~a
Lead Angle When Mounted:       ~a
Nose Radius:                   ~a
Min. Bore Diameter:            ~a
Max. Depth of Cut:             ~a
```

```
Radial Adjustment Range:      ~a
Effective Cutting Length:     ~a
Overall Length:               ~a
Shank Type:                   ~a
Shank Diameter:               ~a
Tool Material:                ~a
Tool Material Class:          ~a
Tool Material Grade:          ~a
Tool Material Construction:   ~a
Primary Relief Angle:         ~a
Minimum Thread Pitch:         ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data))
  )


(defun non-ins-forw-counter-bore (tool-data)
  (format nil "Drawing Number:                ~a
Cutting Diameter:             ~a
Diameter Designation:         ~a
Effective Cutting Length:     ~a
Overall Length:               ~a
Cutter Body Type:             ~a
Coolant Feeding:              ~a
Number of Flutes:             ~a
Cutter End Type:              ~a
Nose Radius:                  ~a
Minimum Cutting Diameter:     ~a
Hand of Cut:                  ~a
Shank Type:                   ~a
Shank Diameter:               ~a
Tool Material:                ~a
Tool Material Class:          ~a
Tool Material Grade:          ~a
Tool Material Construction:   ~a
Pilot Body Diameter:          ~a
Diameter Designation:         ~a
Pilot Locating Length:        ~a
Land Type-Relief:             ~a
Land Width:                   ~a
Primary Clearance Angle:      ~a
Secondary Clearance Angle:    ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) [illegible]
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) [illegible]
10 tool-data) (nth 11 tool-data)
          (nth 25 tool-data) (nth 13 tool-data) (nth 14 tool-data) [illegible]
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) [illegible]
(nth 22 tool-data) (nth 23 tool-data)
          (nth 24 tool-data))
  )


(defun non-ins-rev-counter-bore (tool-data)
  (format nil "Drawing Number:                ~a
Cutting Diameter:             ~a
```

```
Diameter Designation:          ~a
Effective Cutting Length:      ~a
Overall Length:                ~a
Cutter Body Type:              ~a
Coolant Feeding:               ~a
Number of Flutes:              ~a
Nose Radius:                   ~a
Minimum Cutting Diameter:      ~a
Hand of Cut:                   ~a
Tool Material:                 ~a
Tool Material Class:           ~a
Tool Material Grade:           ~a
Tool Material Construction:    ~a
Pilot Body Diameter:           ~a
Diameter Designation:          ~a
Pilot Locating Length:         ~a
Shank Type:                    ~a
Shank Diameter:                ~a
Land Type-Rellef:              ~a
Land Width:                    ~a
Primary Clearance Angle:       ~a
Secondary Clearance Angle:     ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data))
  )


(defun non-ins-forw-counter-sink (tool-data)
  (format nil "Drawing Number:                    ~a
Outside Diameter:                   ~a
Diameter Designation:               ~a
Min. Cutting Diameter:              ~a
Max, Cutting Diameter:              ~a
Cutter Body Type:                   ~a
Lead or Included Cutting Angle:     ~a
Coolant Feeding:                    ~a
Number of Flutes:                   ~a
Effective Cutting Length:           ~a
Overall Length:                     ~a
Cutter End Type:                    ~a
Hand of Cut:                        ~a
Shank Type:                         ~a
Shank Diameter:                     ~a
Tool Material:                      ~a
Tool Material Class:                ~a
Tool Material Grade:                ~a
Tool Material Construction:         ~a
Pilot Body Diameter:                ~a
Diameter Designation:               ~a
Pilot Locating Length:              ~a
Land Type-Rellef:                   ~a
Land Width:                         ~a
Primary Clearance Angle:            ~a
Secondary Clearance Angle:          ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
```

```
         (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
         (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
         (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
         (nth 24 tool-data) (nth 25 tool-data))
  )


(defun non-ins-back-counter-sink (tool-data)
  (format nil "Drawing Number:               ~a
Outside Diameter:                  ~a
Diameter Designation:              ~a
Min. Cutting Diameter:             ~a
Max. Cutting Diameter:             ~a
Cutter Body Type:                  ~a
Lead or Included Cutting Angle:    ~a
Coolant Feeding:                   ~a
Number of Flutes:                  ~a
Effective Cutting Length:          ~a
Overall Length:                    ~a
Hand of Cut:                       ~a
Tool Material:                     ~a
Tool Material Class:               ~a
Tool Material Grade:               ~a
Tool Material Construction:        ~a
Pilot Body Diameter:               ~a
Diameter Designation:              ~a
Pilot Locating Length:             ~a
Shank Type:                        ~a
Shank Diameter:                    ~a
Land Type-Relief:                  ~a
Land Width:                        ~a
Primary Clearance Angle:           ~a
Secondary Clearance Angle:         ~a"
         (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
         (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
         (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
         (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
         (nth 24 tool-data))
  )


(defun inserts (tool-data)
      (format nil "Drawing Number:                 ~a
Manufacturer's Insert Designation:           ~a
Insert Manufacturer:       ~a
Insert Chape Designation:       ~a
Insert Thickness:               ~a
Inscribed Circle or Size of Insert:          ~a
Nose Style:              ~a
Nose Radius Size/Flat Length:                ~a
Rake Style:                     ~a
Relief Style:            ~a
Relief Angle:              ~a
Chip Control:                   ~a
Cutting Edge Configuration:              ~a
```

```
Surface Condition:              ~a
Number of Indexable Cutting Edges:              ~a
Tool Material:              ~a
Tool Material Class:              ~a
Tool Material Grade:              ~a
Code Group Identifier:    ~a
Cut-off/Groove Tool Width of Cut:              ~a
Cut-off/Groove Tool Depth of Cut:              ~a
Thread Type Category:              ~a
Thread Application Category:              ~a
Minimum External Thread Pitch:              ~a
Maximum External Thread Pitch:        ~a
Minimum Internal Thread Pitch:              ~a
Maximum Internal Thread Pitch:     ~a"
              (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data)
(nth 4 tool-data) (nth 5 tool-data)
              (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data)
(nth 10 tool-data) (nth 11 tool-data)
              (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-
data) (nth 16 tool-data) (nth 17 tool-data)
              (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-
data) (nth 22 tool-data) (nth 23 tool-data)
              (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data))
  )


(defun spade-drill-blades (tool-data)
  (format nil "Drawing Number:              ~a
Cutting Diameter:              ~a
Diameter Designation:              ~a
Cutting Edge Type:        ~a
Point Grind Angle:              ~a
Length Gage:              ~a
Blade Type:        ~a
Blade Thickness:              ~a
Depth of Cut:              ~a
Tool Material:              ~a
Tool Material Class:              ~a
Tool Material Grade:              ~a
Holder Series Designation:          ~a
Group Code Identifier:              ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data))
  )


(defun ins-mill-body (tool-data)
  (format nil "Drawing Number:              ~a
Manufacturer's Insert Designation:              ~a
Mill Body Manufacturer:        ~a
Rated Cutting Diameter-Decimal:              ~a
Rated Cutting Diameter Designation:              ~a
Cutter Body Type:              ~a
Lead Angle:        ~a
Coolant Feeding:              ~a
Cutter End Type:              ~a
Effective Cutting Length:              ~a
Overall Length:              ~a
Depth of Cut:              ~a
```

```
Helix Direction:            ~a
Helix Angle:                ~a
Radial Rake Geometry:            ~a
Radial Rake Angle:          ~a
Axial Rake:          ~a
Axial Rake Angle:           ~a
Shank or Drive Type:        ~a
Shank, Arbor Hole, or Taper Gage - Size:         ~a
Number of Insert Pockets:                 ~a
Number of Effective Cutting Edges:                      ~a
Number of Flutes:           ~a
Method of Insert Containment:               ~a
Position of Insert in Cutter Body:           ~a
Insert Group Code Identifier:          ~a
Inscribed Circle or Size of Required Insert/Blade:    ~a
Required Insert Thickness:            ~a
Minimum Acceptable Nose Radius:             ~a
Maximum Acceptable Nose Radius:             ~a
Recommended Wiper Blade Thickness:                     ~a
Neck Diameter:              ~a
Neck Length:                ~a
Application Code:           ~a"
        (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
        (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
        (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
        (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
        (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
        (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) (nth 33 tool-data)
  )


(defun ins-drill-body (tool-data)
  (format nil "Drawing Number:              ~a
Manufacturer's Insert Designation:              ~a
Mill Body Manufacturer:        ~a
Rated Cutting Diameter-Decimal:             ~a
Rated Cutting Diameter Designation:         ~a
Effective Cutting Length:      ~a
Overall Length:             ~a
Cutter Body Type:           ~a
Coolant Feeding:            ~a
Cutter Point or End Type:          ~a
Helix Direction:            ~a
Hand of Cut:                ~a
Rake Geometry Category:            ~a
Cutter Shank Type:        ~a
Shank Diameter or Taper Gage Size:          ~a
Number of Insert Pockets:          ~a
Number of Flutes:           ~a
Insert/Blade Group Code ID:                 ~a
Inscribed Circle or Size of Required Insert:                  ~a
Required Insert Thickness:           ~a
Minimum Acceptable Nose Radius:             ~a
Maximum Acceptable Nose Radius:             ~a"
        (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
        (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
```

```
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data))
  )


(defun ins-reamer-body (tool-data)
  (format nil "Drawing Number:                       ~a
Manufacturer's Insert Reamer Designation:            ~a
Reamer Body Manufacturer:           ~a
Minimum Rated Cutting Diameter:            ~a
Maximum Rated Cutting Diameter:            ~a
Rated Cutting Diameter Designation:        ~a
Effective Cutting Length:         ~a
Overall Length:                ~a
Reamer Body Type:              ~a
Coolant Feeding:               ~a
Reamer End Type:               ~a
Helix Direction:               ~a
Hand of Cut:                   ~a
Rake Geometry Category:                   ~a
Shank Type:          ~a
Shank Diameter or Taper Gage Size:         ~a
Number of Insert Pockets:              ~a
Number of Flutes:              ~a
Insert/Blade Group Code ID:               ~a
Inscribed Circle or Size of Required Insert:                ~a
Required Insert Thickness:                ~a
Minimum Acceptable Nose Radius:            ~a
Maximum Acceptable Nose Radius:            ~a
Recommended Wiper Blade Insert:            ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data))
  )


(defun ins-borer (tool-data)
  (format nil "Drawing Number:                       ~a
Manufacturer's Boring Tool Designation:            ~a
Boring Tool Manufacturer:         ~a
Boring Tool Body Type:            ~a
Boring Tool End Type:             ~a
Lead Angle When Mounted:                ~a
Minimum Bore Diameter:            ~a
Maximum Radial Depth of Cut:           ~a
Radial Adjustment Range:        ~a
Effective Cutting Length:         ~a
Overall Length:                ~a
Rake Geometry Category:                   ~a
Boring Tool Shank Type:          ~a
Boring Tool Shank Diameter or Size:        ~a
Number of Insert Pockets:              ~a
Insert Group Code ID:             ~a
Inscribed Circle or Size of Required Insert:                ~a
Required Insert Thickness:                ~a
Minimum Acceptable Nose Radius:            ~a
```

```
Maximum Acceptable Nose Radius:               ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data))
  )



(defun turn-bore-ins-holder (tool-data)
  (format nil "Drawing Number:                 ~a
Manufacturer's Tool Designation:                 ~a
Tool Body Manufacturer:          ~a
Operation Categories:                ~a
Coolant Required:                ~a
Hand of Tool:                    ~a
Rake Geometry Category:                    ~a
Lead Angle:                  ~a
Point Angle:                 ~a
Insert Shape Designation:                ~a
Overall Length:                  ~a
Cutter Shank Type:               ~a
Shank Height or Diameter:                      ~a
Shank Width:                 ~a
Distance from Centerline of Shank to tip of Insert:              ~a
Number of Insert Pockets:                ~a
Insert Seat Designation:                   ~a
Insert Pin Designation:            ~a
Insert Clamp Designation:            ~a
Insert Group Code ID:                  ~a
Inscribed Circle or Size of Required Insert:                    ~a
Required Insert Thickness:                ~a
Minimum Acceptable Nose Radius:               ~a
Maximum Acceptable Nose Radius:               ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data))
  )



(defun bore-ins-drill-ins (tool-data)
  (format nil "Drawing Number:                 ~a
Manufacturer's Insert Drill Designation:              ~a
Drill Body Manufacturer:         ~a
Rated Cutting Diameter Decimal:              ~a
Rated Cutting Diameter Designation:          ~a
Effective Cutting Length:        ~a
Overall Length:                  ~a
Cutter Body Type:                ~a
Coolant Feeding:                 ~a
Cutter Point or End Type:                ~a
Helix Direction:                 ~a
Hand of Cut:                     ~a
Rake Geometry Category:                  ~a
Cutter Shank Type:           ~a
```

```
Shank Diameter or Taper Gage Size:          ~a
Number of Insert Pockets:             ~a
Number of Flutes:             ~a
Insert ID Outside:                ~a
Manufacturer's Insert Designation:                ~a
Insert Manufacturer:          ~a
Tool Material:                ~a
Tool Material Class:          ~a
Tool Material Grade:          ~a
Insert ID Inside:                 ~a
Manufacturer's Insert Designation:                ~a
Insert Manufacturer:          ~a
Tool Material:                ~a
Tool Material Class:          ~a
Tool Material Grade:          ~a
Insert Shape Designation:       ~a
Insert Thickness:               ~a
Inscribed Circle or Size of Insert:                     ~a
Nose Style:                   ~a
Nose Radius Size:             ~a
Rake Style:            ~a
Relief Style:        ~a
Relief Angle:                 ~a
Chip Control:          ~a
Cutting Edge Condition:                ~a
Surface Condition:              ~a
Number of Indexable Cutting Edges:          ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
          (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
          (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) (nth 33 tool-data)
(nth 34 tool-data) (nth 35 tool-data)
          (nth 36 tool-data) (nth 37 tool-data) (nth 38 tool-data) (nth 39 tool-data)
(nth 40 tool-data))
  )


(defun comp-ins-mill-ins (tool-data)
  (format nil "Drawing Number:                ~a
Manufacturer's Insert Mill Designation:              ~a
Mill Body Manufacturer:           ~a
Rated Cutting Diameter Decimal:               ~a
Rated Cutting Diameter Designation:           ~a
Cutter Body Type:             ~a
Lead Angle:            ~a
Coolant Feeding:              ~a
Cutter End Type:              ~a
Effective Cutting Length:       ~a
Overall Length:               ~a
Hand of Cut:                  ~a
Helix Direction:              ~a
Helix Angle:                  ~a
Radial Rake Geometry:               ~a
Radial Rake Angle:            ~a
Axial Rake:          ~a
Axial Rake Angle:           ~a
```

```
Shank or Drive Type:            ~a
Shank, Arbor Hole, or Taper Gage Size:          ~a
Number of Insert Pockets:            ~a
Number of Effective Cutting Edges:          ~a
Number of Flutes:            ~a
Method of Insert Containment:            ~a
Position of Insert in Cutter Body:            ~a
Insert ID:            ~a
Manufacturer's Insert Designation:            ~a
Insert Manufacturer:            ~a
Insert Shape Designation:         ~a
Insert Thickness:            ~a
Inscribed Circle or Size of Insert:               ~a
Nose Style:            ~a
Nose Radius Size:            ~a
Rake Style:         ~a
Relief Style:         ~a
Relief Angle:            ~a
Chip Control:         ~a
Cutting Edge Condition:            ~a
Surface Condition:            ~a
Number of Indexable Cutting Edges:         ~a
Tool Material:            ~a
Tool Material Class:            ~a
Tool Material Grade:            ~a
Recommended Wiper Blade Insert:            ~a
Neck Diameter:            ~a
Neck Length:            ~a
Application Code:            ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
          (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
          (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) (nth 33 tool-data)
(nth 34 tool-data) (nth 35 tool-data)
          (nth 36 tool-data) (nth 37 tool-data) (nth 38 tool-data) (nth 39 tool-data)
(nth 40 tool-data) (nth 41 tool-data)
          (nth 42 tool-data) (nth 43 tool-data) (nth 44 tool-data) (nth 45 tool-data)
(nth 46 tool-data))
  )


(defun comp-spade-drill-blade (tool-data)
  (format nil "Drawing Number:            ~a
Manufacturer's Insert Drill Identifier:            ~a
Drill Body Manufacturer:            ~a
Cutting Diameter:            ~a
Diameter Designation:            ~a
Effective Cutting Length:         ~a
Overall Length:            ~a
Cutter Body Type:            ~a
Coolant Feeding:            ~a
Cutter Point or End Type:            ~a
Helix Direction:            ~a
Hand of Cut:            ~a
Radial Rake Category:            ~a
Cutter Shank Type:         ~a
```

```
Shank Diameter or Taper Gage Size:          ~a
Spade Blade ID:                   ~a
Cutting Edge Type:            ~a
Point Grind Angle:                   ~a
Regrindable:                  ~a
Blade Type:          ~a
Blade Thickness:             ~a
Tool Material:               ~a
Tool Material Class:         ~a
Tool Material Grade:         ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data))
  )


(defun comb-ins-reamer-ins (tool-data)
  (format nil "Drawing Number:                ~a
Manufacturer's Insert Reamor ID:              ~a
Reamer Body Manufacturer:          ~a
Minimum Rated Cutting Diameter:            ~a
Maximum Rated Cutting Diameter:            ~a
Rated Cutting Diameter Designation:        ~a
Effective Cutting Length:        ~a
Overall Length:              ~a
Reamer Body Type:            ~a
Coolant Feeding:             ~a
Reamer End Type:             ~a
Helix Direction:             ~a
Hand of Cut:                 ~a
Rake Geometry Category:                ~a
Shank Type:          ~a
Shank Diameter or Taper Gage Size: ~a
Number of Insert Pockets:              ~a
Number of Flutes:            ~a
Insert ID:                             ~a
Manufacturer's Insert Designation:     ~a
Insert Manufacturer:         ~a
Insert Shape Designation:        ~a
Insert Thickness:              ~a
Inscribed Circle or Size of Insert:                   ~a
Nose Style:                  ~a
Nose Radius Size:            ~a
Rake Style:          ~a
Tool Material:               ~a
Tool Material Class:         ~a
Tool Material Grade:         ~a
Relief Style:        ~a
Relief Angle:              ~a
Chip Control:          ~a
Cutting Edge Condition:             ~a
Surface Condition:            ~a
Number of Indexable Cutting Edges:          ~a
Recommended Wiper Blade Insert:              ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
```

```
        (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
        (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
        (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
        (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) (nth 33 tool-data)
(nth 34 tool-data) (nth 35 tool-data)
        (nth 36 tool-data))

 )



(defun comp-ins-boring-ins (tool-data)
  (format nil "Drawing Number:              ~a
Manufacturer's Boring Tool Designation:          ~a
Boring Tool Manufacturer:          ~a
Boring Tool Body Type:             ~a
Boring Tool End Type:              ~a
Lead Angle:                ~a
Minimum Bore Diameter:             ~a
Maximum Radial Depth of Cut:              ~a
Radial Adjustment Range:           ~a
Effective Cutting Length:        ~a
Overall Length:              ~a
Rake Geometry Category:                ~a
Boring Tool Shank Type:          ~a
Boring Tool Shank Diameter or Size: ~a
Number of Insert Pockets:                ~a
Insert ID:                               ~a
Manufacturer's Insert Designation:       ~a
Insert Manufacturer:         ~a
Insert Shape Designation:        ~a
Insert Thickness:              ~a
Inscribed Circle or Size of Insert:                ~a
Nose Style:                  ~a
Nose Radius Size:            ~a
Rake Style:               ~a
Relief Style:         ~a
Relief Angle:                ~a
Chip Control:             ~a
Cutting Edge Condition:              ~a
Surface Condition:             ~a
Number of Indexable Cutting Edges:          ~a
Tool Material:               ~a
Tool Material Class:         ~a
Tool Material Grade:         ~a"
        (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) (nth 3 tool-data) (nth
4 tool-data) (nth 5 tool-data)
        (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) (nth 9 tool-data) (nth
10 tool-data) (nth 11 tool-data)
        (nth 12 tool-data) (nth 13 tool-data) (nth 14 tool-data) (nth 15 tool-data)
(nth 16 tool-data) (nth 17 tool-data)
        (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) (nth 21 tool-data)
(nth 22 tool-data) (nth 23 tool-data)
        (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) (nth 27 tool-data)
(nth 28 tool-data) (nth 29 tool-data)
        (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data))
 )



(defun comp-turn-bore-ins-holder-ins (tool-data)
```

```
  (format nil "Drawing Number:                ~a
Manufacturer's Tool Identifier:            ~a
Tool Body Manufacturer:         ~a
Operation Categories:             ~a
Coolant Feeding:              ~a
Hand of Tool:                  ~a
Rake Geometry Category:                  ~a
Lead Angle:                ~a
Point Angle:                  ~a
Overall Length:               ~a
Cutter Shank Type:          ~a
Shank Height or Diameter:          ~a
Shank Width:             ~a
Distance from Centerline of Shank to Tip of Insert:        ~a
Number of Insert Pockets:               ~a
Insert Seat Designation:                ~a
Insert Pin Designation:              ~a
Insert Clamp Designation:              ~a
Insert ID:                    ~a
Manufacturer's Insert Designation:                ~a
Insert Manufacturer:          ~a
Insert Shape Designation:        ~a
Insert Thickness:                ~a
Inscribed Circle or Size of Insert:                    ~a
Nose Style:                    ~a
Nose Radius Size:              ~a
Insert Rake Style:                ~a
Insert Relief Style:           ~a
Relief Angle:                 ~a
Chip Control:               ~a
Cutting Edge Condition:              ~a
Surface Condition:                ~a
Number of Indexable Cutting Edges:          ~a
Tool Material:                    ~a
Tool Material Class:              ~a
Tool Material Grade:              ~a
Group Code Identifier:               ~a
Cut-Off/Groove Tool Width of Cut:                 ~a
Cut-Off/Groove Tool Depth of Cut:                  ~a
Thread Type Category:               ~a
Thread Application Category:              ~a
Minimum External Thread Pitch:           ~a
Maximum External Thread Pitch:           ~a
Minimum Internal Thread Pitch:           ~a
Maximum Internal Thread Pitch:           ~a"
          (nth 0 tool-data) (nth 1 tool-data) (nth 2 tool-data) [illegible]
 [illegible] tool-data) (nth 5 tool-data)
          (nth 6 tool-data) (nth 7 tool-data) (nth 8 tool-data) [illegible]
 [illegible] tool-data) (nth 11 tool-data)
          (nth 12 tool-data) (nth 13 tool-data) (nth 14 [illegible]
(nth 16 tool-data) (nth 17 tool-data)
          (nth 18 tool-data) (nth 19 tool-data) (nth 20 tool-data) [illegible]
(nth 22 tool-data) (nth 23 tool-data)
          (nth 24 tool-data) (nth 25 tool-data) (nth 26 tool-data) [illegible]
(nth 28 tool-data) (nth 29 tool-data)
          (nth 30 tool-data) (nth 31 tool-data) (nth 32 tool-data) [illegible]
(nth 34 tool-data) (nth 35 tool-data)
          (nth 36 tool-data) (nth 37 tool-data) (nth 38 tool-data) [illegible]
(nth 40 tool-data) (nth 41 tool-data)
          (nth 42 tool-data) (nth 43 tool-data) (nth 44 tool-data))
  )
```

Done (Setups)

```
(defmethod (colon-return-to-man-plan command-icon) (&rest junk-icon-args)
  "exit current layout and return to manufacturing planning environment"
  (declare (ignore junk-icon-args))
  (let ((man-features  (select :type 'manufacturing-feature-mixin :test (not (e ...
(the d1-feature-type) "starting block"))))
        (setup-features nil)
        (duplicate-features nil)
        (difference nil))
     (dolist (it (select :type 'setup-face))
            (setf setup-features (append setup-features (the d2-features ...
            )
     (dolist (it man-features)
            (if (> (count it setup-features) 1)
               (setf duplicate-features (append duplicate-features (list it))))
            )
     (if (not (null duplicate-features))
        (pop-up-message
         (format nil "~a is duplicated in more than one setup" duplicate-features)
:button-list '(Continue))
        )
     (setf difference (set-difference man-features setup-features))
     (if (not (null difference))
       (pop-up-message (format nil "~a is not in a setup" difference)
                   :button-list '(Continue))
       )
     (when (and (null difference) (null duplicate-features))
          (select-layout :name 'ui-layout-man-plan)
          (set-pane-label (get-pane-named 'graphics) "Manufacturing Planning ...
Level")
          (colon-clear-and-redraw-wire (the part-model))
          )
     )
   )
```

Printing the Plan, Tool Data, and Evaluation Form

Printing the Plan

```
(define-part-method (header-printout plan) (print-lists &aux setups)
  (setf setups (select :type 'setup-face))
  (format print-lists " ------------------------------------------------------------
----------------------------------------------------
  JCN  ~25a              RDS MANUFACTURING ORDER
of
---------------------------------------------------------------------------------
-------------------------------|
  RDS Planner:  ~20a  Programmer:  ~20a   Lead Tech:  ~20a
---------------------------------------------------------------------------------
-------------------------------|
| Project Name/Drawing Number:  ~20a Material:  ~20a Sequence No:
 ~%~a~%~a~%~a~%~a~%"
                (the job-control-number) (the planner) 'who-knows (the ...
technician) 'part-no
                (the part-model d2-starting-block material)
                "|------------------------------------------------------------
--------------------------------------------- "
```

```
                "|Machine/              Description of Work           [illegible]
 No. of Depth  Total | Speed |Feed |  Est       Act    "
                ".Oper No/Date
 Passes.of Cut Depth |  RPM     IPM | Time  | Time  |"
                " -----------------------------------------------------------------
---------------------------------------------|"
                )
  (dolist (it setups)
         (operations-printout it print-lists)
         )
  ;
  )


(define-part-method (plan-printout plan) ()
  (with-open-file (print-lists "/users/rds/rds/plan.done"
                              :direction  :output
                              :if-exists  :overwrite
                              :if-does-not-exist  :create)
                (header-printout (the) print-lists)
                )
  )


(define-part-method (tool-printout plan) (&aux tool-data)
  (setf tool-data nil)
  (dolist (it (select :type 'setup-face))
         (setf tool-data (append tool-data (the tool-list (:from [illegible]
         )
  (with-open-file (print-tools "/users/rds/rds/tools.done"
                              :direction  :output
                              :if-exists  :overwrite
                              :if-does-not-exist  :create)
                (dolist (it tool-data)
                        (format print-tools "Here is your tool data~%[illegible]" [illegible]
tool it))
                        )
                )
  )
```

Printing the Evaluation Form

```
(defun evaluation ()
 (with-open-file (print-eval "/users/rds/rds/eval.done"
                              :direction  :output
                              :if-exists  :overwrite
                              :if-does-not-exist  :create)
                (format print-eval "|-------------------------------------
---------------------------------------------------------------
  Tryout Evaluation            JCN  ~25a     | Sequence No:
Page    of
 ----------------------------------------------------------------------------
-----------------------------------|
  Project Name/Drawing Number:  ~20a                      Material: [illegible]
 ~%~a~%~a~%~a~%~a~%"
                        (the part-model fabrication-planner [illegible]
[illegible]
                        (the part-model o2-starting-block material)
                        " ------------------------------------------
------------------------------------------------------------ "
```

Comments	Oper No	Cutters New	Cutters Used	Speed %	Load Meter	

```
------------------------------------------------------------------------------------------------------
--------------------------------;

------------------------------------------------------------------------------------------------------
--------------------------------.

------------------------------------------------------------------------------------------------------
--------------------------------,

------------------------------------------------------------------------------------------------------
--------------------------------!

------------------------------------------------------------------------------------------------
--------------------------------|

------------------------------------------------------------------------------------------------
-------------------------------- "

                        )
                )
  )
```

Miscellaneous Functions

```
([illegible] ops-length (osl &aux ops-numbs)
  ([illegible])
  ([illegible] (length osl)) ops-numbs)
          ([illegible] ops-numbs (append ops-numbs (list ([illegible]))))
          )
  )


([illegible]-part-method (get-correct-operation manufacturing-feature-mixin) ([illegible]
  (let ((correct-operation nil))
    ([illegible] ([illegible] (the no-ops-list) correct-operation)
          ([illegible] (member (second osl) [illegible])
             (setf correct-operation (append correct-operation [illegible]
          )
     )
  )


([illegible] today-date (&aux todays-date month)
  ([illegible] todays-date (multiple-value-list (get-decoded-time))
       month ([illegible] '("Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" [illegible]
" [illegible]") (1- ([illegible] todays-date 4))))
  (format nil "~a ~a ~a" (fourth todays-date) month (sixth todays-[illegible]
)


([illegible] ([illegible] property-icon) (&rest [illegible])
  ([illegible] (the part-model (the name ([illegible]
  ([illegible] part-instance self)
  )
```

```
(defun select-a-setup (&aux choices readable-choices this-face)
  (setf choices (select :type 'setup-face))
  (setf readable-choices (mapcar #'get-print-name choices))
  (setf this-face (nth (position (menu-choose readable-choices "Select a Setup")
readable-choices) choices))
  this-face
  )

(defun view-setups ()
  (pop-up-message (format nil "Here is a list of setups and the features in each
setup~%~a" (setups-string))
                  :button-list '(Continue))
  )

(defun setups-string (&aux setup-list)
  (setf setup-list nil)
  (dolist (it (select :type 'setup-face) setup-list)
          (setf setup-list (append setup-list
                             (list (format nil "~a~%" (get-print-name it)) [illegible]
string (the a2-features (:from it)))))
          )
  )

(defun add-features (&aux this-face)
  (setf this-face (select-a-setup))
  (do.on-add-a-feature this-face)
  )

(defun delete-feature (&aux this-face)
  (setf this-face (select-a-setup))
  (do.on-delete-a-feature this-face)
  )


(defun features-string (features &aux features-list)
  (setf features-list nil)
  (dolist (it features features-list)
          (setf features-list (append features-list (list (format [illegible]
          )
  )
```

BIBLIOGRAPHY

1. Burgess, J. D. "A Review of Computer Aided Process Planning System," Australia, 1984, Presented at the SME AUTOMACH Australia '84 Conference, Rpt., in CAPP: From Design to Production. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1988.

2. Chang T.C., Expert Process Planning for Manufacturing, Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.

3. Concept Modeller Reference Manual, Release 1.3, Wisdom Systems, Cleveland, Ohio, July 1991.

4. Ham, Inyong, Chingping Han, and Jiankang Li, "Development of an In-House Computer Automated Process Planning System Based on Group Technology Concept." May 1987, Presented at the 15th North American Research Conference. Rpt. in CAPP: From Design to Production. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1988.

5. LeClair S. R., "The Rapid Design System: Memory-Driven Feature-Based Design", *Proceedings of the 1991 IEEE Conference on Systems Engineering,* August 3, 1991 Dayton Ohio.

6. Mason, Anthony K. and Andrew Young, "Computer Aided Process Planning for Printed Circuit Board Assembly." November 1987, Presented at the CASA/SME AUTOFACT '87 Conference. Rpt. in CAPP: From Design to Production. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1988.

7. MetCAPP User's Guide, Release 2, IAMS, Cincinnati, Ohio, September, 1990.

8. Mouleeswaran, C. B. and R. H. Phillips, "A Knowledge-Based Approach to Generative Process Planning." November 1985, Presented at the CASA/SME AUTOFACT '85 Conference. Rpt. in CAPP: From Design to Production. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1988.

9. Radack G. M., Jacobsohn J. F., Merat F. L., "Positioning Form Features within the Rapid Design System", *Proceedings of the 1991 IEEE Conference on Systems Engineering,* August 3, 1991 Dayton Ohio.

10. Sims, Jeffery, "CAPP/SFC: Integration Through the Process Plan." February 1985, Presented at the CASA/SME FMS for Electronics Conference. Rpt. in CAPP: From Design to Production. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1988.

11. Smith, Robert M. "Computer-aided, Fully Generative Process Planning." Manufacturing Engineering, May 1981. Rpt. in CAPP: Computer Aided Process Planning. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1985.

12. Steudel, Harold J. "Computer-aided Process Planning: Past, Present and Future." International Journal of Production Research, 22 (1984). Rpt. in CAPP: Computer Aided Process Planning. Ed. Joseph Tulkoff. Dearborn, Michigan: Society of Manufacturing Engineers, 1985.

13. Westhoven, Timothy E. Feature Sequencing In Process Planning Using an Episodal Associative Memory, Masters Thesis, Wright State University, November 1991.