EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

UNCLASSIFIED P A MOORE DEC 84 AFIT/GCS/ENG/84D-18 F/G 9/2 NL
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS

THESIS

Paul A. Moore, B.S.
Captain, USAF

AFIT/GCS/ENG/84D-18

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

85 03 13 148
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS

THESIS

Paul A. Moore, B.S.
Captain, USAF

AFIT/GCS/ENG/84D-18

Approved for public release; distribution unlimited
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS

THESIS

Presented to the Faculty of the School of Engineering of the Air Force Institute of Technology Air University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Systems

Paul A. Moore, B.S.
Captain, USAF

December 1984

Approved for public release; distribution unlimited
Preface

This investigation presents a foundation for a microcomputer software development environment. The resulting implementation of this thesis investigation is a prototype software development environment hosted on a microcomputer; the environment includes a flexible user interface, an installation program, a communications program, and an initial library of reusable software. The last chapter of this report outlines follow-on development efforts necessary to create a more complete microcomputer software development environment. This software development environment is formally identified as the "Microcomputer Software Development Workbench".

I wish to express my sincere gratitude to my thesis advisor, Dr. Gary B. Lamont, for his professional guidance, suggestions, and critiques throughout the thesis effort. They were invaluable to the success of this project. I would also like to thank the members of my thesis committee Dr. Tom Hartrum, Captain Pat Lawlis, and Captain Rob Milne, for their support and suggestions.

Finally, I offer my love and thanks to my loving wife for her support through this trying time. Her prayers, love, and devotion were a continual encouragement to me. I also wish to thank the other members of the Body of Christ for their prayer support.

Paul A. Moore
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>Abstract</td>
<td>viii</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>I - 1</td>
</tr>
<tr>
<td>Thesis Objective</td>
<td>- 1</td>
</tr>
<tr>
<td>Background</td>
<td>- 1</td>
</tr>
<tr>
<td>Problem Statement</td>
<td>- 9</td>
</tr>
<tr>
<td>Scope of the Thesis Investigation</td>
<td>- 9</td>
</tr>
<tr>
<td>Assumptions</td>
<td>- 10</td>
</tr>
<tr>
<td>Approach</td>
<td>- 11</td>
</tr>
<tr>
<td>Summary</td>
<td>- 12</td>
</tr>
<tr>
<td>II. Requirements Definition</td>
<td>II - 1</td>
</tr>
<tr>
<td>Introduction</td>
<td>- 1</td>
</tr>
<tr>
<td>Environment Considerations</td>
<td>- 2</td>
</tr>
<tr>
<td>Limitations of Microcomputers</td>
<td>- 3</td>
</tr>
<tr>
<td>Environment Goals and Characteristics</td>
<td>- 5</td>
</tr>
<tr>
<td>Minimum Functional Requirements</td>
<td>- 8</td>
</tr>
<tr>
<td>Functional Implementation Priorities</td>
<td>- 11</td>
</tr>
<tr>
<td>Additional Requirements</td>
<td>- 12</td>
</tr>
<tr>
<td>Summary</td>
<td>- 13</td>
</tr>
<tr>
<td>III. Preliminary Design</td>
<td>III - 1</td>
</tr>
<tr>
<td>Introduction</td>
<td>- 1</td>
</tr>
<tr>
<td>MICROSDW Configuration Models</td>
<td>- 3</td>
</tr>
<tr>
<td>Resolution of Requirements</td>
<td>- 11</td>
</tr>
<tr>
<td>User Interface Design</td>
<td>- 12</td>
</tr>
<tr>
<td>Introduction / Background</td>
<td>- 12</td>
</tr>
<tr>
<td>Organization / Current Implementation</td>
<td>- 13</td>
</tr>
<tr>
<td>Improvements</td>
<td>- 13</td>
</tr>
<tr>
<td>MICROSDW Built-in Functions</td>
<td>- 15</td>
</tr>
<tr>
<td>Keyword Menu Structure Installation</td>
<td>- 16</td>
</tr>
<tr>
<td>Keyword Abbreviations</td>
<td>- 21</td>
</tr>
<tr>
<td>Function Key Selection of Keywords</td>
<td>- 23</td>
</tr>
<tr>
<td>Keyword Length Expansion</td>
<td>- 24</td>
</tr>
<tr>
<td>Include File Constant and Type Definitions</td>
<td>- 25</td>
</tr>
<tr>
<td>MICROSDW Structure</td>
<td>- 25</td>
</tr>
<tr>
<td>Summary</td>
<td>- 26</td>
</tr>
<tr>
<td>Section</td>
<td>Pages</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>IV. Detailed Design</td>
<td>IV-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>-1</td>
</tr>
<tr>
<td>Software Development Tool Selection</td>
<td>-4</td>
</tr>
<tr>
<td>Selection Criteria</td>
<td>-5</td>
</tr>
<tr>
<td>Life-Cycle Tool Selections</td>
<td>-7</td>
</tr>
<tr>
<td>Utility Tool Selections</td>
<td>-14</td>
</tr>
<tr>
<td>Install Program Retailed Design</td>
<td>-16</td>
</tr>
<tr>
<td>Static Data Structures</td>
<td>-20</td>
</tr>
<tr>
<td>Dynamic Data Structures</td>
<td>-22</td>
</tr>
<tr>
<td>Static Structure Creation</td>
<td>-26</td>
</tr>
<tr>
<td>Summary</td>
<td>-28</td>
</tr>
<tr>
<td>V. Implementation</td>
<td>V-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>-1</td>
</tr>
<tr>
<td>Implementation Language</td>
<td>-2</td>
</tr>
<tr>
<td>Host Microcomputer/Operating System</td>
<td>-3</td>
</tr>
<tr>
<td>Implementation Standards</td>
<td>-5</td>
</tr>
<tr>
<td>Implementation Strategy</td>
<td>-7</td>
</tr>
<tr>
<td>Summary</td>
<td>-13</td>
</tr>
<tr>
<td>VI. Integration</td>
<td>VI-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>-1</td>
</tr>
<tr>
<td>Integration of Software Development Tools</td>
<td>-1</td>
</tr>
<tr>
<td>Integrating the MICROSDW with Different</td>
<td>-3</td>
</tr>
<tr>
<td>Environments</td>
<td>-7</td>
</tr>
<tr>
<td>Integration with the AFIT DEL SDW</td>
<td>-5</td>
</tr>
<tr>
<td>Summary</td>
<td>-7</td>
</tr>
<tr>
<td>VII. Operations and Maintenance</td>
<td>VII-1</td>
</tr>
<tr>
<td>VIII. Conclusions and Recommendations</td>
<td>VIII-1</td>
</tr>
<tr>
<td>Introduction</td>
<td>-1</td>
</tr>
<tr>
<td>MICROSDW Development Summary</td>
<td>-1</td>
</tr>
<tr>
<td>Recommendations for MICROSDW Expansion</td>
<td>-2</td>
</tr>
<tr>
<td>Summary</td>
<td>-5</td>
</tr>
<tr>
<td>Appendix A Requirements Definition SADTs</td>
<td>A-1</td>
</tr>
<tr>
<td>Appendix B Preliminary Design to Requirements</td>
<td>B-1</td>
</tr>
<tr>
<td>Crossreference</td>
<td></td>
</tr>
<tr>
<td>Appendix C MICROSDW Structure Charts</td>
<td>C-1</td>
</tr>
<tr>
<td>Appendix D MICROSDW System Data Dictionary</td>
<td>D-1</td>
</tr>
<tr>
<td>Appendix E Menu Language Definition</td>
<td>E-1</td>
</tr>
<tr>
<td>Appendix F Microcomputer Software Development Tools And Environments Listing</td>
<td>F - 1</td>
</tr>
<tr>
<td>Appendix G User's Manual</td>
<td>G - 1</td>
</tr>
<tr>
<td>Appendix H Programmer/Maintenance Manual</td>
<td>H - 1</td>
</tr>
<tr>
<td>Appendix I Program Listings</td>
<td>I - 1</td>
</tr>
<tr>
<td>Bibliography</td>
<td>BIB - 1</td>
</tr>
<tr>
<td>VITA</td>
<td>VITA - 1</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I - 1</td>
<td>Software Life-Cycle</td>
<td>I - 3</td>
</tr>
<tr>
<td>III - 1</td>
<td>MICROSDW Configuration Model</td>
<td>III - 5</td>
</tr>
<tr>
<td>III - 2</td>
<td>Example MICROSDW Life-Cycle Tool Configuration</td>
<td>III - 6</td>
</tr>
<tr>
<td>III - 3</td>
<td>Example MICROSDW Utility Tool Configuration</td>
<td>III - 7</td>
</tr>
<tr>
<td>III - 4</td>
<td>Example MICROSDW Built-in Function Configuration</td>
<td>III - 8</td>
</tr>
<tr>
<td>III - 5</td>
<td>Example MICROSDW Hardware Configuration</td>
<td>III - 10</td>
</tr>
<tr>
<td>III - 6</td>
<td>WORDS and NUMBERS example</td>
<td>III - 18</td>
</tr>
<tr>
<td>III - 7</td>
<td>Text Menu Specification</td>
<td>III - 20</td>
</tr>
<tr>
<td>IV - 1</td>
<td>Highlevel Structure of BUILDDAT</td>
<td>IV - 17</td>
</tr>
<tr>
<td>IV - 2</td>
<td>Menu Definition File Example</td>
<td>IV - 19</td>
</tr>
<tr>
<td>IV - 3</td>
<td>Decoding Paths Matching Figure IV-2</td>
<td>IV - 20</td>
</tr>
<tr>
<td>IV - 4</td>
<td>Keyword List Matching Figure IV-2</td>
<td>IV - 21</td>
</tr>
<tr>
<td>IV - 5</td>
<td>Keyword Structure Matching Figure IV-2</td>
<td>IV - 23</td>
</tr>
<tr>
<td>IV - 6</td>
<td>Menu Structure Matching Figure IV-2</td>
<td>IV - 24</td>
</tr>
<tr>
<td>IV - 7</td>
<td>Menu Decoding Pointer Structure Matching Figure IV-2</td>
<td>IV - 25</td>
</tr>
<tr>
<td>V - 1</td>
<td>File Header Template</td>
<td>V - 6</td>
</tr>
<tr>
<td>V - 2</td>
<td>Procedure Header Template</td>
<td>V - 7</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Page</th>
<th>Table Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>II - 1</td>
<td>MICROSDW Hardware Requirements</td>
</tr>
<tr>
<td>IV - 1</td>
<td>Comparison of Pascal/MT+ and TURBO-Pascal</td>
</tr>
<tr>
<td>V - 1</td>
<td>DEC Rainbow-100 Hardware Features</td>
</tr>
</tbody>
</table>
Abstract

The purpose of this investigation is to 1) design an integrated and automated software development environment for microcomputers, 2) to implement a portion of that environment and, 3) to modify the VAX-11/780 VMS Software Development Workbench to support microcomputer workstations.

The central portion of the initial microcomputer software development environment is a flexible user interface adapted from a previous microcomputer software development effort. The user interface is enhanced by an extensive installation program which allows 1) terminal definition, 2) help message specification and, 3) reconfiguration of the keyword menu system.

The result of the investigation is a prototype software development environment hosted on DEC Rainbow 100 microcomputer under three operating systems: CP/M-80, CP/M-86, and MS-DOS. The environment includes a flexible user interface, an installation program, a communications program, and an initial library of reusable software.
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS

Thesis Objective

The objective of this thesis effort is to perform the initial development and implementation of a microcomputer software development environment for the Air Force Institute of Technology (AFIT). This software development environment is entitled the Microcomputer Software Development Workbench (MICROSDW). The MICROSDW supports the development of software from conception through retirement with software tools. The MICROSDW is an extension of the Software Development Workbench (SDW) developed at AFIT in 1982.

Background

The essence of a software development environment is the synergistic integration of tools in order to provide strong, close support for a software project. This environment must have at least these five characteristics: breadth of scope and applicability, user friendliness, reusability of internal components, tight integration of capabilities, and use of a central information repository (29:36). How well the environment supports the software life-cycle from inception of a software project to its termination determines the breadth of scope and applicability of an environment. User friendliness is a subjective measurement of how well the
environment supports the day-to-day activities of the user. Internal components of the environment must be reusable so that the environment can be easily adapted to changing user requirements. The environment must be tightly integrated to improve user friendliness and to present the appearance that the tools are working together. The final and possibly most important characteristic of an effective software environment is that it be coordinated and focused by access to a central repository of data (29:40). An environment is an information utility whose purpose is to provide software workers with the information they need when they need it (29:40). The role of the environment tools is to capture, store, analyze, format, organize, and distribute information.

The Software Development Workbench (SDW) is a prototype integrated software development environment created in 1982 as part of an Air Force Institute of Technology (AFIT) research project directed by Dr. Gary B. Lamont (15). The SDW is implemented on the AFIT Digital Engineering Laboratory (DEL) VAX 11/780 VMS operating system. The SDW stores information about software development projects in separate integrated databases for each project. The tools of the SDW are integrated by the SDW executive (SDWE) program. The SDWE provides flexible access to the SDW tools and data bases through a hierarchical menu system organized into functional tool groups and levels. The tools integrated by the SDW support the entire software life-cycle.

The SDW supports a software life-cycle model composed of
The following phases: requirements definition, preliminary design, detailed design, implementation (coding), integration, and maintenance (See Figure 1). The life-cycle phases are briefly described in the following paragraphs.

Testing and validation is performed during the following phases of the life-cycle to ensure the correctness of each phase in relation to the previous phase and established requirements.

During the requirements definition phase, the detailed requirements for what the software system must do are analyzed and documented. During the preliminary design phase, the requirements are organized into functional areas. The functional areas are decomposed into components and
subcomponents in a hierarchical manner.

During the detailed design phase, the components from the preliminary design phase are decomposed further and algorithms designed to accomplish the function(s) assigned to each component.

The implementation phase involves translating the detailed design of the software system into a programming language and testing components of the system.

During the integration phase, all portions of the software system are brought together as a cohesive unit, installed on the target hardware, and tested.

The operations and maintenance phase is the operational use of the software system by the users and maintenance of the software system. Maintenance includes the correction of errors and addition of new functions to the system. The maintenance phase is an evolutionary phase during which the software system evolves until its eventual termination.

Five objectives of the SDW which support software workers and the software life-cycle model just described are: 1) to reduce the number of software errors, 2) to be responsive to changing requirements, 3) to provide for rapid assessment of design alternatives, 4) to provide interactive automated documentation support, and 5) to provide mechanisms to assist software managers in planning and tracking software development projects (16:4).

Currently the SDW does not support the use of distributed microcomputer workstations as part of the
software development environment. The effectiveness of using distributed microcomputer workstations in integrated software development environments has been reported by several corporations including Xerox (18:377), TRW (3:154), and Boeing Computer Services Company (41:134). These environments are discussed next.

Boeing's ARGUS system is based on CAD/CAM principles applied to "Computer-Assisted Software Engineering." The first version of a comprehensive environment has been implemented on a super-microcomputer, the ONYX C8002, running the UNIX operating system (41:131). ARGUS system tools are arranged in five "toolboxes": 1. Manager's Toolbox, 2. Designer's Toolbox, 3. Programmer's Toolbox, 4. Verifier's Toolbox, and 5. General/Utility Tools. The Manager's Toolbox includes tools for managing project resources and controlling project expenditures (41:131). The Designer's Toolbox includes tools for documenting and controlling the formal design of software systems (41:131). The Programmer's Toolbox contains various coding and debugging tools (41:131). The Verifier's Toolbox contains tools for evaluating software design models and programs (41:132).

A summary of Dr. Leon Stucki's observations about the ARGUS distributed architecture including micro-based workstations follows. First, significant software systems for mainframes can be successfully developed on micro-based systems. Second, the productivity of programmers can be significantly increased when they can use a dedicated modern
software engineering environment. Third, the application of 
CAD/CAM-like principles in the ARGUS environment 
significantly increased analyst productivity (41:134).

The TRW Software Productivity System (SPS) was created 
as a result of a 1980 software productivity study at TRW. 
The conclusions reached by TRW as a result of their study and 
experience in implementing a micro-based local area network 
software engineering environment are an excellent summary of 
current knowledge in the field of distributed microcomputer 
environments. The conclusions, published in the IEEE 
Proceedings of the 1982 International Conference on Software 
Engineering article "The TRW Software Productivity System," 
by Barry W. Boehm et.al. follow (3:155-156):

1. Significant productivity gains require an 
   integrated program of initiatives in several 
   areas.

2. An integrated software productivity improvement 
   program can have an extremely large payoff (a 
   factor of 4 by 1990).

3. Improving software productivity involves a 
   long, sustained effort.

4. In the very long run, the biggest productivity 
   gains will come from increased use of existing 
   software.

5. Software support environment requirements are 
   still too incompletely understood to specify 
   precisely.

6. No single software support system architecture 
   will be optimal for all organizations. For 
   example, the source-target concept of operation 
   most appropriate to TRW is unnecessary for 
   organizations with a single type of target 
   computer.

7. The multiple relational-hierarchical database
concept simplifies many software support functions. It allows the support system to capitalize on the strengths of each type of database while largely avoiding their weaknesses.

8. A rapid-prototyping capability is essential to the evolutionary development of a software support environment. Unix has provided an excellent rapid-prototyping capability.

9. User-interface standards are essential for preserving the conceptual integrity of an evolving support system. An excellent way to implement such standards is to embed them into a family of toolbuilders' utilities supporting error processing, help messages, master database access, forms management, etc.

10. User acceptance of novel development environments is a gradual process which requires careful nurturing by the sponsoring organization. Involvement of the user community in planning the growth and direction of the environment will help ensure their acceptance of it.

One of the SPS-1 systems implemented at TRW includes one DEC VAX 11/780 UNIX source machine, four VAX 11/780 VMS target machines, and five LSI-11/23 based semi-personal microcomputers. The acquisition of the LSI-11/23's represented a compromise between cost and purely personal terminals based on equipment which supported UNIX available during Spring 1981 (3:154).

The Xerox Star system, officially known as the Xerox 8010 Information System, is a workstation for professionals that provides a comprehensive set of capabilities for office and program development environments (18:377). The Star system was developed using an integrated, distributed development environment of personal computers connected by a
local area network (Ethernet). As the Star system was developed it was used to aid in its own development. The Star workstation software was built on a specialized operating system known as Pilot (18:378). The Pilot operating system was undergoing major revisions during the Star development. Pilot had "six major releases during the Star development (18:378)." The Star workstation hardware is composed of: a Xerox 8000 processor; an 8 or 24 megabyte (MB) rigid disk; a floppy disk; connections for Ethernet, optional controllers, and a user terminal; a 22 bit virtual and 20 bit physical address space; typically 512KB of RAM; and a user terminal with an 808 raster by 1024 pixel bitmapped display, a keyboard, and a pointing device called a "mouse" (18:378). File and printer servers on the Ethernet were used to meet the large file storage and printing needs of the Star development. Library servers mediated access to libraries of source files stored on file servers (18:379). A summary of the advantages of personal development environments identified during the Star development follow (18:380):

- Any degree to which a personal machine can approximate the target machine is a major benefit.
- The operational independence of personal computers meant that a single workstation failure affected only one person (system reliability is increased).
- Personal computers provide consistent feedback and response.
- The physical programming environment was free from many traditional restrictions - extreme noise, cold air, etc.
- The power of the personal development environment reduced programmers' reliance on printed output.

- Hiring and retaining talented staff was easier when a programmer could be offered his own personal computer with a powerful set of development tools.

The previous three examples demonstrate that personal microcomputers in a distributed software development environment can be effective in improving the efficiency and productivity of programmers. The examples also illustrate that a variety of microcomputer configurations can be used to create an effective software development environment.

Problem Statement

A life-cycle oriented software development environment, the SDW has been partially implemented on the AFIT Digital Engineering Laboratory (DEL) VAX 11/780 to support AFIT student and faculty software development projects. The objective of this thesis effort is:

1. to examine state-of-the-art software development environments;
2. to analyze the capabilities of microcomputer workstations in the software development environment;
3. to modify the Software Development Workbench to support distributed microcomputer workstations;
4. to design an integrated and automated software development environment for microcomputer workstations;
5. to implement a portion of that environment.

Scope of the Thesis Investigation

The purpose of this thesis effort is to create a software development environment for microcomputers. The software development environment will utilize existing
software development tools where possible to avoid "reinventing the wheel." Thesis research will focus on determining what software development tools can be hosted on microcomputers commonly available to AFIT students and faculty and how the microcomputers should communicate with the SDW. The intention of this thesis effort is to increase the utilization of existing computer hardware and software tools and the productivity of AFIT students and faculty. SDW tools will be rehosted to the microworkstation when necessary and possible. New software tools may be developed to support the microworkstation environment. The preliminary and detailed design for a microcomputer workstation software development environment will be completed and a prototype workstation environment will be implemented.

Assumptions

The prototype microcomputer workstation software development environment will initially be implemented on a microcomputer supporting CP/M-80, CP/M-86, or MS-DOS. Microcomputers supporting these operating systems are commonly available to most AFIT students and faculty and are available to support this thesis effort. The software developed for the microworkstation must be easily transportable to other microcomputers. It is also assumed that the distributed workstation environment is implementable and that it will be an improvement over the existing single computer, single operating system environment (research and
experience will prove or disprove this assumption).

Approach

The thesis effort begins with an extensive literature review of software development environments and the use of microcomputers in those environments. The purpose of the research is to gain an in-depth knowledge of software development environments and the capability of microcomputers to support software development activities. The knowledge gained through the literature review will be applied to the development of the MICROSDW environment.

The design of the MICROSDW, an integrated and automated software development environment for microcomputer workstations follows the life-cycle model described in the background section: requirements definition, preliminary design, detailed design, implementation, integration and maintenance.

During the requirements definition phase, the requirements for the microcomputer software development workstation (MICROSDW) are established and criteria set to determine how well the MICROSDW prototype implementation meets the requirements.

The components of the MICROSDW are decomposed into functional groups during the Preliminary Design phase. The result of this phase is a document with a corresponding data dictionary describing the functional organization of the MICROSDW.
In the Detailed Design phase, a MICROSDW program is designed as an interface to control the tools selected for hosting on the SDW microworkstation. The result of this phase is a document describing the MICROSDW Detailed Design. An implementation language for the MICROSDW program is chosen during this phase, and software development tools are selected for hosting on the MICROSDW.

During the implementation phase, the MICROSDW program is coded and validated. The tools selected for inclusion in the MICROSDW are loaded onto the selected host microcomputer during this phase.

During the integration phase, the MICROSDW environment is integrated with the SDW environment on the DEL VAX 11/780.

The operation and maintenance phase of the MICROSDW development involves using and testing the extended SDW operationally.

**Summary**

The extension of the Software Development Workbench to support distributed microworkstations will provide a significant improvement in the capabilities of the Software Development Workbench. The implementation of a prototype microcomputer workstation integrated with the Software Development Workbench may significantly expand its availability to, and productivity of, AFIT students and faculty.
II. Requirements Definition

Introduction

The purpose of this chapter is to define the requirements for the MICROSDW. This chapter establishes the requirements for the development of a Software Development environment for microcomputers. The requirements are the basis for the following design of the MICROSDW environment. The chapter is organized in seven sections: environment considerations, limitations of microcomputers, environment goals and characteristics, minimum functional requirements, functional implementation priorities, additional requirements, and summary.

The Environment Considerations section discusses the impact the destination environment has on establishing the requirements for the MICROSDW. The limitations of microcomputers section briefly discusses some of the limitations of microcomputers today and the affect of the limitations on the environment. The environment goals and characteristics section describes the goals and characteristics desired in software development environments and software development tools. The section on minimal functional requirements describes the functional capabilities necessary to meet the goals and characteristics of a software development environment. The section on implementation priorities discusses the priority for implementing the different portions of the MICROSDW. The
additional requirements section discusses the requirement for portability of the MICROSDW and guidelines for the computer hardware required to support the MICROSDW.

Appendix A presents a graphical representation of the requirements for the MICROSDW user interface. Structured Analysis and Design Technique (SADT) activity diagrams were chosen as the graphical representation because they show decomposition of activities; decomposition of information into input, output, and control; and hierarchical relationships in decomposition of activities.

Environment Considerations

The establishment of clear, concise, detailed requirements is one of the keys to a successful software development project. It is also one of the most difficult tasks of the development process. The definition of requirements for a software development environment is specially difficult because of the varying environments into which the programming support environment itself must fit and the processes it must support. A noted computer scientist, Leon Osterweil said,

"The task of creating effective [software development] environments is so difficult because it is tantamount to understanding the fundamental nature of the software processes" (29:36).

The software development environment must be tailored to the environment it must support. The environment it must support is composed of people, policies, procedures,
methodologies, budgets, projects, information, existing software, and existing facilities (33:14).

To further complicate the definition of requirements for a software development environment each phase of the software life-cycle may be supported with a different methodology. Requirements may be specified using a Requirements Specification Language or a Structured Analysis and Design Technique (SADT). Preliminary Design may be supported using Jackson's Method (32:152; 22). Detailed Design may be accomplished using a Program Design Language (PDL). The key to supporting an overall life-cycle methodology composed of separate methodologies is the ability to trace and transform data from one phase of the life-cycle to the next without loss of information, or completeness and consistency.

The software life-cycle is an iterative process composed of iterative processes. During software development information will be concurrently changing throughout the life-cycle phases. This compounds the need to trace information throughout the life-cycle and to determine the affect that changing information in one phase has on the phases preceding and following it.

**Limitations of Microcomputers**

The requirements for a software development environment for microcomputer workstations does not differ from those for an environment implemented on a larger computer. However, the limitations of a host computer will affect the size of...
the software tools which may be implemented, the amount of
information which may be available for immediate access, and
how quickly information can be processed.

The primary limitations of microcomputers are memory
addressing, secondary storage capacity, and operating system
capabilities. The limitations which microcomputer hosts
impose on the implementation of software development
environments are continually decreasing as the capabilities
of microcomputers and their peripheral devices increase. For
example, secondary hard-disk storage of 165 Megabytes may be
purchased for under $5000 (10:89).

The capabilities of common microcomputer operating
systems such as CP/M and MS-DOS are significant but they lack
many of the capabilities of larger operating systems. The
most severe limitation of current microcomputer operating
systems is the lack of a hierarchical file structure and
storage space to facilitate separation and classification of
data. The second most severe limitation of microcomputer
operating systems is the lack of support for creation and use
of flexible, interactive "script" or "macro" files to use for
controlling execution of programs. Microcomputer operating
systems are also limited in their file access methods, file
protection facilities, the ability to run multiple processes,
and support interprocess communication. New versions of
existing microcomputer operating systems and new operating
systems are constantly reducing the former limitations of
microcomputers.

The requirements for the MICROSDW are also guided by the desire to have the MICROSDW function as an extension of the AFIT Software Development Workbench (SDW). For the MICROSDW to be an extension of the SDW it should have a similar user interface and implement a subset of the SDW software tools augmented by unique MICROSDW functions. The user interface will be similar in appearance, but not in implementation, due to the limitations and capabilities of the microcomputer host. A consistent user interface has the advantage of reducing user training time and confusion when switching from one programming environment to another.

Environment Goals and Characteristics

The following paragraphs outline the goals to be achieved through the use of an integrated software development environment; the characteristics required in a software development environment; and the characteristics required in the tools the environment is composed of. The goals and characteristics which follow have been consolidated from a number of sources (15, 11, 29, 39, 14).

The primary goals of an integrated software development environment follow:

1. To improve the productivity of software developers in all stages of the software life-cycle.

2. To produce flexible, reliable software with a minimum of errors that meets the needs (requirements) of the user (15, 32, 34, 40).
3. To increase the use of existing software.

4. To provide for smooth transitions between life-cycle phases and traceability of information from one phase to the next.

5. To provide automated creation and maintenance of life-cycle documents (15:35).

6. To reduce the time necessary to train software developers.

7. To increase the availability of information about software development projects to the developers and management.

**Software Development Environment Characteristics.** The key characteristics of the development environment are the next lower level of requirements in a "requirements hierarchy". The characteristics of the environment indicate what character traits the environment must have to satisfy the goals of the environment. Following is a list of characteristics a software development environment must have:


2. Must provide for automated and traceable transitions between the life-cycle phases and products (43:8; 14:50).

3. Must be flexible, allowing for the addition, deletion, or modification of the components (tools) of the environment (11:24; 15:34,43).


5. Must provide on-line Help and/or Tutorial facilities (11:24; 39:232).

6. Must support interactive software development.

8. Must provide a means of storing and updating software development information in a central repository for project information (11:8,16; 29:39).


10. Must support pre-fabrication and reuse of software (29:38; 15:39).

11. Must provide for checking the completeness and consistency of life-cycle products (15:41).

12. Must provide for communication (information transfer/sharing) with other computer systems (14:51).

Software Tool Characteristics. The characteristics of individual software tools are the next level of requirements below the software environment characteristics and environment goals. The software tools must have these characteristics to meet the goals and required characteristics of the environment:

1. Should provide for graphical display of information.

2. Must provide on-line Help and/or Tutorial facilities (14:50; 39:232).

3. Should be adaptable to the experienced and inexperienced user (14:50; 33:14).

4. Must be functionally complete (i.e., support a single or small number of functions) (14:50; 11:24).

5. Must not require knowledge of how to use another software tool to use it.

6. Must provide for collection of "useful" information about its use/user (14:50; 11:25).

7. Must be consistent with other tools in the environment (user interface, functionality, communication protocol) (14:50; 11:14,24).
Minimum Functional Requirements

Certain functional capabilities are necessary to meet the goals and characteristics required in a software development environment. The MICROSDW implements a subset of the capabilities of the SDW. The capabilities chosen for the MICROSDW are designed to meet the goals and characteristics discussed earlier. The functional capabilities chosen for the MICROSDW are designed to provide automated support for all phases of the software life-cycle with additional utility functions to support the "everyday" activities of the user. The functional requirements for the MICROSDW environment follow.

User Interface. This function is required to provide the user with a "friendly" interface to the software tools, information, and operating system of the MICROSDW. The user interface shall provide the following capabilities:

1. To execute MICROSDW tools (programs).
2. To Locate MICROSDW programs, documents, or data files.
3. To view "Help" or "Tutorial" information the operation of the MICROSDW and the tools it is composed of.
4. To modify the configuration of the MICROSDW environment, including:
   - Addition, Deletion, and Movement of software tools
   - File manipulation (Delete, Move, Copy, View)
5. To communicate with the user through interactive prompts and responses on a view screen.

Requirements Specification. This function shall provide for the entry, modification, deletion, tracing, printing, and display of requirements.
Structure Charts. This function shall provide support for the Preliminary Design phase of the software life-cycle. The structure charting function shall provide the following capabilities:

1. Create structure chart nodes and data element names.
2. Delete structure chart nodes and data element names.
3. Modify structure chart nodes and data element names.
4. Display structure chart nodes and data element names.
5. Print structure chart nodes and data element names.
6. Display a parent node and up to seven child nodes.
7. Print a displayed structure chart.
8. Numbering of nodes.
9. Ability to distinguish predefined child nodes.
10. Create "skeletal" data dictionary entries.
11. Create "skeletal" PDL.

Data Dictionary. This function provides support for all phases of the software life-cycle, particularly the Preliminary and Detailed Design phases. Data Dictionaries shall provide a central repository for data about a software project. The Data Dictionary functions shall provide the following capabilities:

1. Create data dictionary entries.
2. Delete data dictionary entries.
3. Modify data dictionary entries and attributes.
4. Display data dictionary entries.
5. Print data dictionary entries.
7. Display selected data dictionary entries.
8. "Batch" print groups (ALL, RELATED, SPECIFIC list, or TYPE) of data dictionary entries in alphabetic or hierarchical order.
9. Print/Display a "who-calls" structure for a given entry (similar to listing related entries).

Program Design Language (PDL). This function shall provide support for the detailed design and coding phases of the software life-cycle. The PDL function shall provide the
following capabilities:

1. Creation of PDL files.
2. Interactive syntax directed editing of PDL files.
3. "Structured" printing of PDL files.
4. Definition of the PDL grammar.
5. Modification of the PDL grammar.

Coding Support. This function shall provide software tools to support the coding phase of the software life-cycle. This function shall provide at least one of each of the following tools: a high level language (e.g. Pascal, Fortran, C, etc.) compiler, assembler, linker, and debugger. Optionally, this function may provide cross-referencers, cross- compilers, and cross-assemblers to support code generation for other computers.

Source Code Control System. This function shall provide the capability to store and retrieve multiple on-line versions of files including, but not limited to, source code, assembler code, manuals, documents, and text files.

Utility Functions. These functions provide support for the miscellaneous activities of the user. The following functions were selected because of their ability to support one or more activities of the user, and because they have the potential to save the user much "busy" work.

Word Processing. This function shall support the text editing, formatting, and printing requirements of the user.

Communications. This function shall provide the user with the capability to communicate with remote computers.
through a modem over standard phone lines. The function will provide the user with the capability to send and receive files, and act as a "dumb" terminal to a remote computer.

**File Compare.** This function shall provide the user with the capability to compare two text or two binary files. The function shall display the differences in the files in a readable format. The differences between two files shall be displayed on a display screen or written to a file.

**Database Management.** This function shall provide the user with a general data management capability. The Database Management function shall provide the user with the ability to create, modify, delete, and combine database files and information. The Database Management function may be used to store information such as Software Trouble Reports, code status, or data dictionary information.

**Functional Implementation Priorities**

The priority for implementation of the minimum functional requirements is:

1. User Interface
2. Coding Support
3. Data Dictionary
4. Program Design Language
5. Structure Charts
6. Requirements Specification
7. Source Code Control System
8. Utility Functions

The User Interface is given the highest priority because it is required to integrate or tie the MICROSDW together as a cohesive environment. The Coding Support function was listed
second so that the environment can start supporting its own development as soon as possible. Also, the Coding Support functions can be provided by existing software. Using the User Interface to initiate the coding support functions will provide early feedback on the performance of the User Interface. The Data Dictionary function was chosen next because of its support for the documentation needs of the software life-cycle and its support of the Structure Charting and PDL functions. The priorities for the rest of the functions was designed to expand environment support for the software life-cycle around the coding phase. The Utility Functions are listed last because most of these functions can be supplied by existing software packages.

Additional Requirements

Portability. One of the goals in developing the MICROSDW system is for it to be available to a large number of people on various microcomputers. Thus, portability to other operating system and hardware environments is required. The portability of a system is determined by how dependent the software is on unique features of the language it is written in, and the operating system and hardware the software runs on. The portability of the MICROSDW can be increased by minimizing and localizing these dependencies. The choice of a host operating system and implementation language is addressed in chapter III.

MICROSDW Hardware Requirements. The MICROSDW
environment is designed to be implemented on a variety of microcomputers. Table II-1 illustrates basic and optional hardware characteristics required in a host microcomputer. The MICROSDW can be hosted on a microcomputer with less than the basic hardware characteristics but performance of the environment may be degraded. For the MICROSDW to function as an extension of the SDW the host microcomputer must have the hardware capability to communicate with the SDW either through a direct connection or remotely through telephone lines with a modem.

Summary

This chapter establishes the requirements for the AFIT Microcomputer Software Development Workbench. The requirements are a guide for the subsequent design and development of the MICROSDW to be measured against. The requirements described are high-level because most of the functional capabilities of the system will be provided by existing software packages. The User Interface function is the first capability to be developed. The User Interface function is required to be very flexible to simplify later modifications to the environment.
# Table II-1
MICROSDW Hardware Requirements

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
<th>Necessary To:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>64 Kilobytes</td>
<td>Support moderate sized program development.</td>
</tr>
<tr>
<td></td>
<td>256 Kilobytes (optional)</td>
<td>Support larger program development.</td>
</tr>
<tr>
<td>Disk Storage</td>
<td>2 Floppy Disk Drives</td>
<td>Provide secondary storage for programs and data.</td>
</tr>
<tr>
<td></td>
<td>Hard Disk Drive (optional)</td>
<td>Increase secondary storage and decrease program/data access times.</td>
</tr>
<tr>
<td>Display Screen</td>
<td>24 lines by 80 Columns</td>
<td>Provide for interactive display of information.</td>
</tr>
<tr>
<td></td>
<td>Cursor Positioning, Reverse Video, Bold</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monochrome or Color Graphics (optional)</td>
<td>Increase user/computer interface bandwidth.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provide for display of graphical software development information.</td>
</tr>
<tr>
<td>Printer</td>
<td>80 Column Dot Matrix</td>
<td>Provide for printing software development documents.</td>
</tr>
<tr>
<td></td>
<td>Graphics Capable (optional)</td>
<td>Provide for printing graphical software development documents.</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Basic Keyboard, including Cursor Control</td>
<td>Provide for user interaction with the operating system and programs.</td>
</tr>
<tr>
<td></td>
<td>and Function Keys</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>Description</td>
<td>Necessary To:</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mouse</td>
<td>Position Pointer on Display Screen (optional)</td>
<td>Provide for increased user/computer interface capabilities and manipulation of graphic displays.</td>
</tr>
<tr>
<td>Ports</td>
<td>1 Parallel or Serial Port for Printer</td>
<td>Provide for printing documents.</td>
</tr>
<tr>
<td></td>
<td>Additional Ports (optional)</td>
<td>Provide for possible access to remote, local, or network computer systems. Provide for additional peripheral devices.</td>
</tr>
<tr>
<td>Modem</td>
<td>300/1200 Baud (optional)</td>
<td>Provide for possible access to remote computer systems.</td>
</tr>
</tbody>
</table>
III. Preliminary Design

Introduction

The purpose of the Preliminary Design phase of the MICROSDW is to organize the characteristics and functions described during the Requirements Definition phase into a functional structure. The functional structure defined during this phase forms the basis for more detailed development during the Detailed Design phase.

This chapter is divided into five sections, Introduction, MICROSDW Configuration Models, Resolution of Requirements, MICROSDW User Interface Design, and Summary. The MICROSDW Configuration Models section describes the overall software and hardware organization of the MICROSDW. The Resolution of Requirements section describes how the requirements for the MICROSDW are resolved by the functions included in the MICROSDW. The main product of the MICROSDW development is a flexible User Interface and a collection of tools supporting software development on microcomputers. The MICROSDW User Interface Design section describes the background, organization, functions, and improvements of the User Interface.

The result of the Preliminary Design phase is a Preliminary Design Document. The Preliminary Design Document for the MICROSDW is composed of this chapter and supplementary information in Appendix C and D.

A critical decision made during the Preliminary Design
phase concerns what method to use for describing the structure of the software system. The following criteria are used in selecting the method of describing the MICROSDW:

- The method must graphically portray the design.
- The method must show the scope of control of a module.
- The method must show the calling order of modules.
- The method must show the decomposition of modules.
- The method must identify input and output to modules.
- The method must use concise names to label modules, input, and output.

The term "module" above refers to a "set of lexically contiguous program statements that can be referred to by name" that constitutes a part of the total structure of a software system (32:60). During Preliminary Design modules appear to be black boxes as little is known about the processes they contain or the details of their eventual implementation. As the software design process continues through Preliminary Design and Detailed Design the designer's thinking matures and refinements are made, such as further decomposition of the modules or pseudo code describing eventual program statements (32:61).

Structure Charts with a supplemental Data Dictionary were chosen to describe the Preliminary Design structure of the MICROSDW. Structure Charts were chosen because they graphically show the decomposition and scope of control of modules; they identify input and output data and control; they show the calling order of modules; and they use a concise naming and numbering scheme to identify modules and data items. A Data Dictionary was chosen to supplement the Structure Charts to document additional information about the
software modules and data items in an organized fashion. The Data Dictionary documents: descriptions of modules and data items; the decisions made within modules; the composition and purpose of data and control items passed between modules; the internal function(s) of a module; and where a module or data item is used within the software system. The Data Dictionary also provides a means of transferring information from the Preliminary Design phase to the following phases of the software life-cycle including the Detailed Design, Coding, Testing, and Maintenance. The Structure Charts for the MICROSDW are located in Appendix C. The Data Dictionary for the MICROSDW is located in Appendix D. The highest level of decomposition of the MICROSDW will be discussed next using configuration models.

MICROSDW Configuration Models

Configuration models show the highest level of organization of the MICROSDW. The configuration models illustrate the conceptual organization of the MICROSDW software, data, and hardware. They illustrate the relationship of the user to the MICROSDW environment. They also illustrate the relationship between MICROSDW software tools and the data they manipulate. The MICROSDW configuration models are divided into four sections; the User Interface, Life-cycle Tools, Utilities, and Built-in Functions (See Figures III-1, 2, 3, 4). The four sections were chosen because of the broad functional capabilities.
they represent; the User Interface for its capability of interfacing with the user; the Life-cycle Tools for their support of the software life-cycle; the Utilities for their support of miscellaneous functions associated with software development and; the Built-in Functions for their supplying additional capabilities to the User Interface.

The Hardware Configuration of the MICROSDW will vary for different host microcomputer systems. The hardware requirements for a microcomputer software development system, Figure III-5 graphically illustrates a possible hardware configuration. A basic configuration includes a system unit, one or more floppy disk drives, a video monitor, a keyboard, and a printer. A system unit is necessary to house the host CPU and the hardware supporting the CPU. One or more floppy disk drives are required to store programs and data. Other storage media such as a hard-disk may be utilized to increase the ability of the MICROSDW to store and access programs and data. A monitor and keyboard are required so the user can communicate with the programs executed on the computer. A printer may be necessary so the output of the programs (data) can be recorded for later analysis by the user or for distribution to other people.
Figure III-1: MICROSDW Configuration Model
Figure III-2: Example MICROSW Life-cycle Tool Configuration
Figure III-3: Example MICROSDW Utility Tool Configuration
Host microcomputer systems may also include a color graphics monitor, a hard-disk drive, modem, local area network connection, and additional communications or input/output devices (See Figure III-5). The additional hardware such as the color monitor, modem, and local area network connection can increase the ability of the user to communicate with the host microcomputer and other computer systems but are not required for a minimal microcomputer software development system. The following section discusses how the MICROSDW system design fulfills its requirements.
Figure III-5: Example MICROSDW Hardware Configuration
Resolution of Requirements

In Chapter II, Requirements Definition, a hierarchy of requirements for the MICROSDW was described:

- GOALS OF SOFTWARE DEVELOPMENT ENVIRONMENT
- SOFTWARE DEVELOPMENT ENVIRONMENT CHARACTERISTICS
- SOFTWARE TOOL CHARACTERISTICS
- MINIMUM FUNCTIONAL REQUIREMENTS

A cross-reference of the requirements to functions and functions to requirements is located in Appendix B. The minimum functional requirements are selected to meet the goals and characteristics of the software development environment. The software tool characteristics are those traits desired in all software tools. The minimum functional requirements are, in part, a resolution of the high level requirements for the MICROSDW. The functions chosen as part of the minimum functional requirements were selected to support all phases of the software life-cycle and to provide a basic environment for software development hosted on a microcomputer. The characteristics of a software tool must be planned for during Preliminary Design and validated in the resolution of requirements. The characteristics and functions of the software tool are then expanded in the
Detailed Design phase, implemented in the Coding phase, and tested in the Testing phase.

The initial MICROSDW design provides for a User Interface to be built to integrate a collection of software development tools, utilities, and functions. The software development tools and utilities may be executed through the User Interface or directly using the host operating system. The ability to execute the software tools through the User Interface or directly through the host operating system gives the user flexibility to use the development environment in the manner most efficient or comfortable to the user. The User Interface provides the user with prompts, menus, help messages, and file manipulation capabilities. These are designed to help the inexperienced user and to aid the experienced user in their interaction with the software development environment. The experienced user may at times find it more expedient to use the software development tools directly without the overhead of an additional interface.

User Interface Design

Introduction/Background. One of the primary goals of a software development environment is to increase the use of existing software. The MICROSDW User Interface puts this goal into action! An existing software system (30) was identified at AFIT which implemented a menu (command / keyword) driven user interface which could be adapted and improved to provide the User Interface for the MICROSDW. This
system will subsequently be referred to as the "CONTROL-CAD" system. The CONTROL-CAD system was originally hosted on a Z-80 based microcomputer system running CP/M and written in Digital Research's PASCAL/MT+ language. The CONTROL-CAD system will be rehosted to other microcomputers and modified to become the MICROSDW User Interface. The CONTROL-CAD system will be rehosted to show that it is portable and to facilitate its further development. The CONTROL-CAD system will be modified to improve its portability, functional capabilities and, user interface characteristics as described in the following sections. A brief overview of the CONTROL-CAD system follows.

Organization/Current Implementation. The CONTROL-CAD system can be divided into four main sections: 1) installation, 2) initialization, 3) User Interface and, 4) CONTROL-CAD functions. The installation program (BUILDDAT) builds the initialization files (DATA.SYS, HELP.SYS) for CONTROL-CAD from ASCII text files. The initialization procedures initialize the User Interface data structures describing the characteristics of the hardware and the menu structure from the initialization files. The User Interface prompts the user for inputs by displaying the valid options, reads user inputs, validates user inputs, and executes the functions selected by the user.

Improvements. The CONTROL-CAD user interface and installation program provide an excellent building block for creating an even more friendly and adaptable user interface
for the MICROSDW and potentially for other software systems. The CONTROL-CAD user interface will be referred to as the MICROSDW User Interface or simply the User Interface. The CONTROL-CAD installation program will be referred to as "the Install program" or BUILDDAT. BUILDDAT will also be the installation program for the MICROSDW User Interface.

There are six proposed improvements to the User Interface and Install programs: 1) new functions to support the MICROSDW, 2) simplification of the specification method for the keyword menu structure, 3) consistent abbreviation of all keywords, 4) function key selection of keywords, 5) keyword length expansion and, 6) implementation of "include" files to describe the MICROSDW global constants and type definitions.

Other improvements which could be made to the menu system include: 1) global definition of alternatives (keywords) included in all menus, 2) definition of individual help and error messages for each menu option, or 3) entry of non-keyword options such as filenames or numbers with automatic syntax checking entered options. These improvements are discussed in the paper "MGEN - A Generator For Menu Driven Programs," by Bertil Friman (11a). These improvements would significantly increase the power of the MICROSDW. These features were not included in the design of the MICROSDW because they would require more resources (time, people) than are available for this thesis effort. The six proposed improvements to the User Interface and Install
programs are discussed in the following paragraphs.

**MICROSDW Built-in Functions.** The MICROSDW User Interface is responsible for interfacing with the user to execute the functions selected by the user. Since the MICROSDW performs different functions than the CONTROL-CAD system the functions to support MICROSDW requirements must be developed and included in the user interface.

The following table lists the additional functions to be built into the User Interface:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW</td>
<td>display a file on the users screen</td>
</tr>
<tr>
<td>TYPE</td>
<td>(same as view)</td>
</tr>
<tr>
<td>PRINT</td>
<td>print a file</td>
</tr>
<tr>
<td>RENAME</td>
<td>to give a file a new name</td>
</tr>
<tr>
<td>DELETE</td>
<td>to delete a file</td>
</tr>
<tr>
<td>DIR</td>
<td>to show a listing of the files on a disk</td>
</tr>
<tr>
<td>RESTORE</td>
<td>to allow a deleted file to be restored</td>
</tr>
<tr>
<td>COPY</td>
<td>to copy one or more files to a new file</td>
</tr>
<tr>
<td>FIND</td>
<td>to search the current disks for a file</td>
</tr>
<tr>
<td>HELP</td>
<td>to access a menu of &quot;help&quot; information</td>
</tr>
<tr>
<td>RUN</td>
<td>to execute other programs</td>
</tr>
</tbody>
</table>

This is not an all inclusive list of functions for eventual inclusion in the User Interface. These functions are selected to provide the user with capability to retrieve basic information about the environment and to initiate other processes from the User Interface.

Two alternatives are available for implementing the functions, one method is to build the functions into the User Interface, the other method is to implement the functions as external programs executed from the User Interface.

Built-in functions have several advantages over external
programs. One advantage is that once the user is using the User Interface it is faster to execute built-in functions than external programs. The reason built-in functions will execute faster than external programs is because internal functions require (at the most) a program overlay to be loaded into memory from disk and a subroutine call to the desired function. External programs require a subroutine call to initiate the external function. The subroutine call may or may not require a program overlay to be loaded from disk depending on how the User Interface is organized. Under CP/M the subroutine to execute the external program must build a "SUBMIT" or "BATCH" file ($$$.SUB) and terminate execution of the User Interface. The operating system must read the $$$.SUB file, interpret the file, execute the desired program, and then execute the User Interface to return the user to the original environment. This shows that it is desirable to build commonly used functions (including access to operating system functions) into the User Interface.

The replacement of the CONTROL-CAD functions initiated from the User Interface with MICROSDW functions necessitates the replacement of the CONTROL-CAD keyword structure with a keyword structure for the MICROSDW.

An improvement to the method of installation of the keyword menu structure is discussed next.

Keyword Menu Structure Installation. The keyword menu structure is currently implemented in two files, WORDS
and NUMBERS, which describe the menu structure. The WORDS file is simply a list of keywords. A number is assigned to each keyword according to its position in the list. The NUMBERS file is a sequence of records which define the menu decoding paths (Par83:V-18). Each record contains the number of the keyword for that record, the number of the record for the next keyword in that particular menu, and the number of the record for the first keyword in the next lower level of the menu structure. Making a change to the menu structure is a labor intensive task even if a person has a complete understanding of the menu structure and the shared path menu decoding algorithm. Figure III-6 Shows an excerpt from the WORDS and NUMBERS files. Simplification of the menu definition method by having the installation program analyze the menu structure and construct the decoding paths would significantly reduce the time necessary to reconfigure the User Interface to support different menu structures. This improvement is also required to meet the requirement for the MICROSDW to be flexible and easily reconfigurable (Appendix B Requirement 2.3).
### WORDS File Example

<table>
<thead>
<tr>
<th>rec#</th>
<th>word#</th>
<th>matchp</th>
<th>nomatchp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>22</td>
<td>9999</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>18</td>
<td>9999</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>17</td>
<td>9999</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>16</td>
<td>9999</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>4</td>
<td>9999</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>2</td>
<td>9999</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>8</td>
<td>9999</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>16</td>
<td>9999</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>12</td>
<td>9999</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>13</td>
<td>9999</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>14</td>
<td>9999</td>
</tr>
</tbody>
</table>

### NUMBERS File Example

Figure III-6: WORDS and NUMBERS Example
; Start of Example Menu Structure
; Define Root Menu
Setup
LifeCycle
Help
Exit

; Define Call routines for Options without submenus
.Setup = Setup
.Exit = STOP

; Define LifeCycle options
$LifeCycle
Require
Structure
PDL
Coding
Testing
Planning

; Define Call routines for LifeCycle options
.LifeCycle.Require = Require
.LifeCycle.Coding = Coding
.LifeCycle.Testing = Testing
.LifeCycle.Planning = Planning

; Define LifeCycle.PDL options
$LifeCycle.PDL
Pre-Fab
ProjA
ProjB

; Define Call Routines for LifeCycle.PDL
.LifeCycle.PDL.Pre-Fab = Design
.LifeCycle.PDL.ProjA = Design
.LifeCycle.PDL.ProjB = Design

; Define LifeCycle.Structure Options, share with PDL
$LifeCycle.Structure = LifeCycle.PDL

; Define Help Options
$Help
MICROSDW
LifeCycle

; Define Help Call Routine
.Help.MICROSDW = Help
.Help.LifeCycle = Help


Figure III-7: Text Menu Specification
Menu definition analysis is separated into three parts, input, process, and output. During input, the menu definition is read from a file, the unique menu keywords are identified and stored in a data structure and, a hierarchical menu structure is created from the definition. During processing, keywords are assigned numbers, and the decoding paths are created from the hierarchical menu structure. During output the list of words and the decoding paths are written to the DATA.SYS file for use by the User Interface menu system.

The data structures used to transfer the keywords and decoding paths to the User Interface were retained from the original CONTROL-CAD implementation. This isolates the changes required to implement the new menu definition method to the installation program. Alternative methods of representing the keyword menu structure in the User Interface were explored by Parisi (30:V). He concluded that the shared path menu structure with one-time storage of unique keywords resulted in minimal storage requirements for the menu structure.

The next improvement discussed is implementation of consistent keyword abbreviations.

**Keyword Abbreviations.** The CONTROL-CAD system implements abbreviations of keywords by including specific abbreviations in the menu structure. To select a keyword option the user must either input the entire keyword or a specific abbreviation of the keyword if one exists. Two
alternatives to this method of implementing abbreviations are, global abbreviations for all keywords, or variable abbreviations for the keywords in each individual menu.

Global abbreviation of keywords means that a keyword always has a fixed minimum abbreviation necessary to identify it no matter where it occurs in the menu structure. Variable abbreviation of keywords in each menu indicates that for a particular menu of keywords each keyword has a minimum abbreviation necessary to identify it in that menu. In this scheme the same keyword can have more than one abbreviation depending on which menus it occurs in.

The memory and processing time requirements for these two methods of abbreviating keywords are significantly different and affect the design decision about which method to implement.

There are three choices for when to calculate the abbreviations for keywords: 1) during installation, 2) during User Interface initialization and, 3) dynamically during interpretation of user inputs.

If the abbreviations are calculated during installation, or initialization additional memory is required to store the abbreviation length for each keyword. Global abbreviations require one additional memory location for each keyword to store the minimum abbreviation length. Variable abbreviations require one additional memory location for each keyword in each menu. The memory requirement for variable abbreviations is significantly greater than for global
abbreviations because each keyword can appear in more than one menu. The processing required for calculating abbreviation lengths during installation or initialization is the same. Calculation of abbreviation lengths during initialization would increase the User Interface initialization time. Dynamic calculation of abbreviations for each menu during interpretation of user inputs would reduce the memory required of store abbreviation lengths but would increase the memory required for algorithms to calculate the abbreviation lengths. More importantly, dynamic calculation of abbreviations could slow down the validation of user inputs to the point where the interface is less responsive.

Global abbreviations are chosen for the User Interface for the following reasons. It has the smallest storage requirements; each keyword has a consistent abbreviation; and the additional processing required in the User Interface is the smallest. Calculation of the length of the abbreviations will be done during installation to reduce initialization time for the User Interface.

The next improvement discussed is function key selection of menu options.

Function Key Selection of Keywords. This improvement to the menu system allows selection of menu options by moving the cursor (via cursor positioning keys) to the desired keyword and then using a function key to select that option. A different function key could be used to allow
the user to select "help" on a menu option before actually selecting the option. These two options require additional routines in the User Interface to allow movement of the cursor to select an option for execution or for help information. The ability to select help information on options before executing them would enhance the friendliness of the User Interface significantly and allow the user to learn the system quickly with fewer errors. The addition of menu option selection via cursor positioning does not require additional information to be included in the decoding path structure. Selection of help information for a keyword from any menu requires a help message number to be added to each decoding path record. The help message number is required so that the help text can be retrieved from the help message file and displayed on the user's screen. To facilitate the portability of the MICROSDW the definition of the character sequences generated by function keys is included in the installation text file KEYBOARD.TXT. The KEYBOARD.TXT file is read by the installation program and the information is included in the DATA.SYS file.

**Keyword Length Expansion.** The current keyword menu implementation allows keywords with a maximum length of nine characters. This is not a serious limitation but it does prohibit the use of filenames as keywords for most microcomputer operating systems. Standard CP/M or MS-DOS filenames can be up to twelve characters. Increasing the maximum length of the keywords will have a direct impact on
the memory required to store the keywords. Currently to store 100 nine character keywords (10 bytes of storage per word including the string length) requires 1000 bytes of storage. 100 twelve character keywords (13 bytes per word) requires 1300 bytes of storage, a 30% increase in storage for a 33% increase in word length.

Include File Constant and Type Definitions. A major drawback in the current implementation of the CONTROL-CAD routines is the lack of use of "include" files to define constants, variable types, and global variables used by the routines. Thus, to change the length of the keywords the "wordlength" constant must be changed in each of the modules that use it and the modules must be recompiled. If "include" files are used to define the constants, variable types, and global variables, the appropriate include file would be changed and the affected modules recompiled. Only one change to the code is required instead of many. Include files also help to insure the consistency of constants, type definitions, and global variables in separately compiled modules. This is specially advantageous when variable types are not checked by the compiler or linker across separately compiled modules.

MICROSDW Structure. Since the MICROSDW system is reusing the CONTROL-CAD system software structure the MICROSDW User Interface structure is generally the same. The MICROSDW structure will be different due to the addition of new capabilities to the User Interface and BUILDDAT.
programs. The MICROSDW User Interface will include additional routines for manipulating the cursor and matching the cursor position to the menu options (keywords). The structure of BUILDDAT will be changed extensively to include automated generation of the keyword decoding paths used by the User Interface.

Summary

This chapter lays the ground work for the MICROSDW User Interface and modified BUILDDAT detailed design. A general overview of the Software and Hardware configuration of the MICROSDW is discussed. The requirements stated in the preceding Requirements Definition Chapter and the functional requirements are cross-referenced in Appendix B to provide traceability between the Requirements Definition and Preliminary Design phases. The design of the MICROSDW is discussed including background information about the CONTROL-CAD system software being reused in the MICROSDW. Appendix C, MICROSDW Structure Charts, and Appendix D, MICROSDW System Data Dictionary, record the structure of the MICROSDW User Interface and BUILDDAT programs as well as the data structures used by the programs. The following chapter, Detailed Design, further expands on the Preliminary Design.
IV. Detailed Design

Introduction

The purpose of the Detailed Design phase of the MICROSDW is to select software development tools to support the various phases of the software life cycle and to transform the structure and functions of the MICROSDW User Interface and Install programs into detailed algorithms. The result of the Detailed Design phase is a Detailed Design document composed of this chapter and appendicies C, D, and E. The detailed algorithms created during this phase are the basis for the Implementation phase.

This chapter is divided into five sections: Introduction, MICROSDW Component Selection, Install program Detailed Design, User Interface Detailed Design, and Summary. The selection of software tools to support the MICROSDW environment is discussed in the MICROSDW Component Selection section. The Install program Detailed Design section describes the development of a keyword menu definition language and changes to an existing Install program. The User Interface Detailed Design section describes additional algorithms and modifications required to implement the improvements discussed in Preliminary Design. The remainder of the Introduction discusses the relationship of the Detailed Design phase to Preliminary Design and Implementation, and documentation of the Detailed Design phase.
**Detailed Design to Preliminary Design and Implementation.** The result of the Preliminary Design phase is the initial input to the Detailed Design phase. During Detailed Design the structure and functions of the software system are refined. The refinement process adds new information to the design such as detailed algorithms. The Preliminary Design Documentation forms a baseline for documenting the Detailed Design. The baseline documentation is then modified to reflect the new information developed during Detailed Design.

The distinction between Preliminary and Detailed Design is not very clear, because the software designer deals with similar problems during both phases. The primary distinction between Preliminary and Detailed Design is the level of abstraction dealt with in each phase. During Preliminary Design the basic functions and structure of the software system are being derived from the Requirements. During Detailed Design the results of Preliminary Design are being refined and expanded. Detailed algorithms are developed or selected to accomplish the functions defined during Preliminary Design. Preliminary Design may also deal with the refinement of the software structure and functions to a detailed level. Often the software designer must do some "detailed design" during Preliminary Design to determine what effect a high level decision will have on lower levels of the design. As Detailed Design progresses it may become apparent that a design decision made during Preliminary Design needs

IV - 2
to be changed to correct, simplify, or improve the software design. Thus an iteration through "preliminary design" is necessary to reflect a modification made to the design during Detailed Design.

The refinement process in Detailed Design proceeds until the software system is ready for implementation. As Detailed Design proceeds, parts of a software system will be ready for implementation before the entire system is ready. To speed the development process, implementation of parts of the software system will often begin while the rest of the system is still in the Detailed Design phase. Parts of a software system may be concurrently in the Preliminary Design, Detailed Design, Implementation and Testing. Iteration through the Preliminary Design phase can also be caused by changes to the requirements.

**Documentation.** The iterative nature of software development has a direct impact on documentation of the software. The documentation is always changing to reflect the changes in design. The constant changes to documentation affects the choice of the method of documentation. The method of documentation must be able to support constant changes without slowing the design effort. Structure Charts with a companion Data Dictionary were chosen to document the Preliminary Design phase of the MICROSDW. The reasons for this choice were stated in the Preliminary Design chapter. The same method of documentation is also chosen for the Detailed Design phase. This choice was made for several
reasons (including those listed in Preliminary Design): The iterative changes to the Structure Charts and Data Dictionary can be made to a single set of information (versus two if another method of documentation is used for Detailed Design); to prevent loss of information in the transition from Preliminary Design to Detailed Design; detailed information and structured English or Program Design Language will be added to the Data Dictionary to reflect Detailed Design information; the resulting documentation can be passed to the implementation phase without loss of information; and follow-on maintenance of one document will be easier than maintaining two documents.

The following section describes the selection of software development tools for inclusion in the MICROSDW environment.

Selection of Software Development Tools for MICROSDW

The selection of software tools for the MICROSDW is a very important activity in creating the MICROSDW environment. The tools selected for inclusion in the MICROSDW impact the environment and its users. The tools affect the environment because by being included in the environment they alter the environment. The tools impact the users because they use the tools, and the tools either increase their productivity or decrease it. If the tool increases their productivity and is easy to use, it is an effective member of the environment. If the tool is not easy to use or decreases productivity, it
will not be used often and should be, deleted, replaced, or improved. The initial selection of tools for the MICROSDW is a starting point for an evolutionary MICROSDW environment. The number of software development tools available for microcomputers is small but steadily increasing as new tools are developed and existing tools are rehosted from larger computers. When new tools are discovered they should be evaluated for inclusion in the MICROSDW.

The following paragraphs discuss the criteria for selecting software tools for the environment and the selections made for Life-cycle and Utility tools.

Selection Criteria. The criteria for selecting a software tool for inclusion in the MICROSDW are divided into two categories: first, those criteria established by the MICROSDW requirements; and second, those imposed by the AFIT environment.

Requirements Criteria. The requirements established the goals for the MICROSDW environment, the characteristics desired in the MICROSDW environment, the characteristics desired in software tools included in the environment, and the minimum functional requirements for the environment. A tool should not be eliminated from consideration because it does not meet all criteria defined in the requirements. Indeed the MICROSDW is intended to be an environment composed of tools which each meet a portion of the requirements. Although the requirements are laid out in a "top-down" hierarchy, tools should be compared with the
requirements in a bottom-up manner. First, does the tool supply a functional requirement of the MICROSDW, if not, should the requirements be modified to include that function? If a tool meets the functional requirements, it should be subjected to the "higher-level" requirements such as, "Is it consistent with other tools in the environment?" and "Is it user friendly?" Some of the requirements, such as user friendliness require subjective decisions to be made by the person selecting the tool. So, the first set of criteria to be applied to a software tool are the requirements for the MICROSDW environment. The next set of criteria to be applied, for this thesis investigation at least, are those imposed by the AFIT environment.

**AFIT Environment Criteria.** The Air Force Institute of Technology is part of the Federal government and as an academic institution does not have large sums of money available for purchasing software development tools. Thus, all software development tools must be available to AFIT for little or no cost. Under this criteria potential software tools may be either public domain software, Air Force proprietary software, available to AFIT through an academic loan arrangement, or be available for a small cost (under $1,000) (15:134). The limited manpower devoted to the MICROSDW project and the time period in which the current research effort must be completed impose a time criteria on tool selection and installation. The software must be easily installable in a short period of time. Limited installation
time implies that the software tool must be highly portable, or previously installed on a similar microcomputer and operating system.

The following paragraphs describe the tools selected to support specific phases or functions of the software life-cycle and the tools selected to support utility or miscellaneous functions associated with software development.

**Life-cycle Tool Selection.** Life-cycle tools are tools which directly support one or more of the software life-cycle phases. Very few tools were found to support life-cycle phases. Most of the tools which were identified were eliminated from consideration because they could not satisfy the AFIT environment criteria of cost or portability. The tools which were identified for possible inclusion are discussed next.

**Requirements Definition.** No tools were identified which directly support the Requirements Definition phase of the software life-cycle. Wordprocessing Utility programs indirectly support Requirements Definition as they are used to document the Requirements. Wordprocessors can also aid in indexing and cross-referencing the Requirements.

**Preliminary Design.** Two commercial tools which support the Preliminary Design phase were identified, the Excelerator and the DEZIGN system. The Excelerator supports on-screen development of graphical representations of software designs such as Data Flow Diagrams, Structure Charts, and Presentation Graphs. The Excelerator also
supports generation of Data Dictionary information for entities stored in the former graphs. This system is specifically designed for the IBM-PC/XT. The system costs $9,500 including special hardware for the IBM-PC/XT and software designed for the hardware. The Excelerator was not included in the MICROSDW because of its cost and dependence on the IBM-PC/XT hardware.

The DEZIGN system implements the Jackson design methodology (9). The DEZIGN system supports on-screen development of Data and Program Structure Diagrams including notation from the Jackson methodology. The DEZIGN system also supports Detailed Design as source code for several different languages. Pascal, C, DBase-II, Ada, and PL/1 can be generated from software design information. The DEZIGN system is hosted on Zenith Z89, Z90, and Z-100 microcomputers running CP/M, the Z-100 running ZDOS, and the Z-100 running CP/M-86. The DEZIGN system is a low cost system, the basic system (not including language generators) costs $125, including the language generators it costs $150. The DEZIGN system is also available at AFIT hosted on the above microcomputers. The DEZIGN system is selected for inclusion in MICROSDW environments hosted on the above microcomputers.

Two public domain Structure Charting tools were also considered for inclusion in the MICROSDW, neither system met the AFIT criteria for portability or short installation time. They were both developed for specific microcomputers and printers, making them difficult and time consuming to rehost.
**Detailed Design.** As discussed above the DEZIGN system also supports Detailed Design. The user can associate pseudo-code or actual implementation language statements with modules in the software structure. Implementation language statements can be generated from the software structure directly if the appropriate language generator is available.

The SUPPORT system (46:1) was also considered to support the Detailed Design requirements of the MICROSDW. The SUPPORT system was developed at the University of Maryland and the National Bureau of Standards. The SUPPORT system is a syntax directed editor written in Pascal for a VAX 11/780 computer, rehosted to an IBM-PC. The SUPPORT system interpretes Program Design Language (PDL) statements and a subset of Pascal. PDL can be to be used to define Detailed Design algorithms for later refinement into Pascal statements. "PDL statements have control structure (IF statements, FOR loops, etc.), but whose internal contents are unstructured English text (46:2)." The SUPPORT system was not selected for inclusion in the MICROSDW because the University of Maryland would not send AFIT a copy of it to rehost.

**Coding.** A plethora of programming language compilers, debuggers, assemblers, and linkers exist for microcomputers. The goal of the MICROSDW is to eventually support more than one implementation language. A programming language must initially be selected for inclusion in the MICROSDW so the MICROSDW User Interface and Installation
program can be implemented in it. In addition to the Requirements and AFIT criteria the following items were considered in selecting an initial implementation language for the MICROSDW:

- Available on multiple microcomputers and microcomputer operating systems.
- Compiler/Linker speed.
- Executable Code speed and size.
- Numeric Accuracy
- Memory management support.
- Ease of use.

It is important that the language selected for implementing the MICROSDW be available for many microcomputers and operating systems so that the MICROSDW and programs developed in the MICROSDW environment will be portable to other systems.

The speed of the compiler is an important consideration in selecting a programming language. A compiler or linker which runs slowly can severely hamper software development, a fast compiler can speed software development. For instance, using similar compiler options, Pascal/MT+ takes over 23 minutes to separately compile about 2800 lines of code (including comments), while TURBO-Pascal only takes 2 minutes and 30 seconds to compile and link the same source files (with minor modifications for compatibility with TURBO-Pascal).

The speed and size of executable programs created by the language compiler/linker are very important in the microcomputer environment. The limited memory available on
most microcomputers today, 48 to 256 kilobytes, makes it essential that the compiler/linker produce executable code which is compact. The more compact the code the larger a program can be without resorting to memory management techniques such as overlays or chaining. The program used as an illustration above uses 38 kilobytes of memory implemented in Pascal/MT+ and only 22 kilobytes in TURBO-Pascal, a 42% savings.

Numeric accuracy is an important factor to consider in the AFIT environment because of the numeric nature of the engineering and programming problems solved using computers. In general the greater the number of digits of accuracy in floating point numbers provided by the language the better. There is a trade-off between processing speed, memory, and numeric accuracy. Greater numeric accuracy requires larger memory representation of numbers. Larger memory representations of numbers require more processing time for software algorithms to perform calculations. If the language supports hardware floating point calculations and the host computer has the floating point hardware, the processing time for numeric calculations, within the limits of the hardware, is reduced significantly compared to software processing.

Memory management support is important due to the large size of many applications programs developed at AFIT on microcomputers. Memory management techniques include: overlays, chaining, and execution of external programs. Overlays are loaded from secondary storage, usually disk,
into an overlay area in the program when a routine in the overlay is referenced by the program. An overlay area is usually shared by more than one overlay so that one area of memory can be used for many different routines. Chaining loads a new program consisting only of program code with no language library routines, from disk replacing the previous program which was executing. Execution of an external program loads a completely new program into memory without preserving any portion of the previous program. Memory management techniques slow program execution but extend the size of programs which may execute in a limited memory environment. At a minimum the language must support overlays.

Ease-of-use is a subjective measure applied to software by users. To one user a program may be "easy" to use, to another user the same program may be "hard" to use. An ease-of-use measure which can be applied to compilers involves the reporting of errors detected in a program by the compiler. Some compilers state the nature of the error in text and pinpoint the location of the error in the program. Other compilers simply print out an error code and an approximate location of the error. The latter are not easy to use because they require the user to look up an explanation of the error code and guess about where the error occurred.

Initially two versions of Pascal were considered for selection as the initial implementation language for the
MICROSDW, Pascal/MT+, and TURBO-Pascal. Pascal/MT+ was considered because the CONTROL-CAD system software was initially implemented in it (30:III-11). Pascal/MT+ was selected for implementation of CONTROL-CAD over 38 other potential language implementations available for microcomputers in early 1983. TURBO-Pascal was not available for consideration at that time so it was compared to Pascal/MT+ for possible inclusion in the MICROSDW. The reasons for choosing TURBO-Pascal over Pascal/MT+ are illustrated in Table IV-1.

**Table IV-1**

Comparison of Pascal/MT+ and TURBO-Pascal

<table>
<thead>
<tr>
<th></th>
<th>TURBO-Pascal</th>
<th>Pascal/MT+</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operating Systems</strong></td>
<td>CP/M, CP/M-86, MS-DOS, PC-DOS</td>
<td>CP/M</td>
</tr>
<tr>
<td><strong>Compiler/Linker Speed</strong></td>
<td>2:30 for 2800 lines</td>
<td>23:00 for 2800 lines</td>
</tr>
<tr>
<td><strong>Executable Code Size</strong></td>
<td>22K (for above)</td>
<td>38K</td>
</tr>
<tr>
<td><strong>Executable Code Speed</strong></td>
<td>Not Tested</td>
<td>Not Tested</td>
</tr>
<tr>
<td><strong>Floating Point</strong></td>
<td>11 digits</td>
<td>6.5 digits</td>
</tr>
<tr>
<td><strong>Numeric Accuracy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory Management</strong></td>
<td>Overlays, Chaining, Program Execution</td>
<td>Overlays</td>
</tr>
</tbody>
</table>

IV - 13
Utility Tool Selection. Utility tools are tools which can improve the productivity of a software developer but don't support a specific phase of software development. For example, a word processor increases the ability of the developer to create and maintain software documentation and to produce the numerous reports and briefings required during software development and maintenance.

Wordprocessing. There is a plethora of word processing programs available for microcomputers. A specific wordprocessor was not selected for the MICROSDW environment. It is assumed that host microcomputer systems will already have a wordprocessor familiar to the user.

Communications. There are probably more public domain communications programs available for microcomputers than there are wordprocessors. Two public domain communications programs are selected for hosting on the MICROSDW, Modem7, and Kermit (26). Modem7 was selected because of its prevalance on home microcomputer systems. Kermit was selected because it is hosted on microcomputers as well as minicomputers and mainframes. These two communications programs should allow a MICROSDW user to transfer information to and from just about any computer system.

Database Management. The MICROSDW requires a database management system to support Data Dictionary functions. A database management system was not selected for inclusion in the MICROSDW. Information about four commercial
database management systems for microcomputers was reviewed (Dbase-II, R:BASE 4000, MDBS-III, and Condor-I). The MDBS-III system was chosen as the most suitable for the MICROSDW environment. The primary reasons it was chosen as the most suitable are: 1) it is implemented on several microcomputer and minicomputer operating systems (CP/M, CP/M-86, MS-DOS, PC-DOS, TRSDOS, AppleDOS, Unix, and XENIX); 2) it has interfaces for six different programming languages (BASIC, FORTRAN, Pascal, COBOL, PL/1, and C). These facts are very important to the portability of the MICROSDW and information stored in MICROSDW databases. The limiting factor to MDBS-III is its cost, $2000+. The other database management systems were not chosen because they are implemented on a limited number of operating systems, not including minicomputer operating systems such as Unix. The other database management systems also do not have interfaces to programming languages, hindering development of application programs to interface with the databases.

**Prefabricated Software Library.** The initial prefabricated software library exists as a collection of source files for different functions and documentation. Selection of software included in the library is arbitrary. The goal of the Software Library is to provide the programmer with reusable routines which perform functions commonly used in programs. The initial library includes the routines described in *Software Tools in Pascal* by Kernigan & Plauger (25). These modules provide many commonly used functions and
programs useful to the software developer, preventing him from having to "reinvent the wheel."

This completes the section on selection of software tools for hosting on the MICROSDW. The following section discusses the detailed design of the Installation program BUILDDAT.

Install Program (BUILDDAT) Detailed Design

This section discusses the detailed design of the enhancements to BUILDDAT. The major enhancement to BUILDDAT is the addition of routines to process a text menu definition and produce the word list and shared decoding path data structures used by the User Interface. Minor additions to BUILDDAT were required to support definition of function key character sequences (keyboard definition). The Menu Definition language is described in detail in Appendix E. BUILDDAT has a simple organization with one procedure to process each of the installation text files. The installation procedure for the MENU.TXT file has an extensive set of subordinate procedures for analyzing the menu text file and creating the keyword list and decoding paths. The following table indicates the names of the procedures and text files processed by them. Figure IV-1 illustrates the high level structure of BUILDDAT.
How to represent the information processed by a program is a key issue in designing the program. This is true with the menu definition problem as well. Some methods of representing information simplify the processing required to manipulate the information. Other representations of information complicate the manipulation of the information and can obscure the actual functions and transformations being performed. As explained in Preliminary Design the shared path decoding structure used by the User Interface was retained because of its space and processing time characteristics. The shared decoding paths are a static data structure which is not changed by the User Interface. This influenced the general input, process, output structure for menu analysis. The text menu definition is read during input, analyzed and stored in an intermediate dynamic structure during processing, and converted to the static shared path structure during output. Figure IV-2 is an example of a small menu definition file.
Figure IV-2: Menu Definition File Example
Static Data Structures. The static shared path decoding structure does not lend itself to menu analysis although it is efficient for displaying menus and validating user inputs. The shared decoding paths are stored in an array of records each containing 3 integer numbers (pointers). The first number is the number of the keyword represented by that record in the menu structure. The second number is a pointer to the first record in the next lower level of the keyword hierarchy (match pointer). The third number is pointer to the next keyword in the current menu (no match pointer). Figure IV-3 shows the decoding paths matching the menu definition in Figure IV-2.

<table>
<thead>
<tr>
<th>rec#</th>
<th>word</th>
<th>word#</th>
<th>matchp</th>
<th>nomatchp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup</td>
<td>14</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>LifeCycle</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Help</td>
<td>4</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Exit</td>
<td>3</td>
<td>22</td>
<td>9999</td>
</tr>
<tr>
<td>5</td>
<td>Require</td>
<td>12</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Structure</td>
<td>15</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>PDL</td>
<td>7</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Coding</td>
<td>1</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>Testing</td>
<td>16</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>Planning</td>
<td>8</td>
<td>18</td>
<td>9999</td>
</tr>
<tr>
<td>11</td>
<td>Pre-Fab</td>
<td>9</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>ProjA</td>
<td>10</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>ProjB</td>
<td>11</td>
<td>17</td>
<td>9999</td>
</tr>
<tr>
<td>14</td>
<td>MICROSDW</td>
<td>6</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>LifeCycle</td>
<td>5</td>
<td>16</td>
<td>9999</td>
</tr>
<tr>
<td>16</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>4</td>
<td>9999</td>
</tr>
<tr>
<td>17</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>2</td>
<td>9999</td>
</tr>
<tr>
<td>18</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>8</td>
<td>9999</td>
</tr>
<tr>
<td>19</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>16</td>
<td>9999</td>
</tr>
<tr>
<td>20</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>21</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>12</td>
<td>9999</td>
</tr>
<tr>
<td>22</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>13</td>
<td>9999</td>
</tr>
<tr>
<td>23</td>
<td>$$$$$$$$</td>
<td>0</td>
<td>14</td>
<td>9999</td>
</tr>
</tbody>
</table>

Figure IV-3: Decoding Paths Matching Figure IV-2
The array of records for the decoding paths actually represent the "tree-like" hierarchy of the menu structure although that is not immediately clear from Figure IV-3. The menu hierarchy is not a true tree structure because some nodes are subtrees of more than one parent node due to the sharing of common submenus between higher level menu options. In a true tree structure each node has only one immediate parent node or else it is the root node which does not have a parent. The keywords for the static decoding paths are stored in a linear array. The keywords were not in any particular order in the original version of BUILDDAT. The keyword array produced by the new version of BUILDDAT is a sorted list of keywords in ascending order. Figure IV-4 shows the keyword list including abbreviation lengths, matching the menu definition in Figure IV-2.

<table>
<thead>
<tr>
<th>word#</th>
<th>keyword</th>
<th>abbrev</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$$$$$$$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Coding</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Design</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Exit</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Help</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>LifeCycle</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>MICROSDW</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>PDL</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Planning</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>Pre-Fab</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>ProjA</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>ProjB</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>Require</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>STOP</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>Setup</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>Structure</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>Testing</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure IV-4: Keyword List Matching Figure IV-2
**Dynamic Data Structures.** Three intermediate data structures are chosen to represent the menu structure prior to its transformation into the static arrays of decoding records, and unique keywords used by the User Interface. A binary tree data structure is selected to store the keywords. A dynamic "semi-tree" structure is used to represent the menu structure. A linked list structure is selected to store the list of routine names to call for the various menu options.

**Keyword Structure.** A height balanced binary tree structure was chosen to store the unique keywords because of its search and insertion time characteristics. A height balanced binary tree has an average and worst case insertion time of Order(log n), where n is the number of nodes in the tree (19:440,454). An unbalanced binary tree has a worst case insertion time of Order(n). An ordered listing of the keywords from a binary tree is produced in Order(n) time. Besides the excellent timing characteristics of height-balanced binary trees, algorithms implementing them are available for use with the MICROSDW. Figure IV-5 illustrates the Keyword Tree matching the Menu Definition in Figure IV-2.
Menu Structure. A dynamic "semi-tree" structure patterned after the structure of the decoding path records is chosen to represent the menu structure because it reflects the hierarchical nature of the menu structure. The dynamic tree structure is also easily transformed to the static decoding path structure. A complete menu path specifies some action or function to be carried out. The last keyword in a menu path points to a termination record. When the input validation routine in the User Interface encounters a termination record it knows that the menu path is complete. Termination records are identified by a unique word "$\text{$$$$$$$$} \text{". The termination record contains the number of the name of the routine to call in the field which would
normally point to the next lower level of the menu structure. Each leaf node in the dynamic menu structure contains a pointer to a "call routine." To conserve space in the static menu structure there is one termination record for each "call routine." For instance, there may be many options in the "HELP" menu but the same termination record is used for each. Figure IV-6 and IV-7 illustrate the menu structure matching the Menu Definition in Figure IV-2.

```
Setup
LifeCycle
  Require
  Structure
    Pre-Fab
    ProjA
    ProjB
  PDL
    Pre-Fab
    ProjA
    ProjB
  Coding
  Testing
  Planning
Help
MICROSDW
LifeCycle
Exit
```

Figure IV-6: Menu Structure Matching Figure IV-2
Note: The Keywords in the nodes above are for clarity. The actual MENU_TREE and CALL_LIST nodes contain pointers to nodes in the Keyword structure containing the appropriate keywords.

Figure IV-7: Menu Decoding Pointer Structure For Figure IV-2
Call Routine List. The unique "call routine" names specified in the menu definition are represented in a linked list. The actual "call routine" names are stored in the dynamic keyword tree. Each node in the "call routine" list contains a pointer to the appropriate node in the keyword tree. Figure IV-7 also illustrates the Call Routine List structure matching the Menu Definition in Figure IV-2.

Static Structure Creation. After the dynamic menu structures have been created they must be transformed into the static data structures used by the User Interface. The transformation process is broken into two sections. The first section creates the keyword list from the keyword tree. The second section creates the decoding path records from the dynamic menu structure and the "call routine" list.

Keyword List Creation. Transformation of the keyword tree into the keyword list is accomplished by traversing the tree using a "Left-Node-Right" (19:229) recursive algorithm. Left-Node-Right may also be referred to as a "Left-Data-Right" or "inorder" algorithm. Left-Node-Right indicates that all nodes in the left subtree are processed, the current node is processed, and then all nodes in the right subtree are processed. The Left-Node-Right traversal of the keyword tree produces a list of the unique keywords in ascending order. When the current node is processed the keyword stored in the tree node is copied to the word list and a number is assigned to the current tree node. The number indicates the location of the
keyword in the keyword list. This number is used during creation of the decoding paths to reference the appropriate keyword. After the sorted list of keywords has been created the minimum number of characters necessary to uniquely identify each keyword is calculated. This number is used by the User Interface to match input keyword abbreviations to the valid keywords in each menu. After the keyword tree has been processed, the menu structure is transformed to the decoding paths.

Decoding Path Creation. Creation of the decoding paths is a three step process. First, the nodes in the call routine list are assigned decoding path record numbers. The nodes in the menu structure were assigned decoding path record numbers as they were created during processing the menu text file. The call routine decoding path record numbers are referenced when the leaf nodes of the menu structure are transformed into decoding path records. Next, the decoding path records for the call routines are created.

The third step in creating the static data structures is the transformation of the menu structure into decoding path records. The dynamic menu is processed using a "Node-Left-Right" recursive algorithm. Node-Left-Right may also be referred to as a "Data-Left-Right" or "preorder" algorithm. First the information in the root node is transformed into a decoding record, then the subtree (Left) containing the remaining options in the current menu is processed. After all of the options in the current menu have
been processed, the subtrees (Right) containing the options for each of the submenus are processed. This continues until all nodes in the menu structure have been processed. To prevent reprocessing of shared submenus before a menu node is processed its decoding record is checked to see if it contains valid information, if it does, the node has already been processed and its processing is terminated.

Summary

This chapter describes the initial components selected for inclusion in the MICROSDW environment and the detailed design for the MICROSDW Install program, BUILDDAT. The components initially selected for the MICROSDW are a starting point for the growth of a complete microcomputer based software development environment. A lack of software development tools for microcomputers delayed total development of the MICROSDW. Several key tools which were not selected were those supporting Requirements Definition, Data Dictionary, and Database Management. The algorithms developed during Detailed Design are the basis for coding accomplished during the following Implementation phase.
V. IMPLEMENTATION

Introduction

This chapter discusses the decisions made during the Implementation phase of the MICROSDW software development. The chapter is composed of six sections; Introduction, Implementation Language, Host Microcomputer/Operating System, Implementation Standards, Implementation Strategy and Summary. The section on implementation language discusses briefly the language selected for implementing the MICROSDW. The Host Microcomputer/Operating System section discusses the initial microcomputer and operating system selected for hosting the MICROSDW. The Implementation Standards section describes the programming conventions and standards used during implementation of the MICROSDW. The section on Implementation Strategy discusses the order for implementing the MICROSDW and the results of implementing and testing the software. The summary section summarizes what parts of the MICROSDW were implemented, problems discovered, and work remaining to be accomplished. The remainder of the Introduction discusses the purpose for implementation and the relationship of the Implementation phase to the other software life-cycle phases.

The purpose of Implementation is to translate the Detailed Design into the language selected for implementing the software. The Detailed Design documentation, Structure Charts and Data Dictionary, are input to the Implementation
phase. The output from the Implementation phase is an executable solution to the Detailed Design model and the source code implementing the solution. Thus, the documentation for the Implementation phase is the source code produced. Implementation is an iterative process like the life-cycle phases preceding it. As the Detailed Design is translated into the host language source code, idiosyncrasies of the host language and hardware environment are discovered which cause changes to the implementation. Also deficiencies in the software design may be discovered which require the Detailed Design to be modified. The executable solution (programs) output from Implementation is input to the Integration, Testing, Operation, and Maintenance phases of the software life-cycle. The programs output from Implementation must be integrated into the host environment and tested for errors. After the programs have been integrated and tested in the host environment they are ready for Operation and Maintenance. The following section discusses selection of the implementation language.

**Implementation Language**

Selection of a programming language for hosting the initial version of the MICROSDW environment was discussed at length in the Detailed Design section on Selection of Software Tools. The initial programming language selected was TURBO-Pascal. TURBO-Pascal has many features which contributed to its selection. The most important feature for
the MICROSDW is its transportability between common microcomputer operating systems (CP/M, CP/M-86, AND MS-DOS) and the ability to configure it for different host microcomputers. Other features which contributed significantly to its selection are its interactive compile and runtime debugging features, the compactness of the executable code it produces, and the speed with which it compiles programs. The transportability of TURBO-Pascal programs between operating systems and hardware systems is verified during Integration and Testing by transferring the MICROSDW to different operating systems and hardware.

**Host Microcomputer/Operating System**

The DEC Rainbow-100 was selected as the initial host microcomputer. The Rainbow-100 was chosen because it supports two microprocessors and three operating systems. The two microprocessors it uses are the Zilog Z80A and the Intel 8088. The Z80A microprocessor supports 8 bit data manipulation and a 64 kilobyte memory address space using an 8 bit data bus. The 8088 microprocessor supports 16 bit data manipulation and a 1 Megabyte memory address space using an 8 bit data bus.

The DEC Rainbow-100 also meets the Hardware Requirements described in Chapter 2. Table V-1 lists the Rainbow-100 hardware features which satisfy the Hardware Requirements of the MICROSDW.
Table V-1
DEC Rainbow-100 Hardware Features

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>Z80A, 8088</td>
</tr>
<tr>
<td>Memory</td>
<td>256 Kilobytes</td>
</tr>
<tr>
<td>Disk Storage</td>
<td>2 Floppy Disk Drives, 386K Storage Each</td>
</tr>
<tr>
<td>Display</td>
<td>24 lines by 80 or 132 Columns, Cursor Positioning, Reverse Video, Bold, etc.</td>
</tr>
<tr>
<td>Printer</td>
<td>80 Column Dot Matrix, 5, 6, 10, 12, 16 Characters Per Inch, Bold, Underline, Graphics, Multiple Character Sets, etc.</td>
</tr>
<tr>
<td>Keyboard</td>
<td>105 Keys total, 18 Key Numeric Keypad, 36 Function Keys (including Cursor Positioning)</td>
</tr>
<tr>
<td>Ports</td>
<td>1 Serial RS232 Printer Port</td>
</tr>
<tr>
<td></td>
<td>1 Serial Communications Port</td>
</tr>
<tr>
<td>Modem</td>
<td>300/1200 Baud (separate from Rainbow 100)</td>
</tr>
</tbody>
</table>

The dual microprocessors in the Rainbow also allow it to support multiple operating systems. The Rainbow supports a hybrid CP/M-80/86 operating system which combines the capabilities of CP/M-80 and CP/M-86. The CP/M-80/86 operating system allows the user to execute, create, or modify programs for either CP/M-80 or CP/M-86. The Rainbow also supports MS-DOS. The Rainbow's support of multiple operating systems strongly influenced its selection as the initial MICROSDW host microcomputer. The MICROSDW can be implemented and tested on three operating systems with only one microcomputer. This significantly reduces the time
required to test the MICROSDW in different operating system environments.

Implementation Standards

Implementation Standards refers to the programming conventions followed during coding and maintenance of MICROSDW programs. Two programming conventions are adopted for the MICROSDW. The first is standard information included in each MICROSDW source code file and procedure. The second convention is the programming style guidelines followed when creating or modifying source code.

File headers are included in each source file to store general information about the file such as how it is used and what procedures and function it contains. The template used for file headers is illustrated in Figure V-1. Procedure headers immediately precede the code for each procedure in a source file. The template used for procedure headers is illustrated in Figure V-2.
Programming style guidelines (convention/standards) can improve the readability and maintainability of computer programs. In the book The Elements of Programming Style, Brian Kernighan and P.J. Plauger describe and illustrate many programming style guidelines (24). Some elements of programming style actually begin during the earlier design phases of software development such as modularization of procedures and localization of input and output. These elements should be continued during Implementation along with
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

UNCLASSIFIED P A MOORE DEC 84 AFIT/GCS/ENG/84D-18 F/G 9/2 NL
the other guidelines. Following is a list of some general programming style guidelines followed in implementing MICROSDW software (24:159-160):

- Write clearly, not sacrificing clarity for "efficiency"
- Use Library Functions whenever possible
- Indent to show the logical structure of a program
- Choose data representations which make the program simple
- Make coupling between modules visible

The next section discusses the strategy used for implementing the MICROSDW.

**Implementation Strategy**

This section describes the implementation strategy for the MICROSDW. A strategy is a plan or method for achieving an end. The implementation strategy for the MICROSDW is a sequence of steps resulting in a prototype MICROSDW environment including a prototype User Interface program, an Installation program; and a Prefabricated software library. The following list of steps is a synopsis of the MICROSDW implementation strategy:

1. Rehost the CONTROL-CAD software on a DEC Rainbow
2. Convert CONTROL-CAD software to TURBO-Pascal
3. Code/Test menu definition interpretation procedures
4. Code/Test the keyword abbreviation procedures
5. Code/Test User Interface Built-In functions
6. Code/Test the Cursor Control/Function Key procedures
7. Code/Test Prefabricated software library procedures and functions

The above steps, the actions required in each step, and the results of each step are described in the following paragraphs.

Step 1. **Rehost CONTROL-CAD.** The CONTROL-CAD software
reused by the MICROSDW was previously hosted on a Heath H19/H29 system. The source code for the system was first transferred to the DEC Rainbow. The text files describing the hardware were modified for the Rainbow and installed in the system file describing the hardware and menu system. The contents of the terminal definition text file were completely replaced. The character sequences for the Rainbow VT-100 type terminal were completely different than the H19/H29 character sequences. The old terminal definition file for the H19/H29 was retained to facilitate later rehosting of the MICROSDW on H19/H29 systems. Due to differences in the way the Rainbow processes cursor positioning escape sequences, the cursor positioning procedure in the User Interface had to be modified. The Rainbow terminal expected the row and column positions to be specified by ASCII characters instead of binary numbers. To enhance transportability of the MICROSDW an unused byte (90) in the terminal definition array was used as a terminal type indicator. The terminal type indicator is used to either generate binary row and column numbers for H19/29 type terminals or ASCII row and column numbers for VT100 type terminals. There were no other significant problems in rehosting the CONTROL-CAD system to the Rainbow. The next step in implementing the MICROSDW is the conversion from Digital Research Pascal/MT+ to Borland International TURBO-Pascal.

**Step 2. Convert to TURBO-Pascal.** Conversion to TURBO-Pascal was accomplished in two steps. The first step
was to convert the Install program to TURBO-Pascal. The experience gained from this conversion was a deciding factor in choosing TURBO-Pascal as the implementation language. The advantages of TURBO-Pascal had to be sufficient to justify the conversion effort, and the conversion had to be straightforward, not requiring rewriting of much source code. The advantages of TURBO-Pascal over Pascal/MT+ were documented in Chapter IV in the selection of software development tools for coding.

The conversion was straightforward, requiring very few changes to the code. The compiler directives in the code had to be modified or deleted to be compatible with TURBO-Pascal. A difference in the number of significant characters in identifiers created some compile errors in TURBO-Pascal that were missed by Pascal/MT+. These errors were easily fixed and the Install program was successfully compiled and executed.

After the Install program (BUILDDAT) was successfully converted to TURBO-Pascal and executed, the User Interface was also converted. Testing was completed after the conversion of the User Interface to TURBO-Pascal. The install files created by BUILDDAT were used to successfully initialize and execute the User Interface. This demonstrated that the conversion was feasible.

The next step in implementing the MICROSDW is coding and testing the menu definition interpretation procedures from the Detailed Design specifications.
Step 3. Code/Test Menu Definition Procedures. This step followed the Detailed Design of the menu definition procedures. Coding was accomplished in three substeps: first, coding and testing of menu definition input and parsing procedures; second, coding and testing of procedures to create the dynamic menu data structures; third, coding and testing of procedures to transform the dynamic data structures into static data structures.

The menu definition input and parsing procedures were coded and tested on a menu definition file matching the menu structure of the CONTROL-CAD system. The procedures were debugged by instrumenting the code with the statements to print out intermediate and final results of parsing input menu text.

Next, the procedures which create the dynamic menu data structures from the parsed menu definition were coded. The binary tree manipulation functions and procedures used to store the keywords were rehosted from the AFIT Unix VAX 11/780. Rehosting the binary tree procedures required minor changes to the source code. The binary tree procedures were then modified to support storage of keywords. The procedures which transform the menu definition text to the dynamic data structures were tested by instrumenting the code with debugging statements and by implementing procedures to print out the dynamic structures.

Finally, the procedures to transform the dynamic structures to static data structures were coded. The
procedures transforming the dynamic keyword tree and menu structures to static structures were tested by instrumenting the code with debugging statements and by printing the static structures (arrays) created by the procedures. When these procedures were sufficiently debugged, their output was written to the installation file MICROSDW.SYS and used to initialize the User Interface. Several minor changes were made to the User Interface procedures which validate user input keywords against the valid options. The changes were made to correct minor changes made in the static menu structures used to display and validate menu keywords. When the User Interface successfully processed the initial menu structure, more extensive menu structures were created via menu definitions and tested on the User Interface and Install programs. Minor errors were detected and corrected in the Install and User Interface programs through this testing.

Step 4. Code/Test Abbreviation Procedures. Two procedures were coded to implement global abbreviation of keywords in the User Interface. A procedure was added to the Install program to calculate keyword abbreviation lengths and a procedure (function actually) was added to the User Interface to compare input keyword abbreviations with valid keywords from the menu structure. A data structure array was also added to the static menu structure to store the abbreviation lengths for each keyword. These procedures were tested statically by validating the calculated abbreviation lengths. The procedures were tested dynamically by entering
abbreviations in response to User Interface prompts and verifying that the correct keyword was matched or that erroneous abbreviations (keywords) were detected.

While testing the abbreviations it was discovered that it was impossible to determine what the minimum abbreviation for a keyword was by looking at the menu prompts. For the abbreviations to be useful to the user they had to be indicated in some way on the display screen. This was accomplished by making the abbreviation characters appear brighter than the other characters in each keyword. This required implementation of two functions to insert "bold-off" and "bold-on" character sequences into the keywords before they were displayed on the screen. Two previously unused portions of the terminal definition array, locations 70-75 and 76-80, were designated to store the "bold-off" and "bold-on" character sequences respectively.

Displaying keyword abbreviations in bold significantly enhanced the usefulness of the abbreviations. Testing the User Interface to see if it correctly interpreted abbreviations was also simplified by this enhancement.

Step 5. Code/Test Built-in Functions. Three of the built-in functions of the MICROSDW were coded and tested; Help, Run, and Directory. The Help procedure was implemented to provide users with assistance in running the MICROSDW. The Run procedure was implemented to provide for execution of other programs from the MICROSDW. The Directory procedure provides for listing which files are on a disk. The Run and
Directory procedures were chosen for implementation because their implementation requires implementation of procedures to support other built-in functions.

The Help procedure was modified to support the "Help" options for the MICROSDW. The modifications were simple, the previous Help keywords supporting the CONTROL-CAD system were replaced with those for the MICROSDW. The HELP.TXT file was also modified, the CONTROL-CAD Help messages were replaced with those for the MICROSDW. The HELP.SYS and MICROSDW.SYS files were then regenerated using BUILDDAT so that the new messages could be accessed by the User Interface. The modified HELP procedure was tested by exercising each of the Help options using the User Interface.

The Run and Directory procedures were coded and tested independent of the User Interface. A simple driver program was used to prompt for input and call the Run and Directory procedures. Using the driver program reduced the time necessary to compile and debug the procedures. Procedures to prompt for filenames and validate the filenames were coded to support Run and Directory. These procedures are also needed to support implementation of other MICROSDW built-in functions. After these procedures were debugged using the driver program they were included in the User Interface and tested using it.

Summary

During the coding phase a prototype Microcomputer
Software Development Workbench was implemented. The Install program, BUILDDAT, was significantly enhanced by adding procedures to interpret a text menu definition and create the static menu structure used by the User Interface.

The addition of consistent keyword abbreviations and on-screen display of the abbreviations also enhanced the user friendliness of the User Interface.

Coding and Testing of procedures to implement cursor movement and function key detection (Step 6) was not completed due to time constraints.

Step 7, Coding/Testing of Prefabricated software library procedures was not completed. A public domain library of software procedures and functions from Kernighan and Plauger's *Software Tools in Pascal* (25) were obtained from Sanders Associates, Inc., Nashua, New Hampshire. The procedures in the library are implemented in TURBO-Pascal, so no conversion or modifications were necessary to host them in the MICROSDW environment.

The following Integration chapter discusses integration of the MICROSDW with the AFIT Digital Engineering Laboratory SDW, rehosting of software development tools, and porting the MICROSDW to other microcomputer operating system and hardware environments.
Introduction

The purpose of the Integration phase of the MICROSDW is to integrate the User Interface and the software development tools, demonstrate the transportability of the MICROSDW to other microcomputer operating systems and microcomputers, and to discuss integration of the MICROSDW with the AFIT Digital Engineering Lab (DEL) SDW. The Integration phase of the software life-cycle follows the Implementation phase and precedes Operation and Maintenance of the MICROSDW.

The following section discusses integration of software development tools.

Integration of Software Development Tools

Integration of software development tools in the MICROSDW is the process of ensuring that software development tools hosted on the MICROSDW interface correctly with each other and with the MICROSDW User Interface. The initial version of the MICROSDW does not include any development tools which transfer information between each other, so they do not need to be integrated with each other. The MICROSDW does include tools which interface with the MICROSDW User Interface. These tools must be integrated with the User Interface.

A software development tool is integrated with the User
Interface by creating an option for the tool in the menu structure and defining the name of the software procedure to call when that option is selected to initiate the software development tool. There are two methods of initiating a software development tool from the User Interface, either by specifying the "Run" procedure as the procedure to execute the tool or by writing a special procedure to execute the tool and specifying that procedure in the menu structure.

The "Run" procedure can be used to execute programs which do not require options to be entered on the command line. Using the "Run" procedure does not require any changes to the User Interface. Since the User Interface does not have a facility built into it to automatically prompt the user for additional command line arguments to pass to tools executed from the User Interface, tools requiring command line arguments must be executed from procedures tailored to prompt for command line arguments and pass them to the tool being executed.

Two software development tools were integrated with the User Interface: the TURBO-Pascal compiler and the Kermit communications program. Neither of these tools require command line arguments so they were integrated with the User Interface using the "Run" procedure. Integration of the tools was successfully tested by executing the User Interface and selecting the appropriate menu options to execute TURBO-Pascal and Kermit.

The next aspect of integrating the MICROSDW is
integrating the MICROSDW on other operating systems and microcomputers.

Integrating the MICROSDW in Different Environments

One of the major goals of the MICROSDW project is to create a software development environment which can be hosted on different microcomputers and microcomputer operating systems. The achievement of this goal requires that the MICROSDW User Interface and Install programs be portable and the software development tools in the MICROSDW be portable to other microcomputers and operating systems. The MICROSDW is designed to be portable from one microcomputer environment to another. The software development tools hosted on the MICROSDW are not necessarily portable. The portability of the MICROSDW User Interface and Install programs is demonstrated by hosting them on other microcomputers and microcomputer operating systems. The portability of the software development tools currently included in the MICROSDW is not demonstrated because they are not all implemented in TURBO-Pascal. The time required to modify them to execute on different operating systems or convert them to TURBO-Pascal is beyond the scope of this thesis investigation.

Portability of the MICROSDW between CP/M-80, CP/M-86, and MS-DOS on the DEC Rainbow is demonstrated by:

1) Compiling the User Interface and Install programs using the TURBO-Pascal compiler appropriate for each operating system.
2) Executing Install to build the MICROSDW.SYS and HELP.SYS files for the operating system.
3) Executing the User Interface.

VI - 3
4) Validating that the User Interface correctly displays the menus and interprets user inputs.

The portability of the MICROSDW between CP/M-80, CP/M-86, and MS-DOS on the Rainbow was successfully demonstrated by following the above steps.

Portability of the MICROSDW between microcomputers running one of the above operating systems is demonstrated by:

1) Moving the executable version of the User Interface and Install programs matching the appropriate operating system from the Rainbow to the destination microcomputer.
2) Moving the MICROSDW text files and system files to the destination microcomputer.
3) a) Modify the terminal definition file (TERMINAL.TXT) contents to be compatible with the terminal on the destination microcomputer, or
   b) If a terminal definition file exists for the terminal delete the TERMINAL.TXT file and copy the existing terminal definition file to TERMINAL.TXT.
4) Executing Install to build the MICROSDW.SYS and HELP.SYS files for the operating system.
5) Executing the User Interface.
6) Validating that the User Interface correctly displays the menus and interprets user inputs.

There is a possibility that the User Interface or Install program may not execute if the destination microcomputer has less than 64K memory and the executable program was created on a system with 64K memory or greater. If the User Interface or Install program will no execute on the destination microcomputer and operating system, the source code for the User Interface and Install programs should be transferred to the destination computer and compiled using TURBO-Pascal, to create new executable programs.
The portability of the MICROSDW User Interface and Install programs is demonstrated by the above examples, thus the goal of creating a portable User Interface for a microcomputer software development environment is met.

Integration of the MICROSDW with the APIT Digital Engineering Lab (DEL) SDW hosted on a VAX 11/780 running VMS is discussed next.

**Integration with the APIT DEL SDW**

The previous two sections discussed integration at two levels, the software development tool level and the microcomputer to microcomputer level. This section discusses integration between the mainframe SDW environment and the microcomputer SDW environment. For separate software development environments to be integrated with each other, they must first be able to communicate or transfer information back and forth. Then, the software development information which is transferred from environment to environment must have the same meaning in both environments. If the same information (data) has different meanings in each environment the result will be confusion about what the information represents.

Communication between computers requires that the computers be connected via a communications link. They could be connected by a computer network, a phone line, a microwave link, or satellite link for example. Once the computers are connected they must have a protocol, hardware, and software,
for exchanging information. The method of establishing a connection between the DEL SDW and the MICROSDW is outside the scope of this thesis investigation. It is assumed that the DEL SDW and MICROSDW can establish a connection via network, phoneline, or direct connection. The initial protocol for exchanging information between MICROSDW environments and between the AFIT DEL SDW and MICROSDW environments is provided by Kermit (Ker84) file transfer programs hosted on each of the environments. Using Kermit, MICROSDW environments can transfer information to or from other MICROSDW environments or the DEL SDW. After establishing communication with the DEL SDW, MICROSDW users can either transfer files to or from the DEL SDW using Kermit or execute DEL SDW software development tools. Thus, the first requirement for integration of the MICROSDW with the DEL SDW, communication, is met.

Integration of software development tools between the DEL SDW and the MICROSDW has not yet been achieved. The few software development tools hosted on the MICROSDW do not produce information (data) which is compatible with DEL SDW tools. Integration between the software development tools hosted on each environment is essential. If the tools are not integrated with each other the information produced in one environment cannot be used by the other. Integration between tools in different environments can be achieved by having the tools format information in the same way so that a tool in the MICROSDW can read and process information.
produced in the DEL SDW and vice-versa. Integration can also be achieved between tools which do not format information the same by creating tools which convert information from MICROSDW format to DEL SDW format and vice-versa.

Integration of MICROSDW and DEL SDW tools should be a high priority for future software development tools hosted in either environment. If the tools are not integrated the MICROSDW and DEL SDW environments will not support each other.

**Summary**

Several aspects of Integration concerning the MICROSDW and AFIT DEL SDW were addressed in this chapter. First, integration of the software development tools within the MICROSDW environment was discussed. Then, portability of the MICROSDW to different microcomputers and microcomputer operating systems was illustrated by rehosting the MICROSDW on different microcomputers and operating systems. Last, integration of the MICROSDW with the AFIT DEL SDW was discussed.
VII. OPERATION AND MAINTENANCE

Operation and Maintenance of the MICROSDW is the use and evolution (modification) of the MICROSDW environment. The Operation and Maintenance phase of the MICROSDW begins after the completion of this initial research and development effort. Provisions for Operating and Maintaining the MICROSDW have been made. A User's Manual (Appendix G) and Programmer/Maintenance Manual (Appendix H) are provided in the back of the thesis. The program listings for the User Interface program (MICROSDW), and Install program (BUILDDAT) are provided in the back of the thesis (Appendix I).
VIII. CONCLUSIONS AND RECOMMENDATIONS

Introduction

This chapter contains a summary of the MICROSDW development effort, conclusions, and recommendations for future expansion of the MICROSDW environment, and Summary.

MICROSDW Development Summary

A prototype microcomputer software development workbench (MICROSDW) environment was created. Development of the environment began with a literature review of software development environments in general, software development environments including microcomputers, software development tools, and software development tools for microcomputers. The development of the MICROSDW then followed a classic software life-cycle development sequence: Requirements Definition, Preliminary Design, Detailed Design, Implementation, Integration, and Operation and Maintenance.

During Requirements Definition the requirements for the MICROSDW were established. The requirements were consolidated from sources identified during the literature search (15; 11; 29; 39; 14).

The functional structure of the MICROSDW User Interface and Install programs was developed during Preliminary Design and is documented in Appendix C and D. The overall conceptual organization of the MICROSDW environment was also described during Preliminary Design. During Preliminary
Design a microcomputer Control System Analysis system, CONTROL-CAD, was identified for potential modification to become the MICROSDW User Interface and Installation programs.

During Detailed Design the initial software development tools were selected for hosting in the MICROSDW environment, the detailed design of the MICROSDW Install program was completed, and a menu definition language (Appendix E) was developed for interpretation by the Install program.

During Implementation modifications to the Install and User Interface programs were coded and tested.

During Integration the MICROSDW was ported from CP/M-80 on the DEC Rainbow to CP/M-86 and MS-DOS and rehosted on other microcomputers to demonstrate its portability. The Kermit communications program (26) was hosted on the AFIT DEL VAX 11/780 and the MICROSDW establishing a communications capability between the AFIT DEL SDW and the MICROSDW.

The objective of creating a prototype microcomputer software development workbench was accomplished. The lack of software development tools for microcomputers delayed the development of a more complete MICROSDW environment.

Recommendations for MICROSDW Expansion

The Current MICROSDW provides a starting point for development of a complete MICROSDW environment supporting all phases of software development. The following paragraphs outline recommended extensions of the MICROSDW to eventually provide a complete MICROSDW environment.
Software Development Tools

**Data Dictionary.** The most important tool to develop is a data dictionary tool to maintain a central repository of software development information for software development projects and the prefabricated software library. This tool is essential for storing and providing information to other MICROSDW development tools. Standard prefabricated software procedures should be developed to provide other software development tools access to information stored in central repositories of software development data. The data dictionary tool should be able to format development data for use by the VAX 11/780 SDW data dictionary tool and accept data for input data from the SDW data dictionary tool.

**Requirements Specification/Definition.** A graphical tool should be developed to support requirements specification/definition. The tool should utilize a data dictionary tool to store and update requirements data.

**Structure Charting Tool.** A structure charting tool should be developed to support on-screen graphical creation and modification of structure charts. The tool should also be integrated with a MICROSDW data dictionary tool.

**Program Design Language (PDL) Tool.** A PDL tool should be developed to support the detailed design and coding phases of the software life-cycle.

**Source Code Control Tool.** A source code control system should be developed to provide the capability...
to store and retrieve multiple versions of files. The Archive program (25:85-108) in the Prefabricated Software library could be used as a basis for a more extensive Source Code Control System.

**User Interface.** The MICROSDW User Interface capabilities should be expanded with additional built-in functions. Other improvements which could be made are (11a):

1) global definition of alternatives included in all menus,
2) definition of individual help and error messages for each menu option, and
3) support for non-keyword prompting and syntax checking.

**Install Program.** The Install program should be improved to support interactive modification of the MICROSDW installation files, and automatic maintenance of multiple copies of install files for other microcomputers. The Install Program would be affected by some of the proposed changes to the User Interface.

**Prefabricated Software Library.** Development of a more extensive prefabricated software library has the potential to save many man hours spent reinventing the same procedures and functions over and over again for every new software development project. Libraries of procedures providing access to microcomputer operating system functions for CP/M-80, CP/M-86, and MS-DOS should be developed first. Use of these libraries could significantly increase the portability of applications developed on one microcomputer operating system to the others. Additional possibilities for
the prefabricated software library include graphics procedures and data structure manipulation procedures for stacks, queues, lists, trees, etc.

**Other Applications for the MICROSDW User Interface.** The User Interface and Install programs for the MICROSDW can be used as the basis for just about any microcomputer application that lends itself to a hierarchical menu structure for prompting the user. To configure the User Interface for other applications the software developer must define a new menu structure, install the menu structure, modify and install the Help text, and replace the procedures called by the User Interface "Select_Routine" procedure with those for the new application. The reusable nature of the User Interface has the potential to significantly reduce development time for other applications.

**Summary**

A prototype software development workbench for microcomputers, the MICROSDW, was created for use by AFIT students and faculty. The prototype MICROSDW was ported to three microcomputer operating systems: CP/M-80, CP/M-86, and MS-DOS. The MICROSDW provides a starting point for developing a fully integrated, transportable software development environment for microcomputers.
APPENDIX A

REQUIREMENTS DEFINITION SADTS
APPENDIX A

REQUIREMENTS DEFINITION SADTS

Introduction
The requirements for transportable User Interface and Install programs for the Microcomputer Software Development Workbench are illustrated by the following Structured Analysis and Design Technique (SADT) activity diagrams and text. SADT activity diagrams show: control, input, and output data; mechanisms used to accomplish activities; and activities. Input data is transformed by an activity into output data. Control data is primarily used by activities to make decisions concerning processing of input data. Control data may also be used as input data by activities. Figure C-1 illustrates a basic activity box with its control data, input data, output data, and mechanism. See (36:36-54) for further details about SADTs. The following SADT diagrams were created using the AFTI Digital Engineering Laboratory (DEL) Software Development Workbench (SDW) AUTOIDEF Requirements Definition tool.

Figure C-1: SADT Activity Diagram
<table>
<thead>
<tr>
<th>Node</th>
<th>Title</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-0</td>
<td>PROVIDE MICROSDW</td>
<td>1</td>
</tr>
<tr>
<td>M0</td>
<td>PROVIDE MICROSDW</td>
<td>2</td>
</tr>
<tr>
<td>M1</td>
<td>Perform Hardware Configuration</td>
<td>3</td>
</tr>
<tr>
<td>M11</td>
<td>Install Terminal Characteristics</td>
<td>4</td>
</tr>
<tr>
<td>M111</td>
<td>Determine Selection Type</td>
<td></td>
</tr>
<tr>
<td>M112</td>
<td>Create Terminal Definition</td>
<td></td>
</tr>
<tr>
<td>M113</td>
<td>Select Predefined Terminal</td>
<td></td>
</tr>
<tr>
<td>M114</td>
<td>Validate Terminal Characteristics</td>
<td></td>
</tr>
<tr>
<td>M12</td>
<td>Install Printer Characteristics</td>
<td>5</td>
</tr>
<tr>
<td>M121</td>
<td>Determine Selection Type</td>
<td></td>
</tr>
<tr>
<td>M122</td>
<td>Create Printer Definition</td>
<td></td>
</tr>
<tr>
<td>M123</td>
<td>Select Predefined Printer</td>
<td></td>
</tr>
<tr>
<td>M124</td>
<td>Validate Printer Characteristics</td>
<td></td>
</tr>
<tr>
<td>M13</td>
<td>Install Keyboard Characteristics</td>
<td>6</td>
</tr>
<tr>
<td>M131</td>
<td>Determine Selection Type</td>
<td></td>
</tr>
<tr>
<td>M132</td>
<td>Create Keyboard Definition</td>
<td></td>
</tr>
<tr>
<td>M133</td>
<td>Select Predefined Keyboard</td>
<td></td>
</tr>
<tr>
<td>M134</td>
<td>Validate Keyboard Characteristics</td>
<td></td>
</tr>
<tr>
<td>M14</td>
<td>Install Communications Characteristics</td>
<td>7</td>
</tr>
<tr>
<td>M141</td>
<td>Determine Selection Type</td>
<td></td>
</tr>
<tr>
<td>M142</td>
<td>Create Communications Definition</td>
<td></td>
</tr>
<tr>
<td>M143</td>
<td>Select Predefined Communications</td>
<td></td>
</tr>
<tr>
<td>M144</td>
<td>Validate Communications Characteristics</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>Perform Software Configuration</td>
<td>8</td>
</tr>
<tr>
<td>M21</td>
<td>Install Software Configuration</td>
<td>9</td>
</tr>
<tr>
<td>M211</td>
<td>Modify Software Config. Text File</td>
<td></td>
</tr>
<tr>
<td>M212</td>
<td>Validate Software Config. Text File</td>
<td></td>
</tr>
<tr>
<td>M213</td>
<td>Build Software Config. Description</td>
<td></td>
</tr>
<tr>
<td>M22</td>
<td>Install Help Messages</td>
<td>10</td>
</tr>
<tr>
<td>M221</td>
<td>Modify Help Text File</td>
<td></td>
</tr>
<tr>
<td>M222</td>
<td>Validate Help Text File</td>
<td></td>
</tr>
<tr>
<td>M223</td>
<td>Build Help Description</td>
<td></td>
</tr>
<tr>
<td>M23</td>
<td>Install Error Messages</td>
<td>11</td>
</tr>
<tr>
<td>M231</td>
<td>Modify Error Text File</td>
<td></td>
</tr>
<tr>
<td>M232</td>
<td>Validate Error Text File</td>
<td></td>
</tr>
<tr>
<td>M233</td>
<td>Build Error Description</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Title</td>
<td>Diagram #</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>M3</td>
<td>Provide User Interface</td>
<td>12</td>
</tr>
<tr>
<td>M31</td>
<td>Initialize User Interface</td>
<td>13</td>
</tr>
<tr>
<td>M311</td>
<td>Initialize Hardware Descriptions</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>M3111 Read Terminal Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M3112 Read Printer Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M3113 Read Keyboard Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M3114 Read Communications Description</td>
<td></td>
</tr>
<tr>
<td>M312</td>
<td>Initialize Software Description</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>M3121 Read Software Config. Desc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M3122 Read Help Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M3123 Read Error Description</td>
<td></td>
</tr>
<tr>
<td>M32</td>
<td>Provide Menu</td>
<td></td>
</tr>
<tr>
<td>M33</td>
<td>Read User Selection</td>
<td></td>
</tr>
<tr>
<td>M34</td>
<td>Validate User Selection</td>
<td></td>
</tr>
<tr>
<td>M35</td>
<td>Perform Selected Function</td>
<td></td>
</tr>
<tr>
<td>M36</td>
<td>Display Status Message</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>Merge Descriptions</td>
<td></td>
</tr>
</tbody>
</table>
MSDW-0 PROVIDE MICRODSW

ABSTRACT: This is the environment node for the MICROSDW. The MICROSDW (Microcomputer Software Development Workbench) provides a software development environment hosted on microcomputers. The environment is designed to support the various phases of the software life-cycle and the activities of the software developer. The MICROSDW also provides for modification of the environment and hosting the environment on different microcomputers.
MSDW0 PROVIDE MICROSDW

ABSTRACT: This node shows an overview of the MICROSDW.

MSDW1 PERFORM HARDWARE CONFIGURATION - This provides for description of the hardware environment to the MICROSDW user interface and the other software components of the MICROSDW.

MSDW2 PERFORM SOFTWARE CONFIGURATION - This provides a means of describing the software environment to the user interface, especially what functions and software tools are available.

MSDW3 PROVIDE USER INTERFACE - This provides the user with build in functions to manipulate information in the environment and the ability to execute software tools (programs) in the environment.

MSDW4 MERGE DESCRIPTIONS - This provides for combining the hardware and software descriptions into one description file. This concentrates system information into three files; the MICROSDW system description file, the HELP message system file, and the ERROR message system file.
MSDW1 PERFORM HARDWARE CONFIGURATION

ABSTRACT: Perform Hardware Configuration - This provides for description of the hardware environment to the MICROSDW user interface and the other software components of the MICROSDW. This node is composed of four functions for installation of different hardware elements of the environment.

MSDW11 INSTALL TERMINAL CHARACTERISTICS - This function provides for modification of the terminal characteristic descriptions contained in a terminal characteristics file.

MSDW12 INSTALL PRINTER CHARACTERISTICS - This function provides for modification of the printer characteristic descriptions contained in a printer characteristics file.

MSDW13 INSTALL KEYBOARD CHARACTERISTICS - This function provides for modification of the keyboard characteristic descriptions contained in a keyboard characteristics file.

MSDW14 INSTALL COMMUNICATIONS CHARACTERISTICS - This function provides for modification of the communications characteristic descriptions contained in a communications characteristics file.
MSDW11 INSTALL TERMINAL CHARACTERISTICS

ABSTRACT: This function provides for modification of the terminal characteristic descriptions contained in a terminal characteristics file. This provides for definition of a new terminal characteristic file, the selection of a previously defined terminal characteristic file, and validation of the terminal characteristic file information for use by the system.

MSDW111 DETERMINE SELECTION TYPE - analyzes the type of terminal installation to be performed. If the user selects to define a terminal the Create Terminal Definition function will be invoked, otherwise, the Select Predefined Terminal function will be invoked.

MSDW112 CREATE TERMINAL DEFINITION - This function will: 1. prompt the user for a terminal name, 2. copy the terminal template file definition to a file with the terminal name, 3. invoke a text editor for the user to modify the new terminal definition file with and, 4. create a new terminal characteristic file by copying the new terminal definition file.

MSDW113 SELECT PREDEFINED TERMINAL - This function displays the predefined terminal definitions, prompts the user to select one of the definitions and, when a selection has been made copies the predefined terminal definition file to the terminal characteristic file.

MSDW114 VALIDATE TERMINAL CHARACTERISTIC - This function reads a terminal characteristic file, validates the format of the data in the file, performs "reasonableness" checks on the data, outputing the terminal description, and an exception report.
MSDW12 INSTALL PRINTER CHARACTERISTICS

ABSTRACT: This function provides for modification of the printer characteristic descriptions contained in a printer characteristics file. This provides for definition of a new printer characteristic file, the selection of a previously defined printer characteristic file, and validation of the printer characteristic file information for use by the system.

MSDW121 DETERMINE SELECTION TYPE – analyzes the type of printer installation to be performed. If the user selects to define a printer the Create Printer Definition function will be invoked, otherwise, the Select Predefined Printer function will be invoked.

MSDW122 CREATE PRINTER DEFINITION – This function will: 1. prompt the user for a printer name, 2. copy the printer template file definition to a file with the printer name, 3. invoke a text editor for the user to modify the new printer definition file with and, 4. create a new printer characteristic file by copying the new printer definition file.

MSDW123 SELECT PREDEFINED PRINTER – This function displays the predefined printer definitions, prompts the user to select one of the definitions and, when a selection has been made copies the predefined printer definition file to the printer characteristic file.

MSDW124 VALIDATE PRINTER CHARACTERISTIC – This function reads a printer characteristic file, validates the format of the data in the file, performs "reasonableness" checks on the data, outputing the printer description, and an exception report.
ABSTRACT: This function provides for modification of the keyboard characteristic descriptions contained in a keyboard characteristics file. This provides for definition of a new keyboard characteristic file, the selection of a previously defined keyboard characteristic file, and validation of the keyboard characteristic file information for use by the system.

MSDW131 DETERMINE SELECTION TYPE - analyzes the type of keyboard installation to be performed. If the user selects to define a keyboard the Create Keyboard Definition function will be invoked, otherwise, the Select Predefined Keyboard function will be invoked.

MSDW132 CREATE KEYBOARD DEFINITION - This function will: 1. prompt the user for a keyboard name, 2. copy the keyboard template file definition to a file with the keyboard name, 3. invoke a text editor for the user to modify the new keyboard definition file with and, 4. create a new keyboard characteristic file by copying the new keyboard definition file.

MSDW133 SELECT PREDEFINED KEYBOARD - This function displays the predefined keyboard definitions, prompts the user to select one of the definitions and, when a selection has been made copies the predefined keyboard definition file to the keyboard characteristic file.

MSDW134 VALIDATE KEYBOARD CHARACTERISTIC - This function reads a keyboard characteristic file, validates the format of the data in the file, performs "reasonableness" checks on the data, outputing the keyboard description, and an exception report.
MSDW14 INSTALL COMMUNICATIONS CHARACTERISTICS

ABSTRACT: This function provides for modification of the communications characteristic descriptions contained in a communications characteristics file. This provides for definition of a new communications characteristic file, the selection of a previously defined communications characteristic file, and validation of the communications characteristic file information for use by the system.

MSDW141 DETERMINE SELECTION TYPE - analyzes the type of communications installation to be performed. If the user selects to define a communications the Create Communications Definition function will be invoked, otherwise, the Select Predefined Communications function will be invoked.

MSDW142 CREATE COMMUNICATIONS DEFINITION - This function will: 1. prompt the user for a communications name, 2. copy the communications template file definition to a file with the communications name, 3. invoke a text editor for the user to modify the new communications definition file with and, 4. create a new communications characteristic file by copying the new communications definition file.

MSDW143 SELECT PREDEFINED COMMUNICATIONS - This function displays the predefined communications definitions, prompts the user to select one of the definitions and, when a selection has been made copies the predefined communications definition file to the communications characteristic file.

MSDW144 VALIDATE COMMUNICATIONS CHARACTERISTIC - This function reads a communications characteristic file, validates the format of the data in the file, performs "reasonableness" checks on the data, and outputs a communications description and exception report.
MSDW2 PERFORM SOFTWARE CONFIGURATION

ABSTRACT: This provides for installation of information related to the software configuration of the MICROSDW.

MSDW21 INSTALL SOFTWARE CONFIGURATION - provides the capability to modify the software configuration of the MICROSDW. The software configuration consists of the functions built into the MICROSDW and the programs external to the MICROSDW which may be executed by the User Interface. This function analyzes the software configuration and produces a software configuration description for use by the User Interface.

MSDW22 INSTALL HELP MESSAGES - provides for the modification of the Help Messages displayed by the Help function of the MICROSDW User Interface. The Help Messages describe the functions/options of the MICROSDW User Interface and the programs it executes. The function produces a Help Description (Index) and Help System file containing the indexed Help Messages.

MSDW23 INSTALL ERROR MESSAGES - provides for the modification of the Error Messages displayed by the User Interface when an error is encountered by the User Interface or its functions. This function produces an Error Description (Index) and an Error System file containing the indexed Error Messages.
MSDW21 INSTALL SOFTWARE CONFIGURATION

ABSTRACT: This function provides the capability to modify the software configuration of the MICROSDW. The software configuration consists of the functions built into the MICROSDW and the programs external to the MICROSDW which may be executed by the User Interface.

MSDW211 MODIFY SOFTWARE CONFIGURATION TEXT FILE - this capability is provided by a text editor on the host microcomputer and user inputs.

MSDW212 VALIDATE SOFTWARE CONFIGURATION TEXT FILE - reads the text file and compares it with the rules for the format of the Software Configuration Text file reporting any exceptions to the rules to the user.

MSDW213 BUILD SOFTWARE CONFIGURATION DESCRIPTION - reads the Software Configuration Text file and creates a Software Configuration Description for access by MICROSDW User Interface. The Software Configuration Description is dated combined with the Hardware descriptions by Merge Descriptions into a MICROSDW System Description. Only Software Configuration Text files with no errors detected by Validate Software Configuration Text file should be input to this function.
MSDW22 INSTALL HELP MESSAGES

ABSTRACT: This provides for the modification of the Help Messages displayed by the Help function of the MICROSDW User Interface. The Help Messages describe the functions/options of the MICROSDW User Interface and the programs it executes.

MSDW221 MODIFY HELP TEXT FILE - this capability is provided by a text editor on the host microcomputer and user inputs.

MSDW222 VALIDATE HELP TEXT FILE - reads the text file and compares it with the rules for the format of the Help Text file reporting any exceptions to the rules to the user.

MSDW223 BUILD HELP SYSTEM FILE - reads the Help Text file and creates a Help System file for access by MICROSDW User Interface. Only Help Text files with no errors detected by Validate Help Text file should be input to this function. A HELP Description is output for combination with the Hardware, Software, and Error Descriptions, to create the MICROSDW Description file. The Help Description is an index to the Help messages in the Help System file.
MSDW23 INSTALL ERROR MESSAGES

ABSTRACT: This function provides for the modification of the Error Messages displayed by the User Interface when an error is encountered by the User Interface or its functions.

MSDW231 MODIFY ERROR TEXT FILE - this capability is provided by a text editor on the host microcomputer and user inputs.

MSDW232 VALIDATE ERROR TEXT FILE - reads the text file and compares it with the rules for the format of the Error Text file reporting any exceptions to the rules to the user.

MSDW233 BUILD ERROR SYSTEM FILE - reads the Error Text file and creates a Error System file for access by MICROSDW User Interface. Only Error Text files with no errors detected by Validate Error Text file should be input to this function. An Error Description is output for combination with the Hardware, Software, and Help Descriptions, to create the MICROSDW Description file. The Error Description is an index to the Error messages in the Error System file.
MSDW3 PROVIDE USER INTERFACE

ABSTRACT: This function provides the interface between the user and the components of the MICROSDW. The User Interface is divided into five subfunctions which provide for initialization of the user interface, prompting the user for inputs, obtaining and interpreting user inputs, and performing functions selected by the user.

MSDW31 INITIALIZE USER INTERFACE - provides for reading the MICROSDW System Description describing the hardware and software components of the MICROSDW and initializing data structures internal to the User Interface.

MSDW32 PROVIDE MENU - builds and displays a menu. The menu system is hierarchical, only menu options valid for a particular location in the hierarchy are displayed.

MSDW33 READ USER SELECTION - provides for reading the menu selection entered by the user.

MSDW34 VALIDATE USER SELECTION - provides for comparing the user selection with the valid options for the current level in the menu hierarchy.

MSDW35 PERFORM SELECTED FUNCTION - determines if the function is internal or external to the User Interface. If the function is internal it is immediately executed. If the function is external to the User Interface it is located on the current disks and executed.
MSDW31 INITIALIZE USER INTERFACE

ABSTRACT: This provides for reading the MICROSDW System Description describing the hardware and software components of the MICROSDW and initializing data structures internal to the User Interface.

MSDW311 INITIALIZE HARDWARE DESCRIPTIONS - is responsible for initialization of all internal Hardware description data structures.

MSDW312 INITIALIZE SOFTWARE DESCRIPTIONS - is responsible for initialization of the internal data structures describing the Software Configuration.
MSDW311 INITIALIZE HARDWARE DESCRIPTIONS

ABSTRACT: This is responsible for initialization of all internal Hardware description data structures.

MSDW3111 READ TERMINAL DESCRIPTION - reads the Terminal description into an internal data structure (TERMINAL).

MSDW3112 READ PRINTER DESCRIPTION - reads the Printer description into an internal data structure (PRINTER).

MSDW3113 READ KEYBOARD DESCRIPTION - reads the Keyboard description into an internal data structure (KEYBOARD).

MSDW3114 READ COMMUNICATIONS DESCRIPTION - reads the description into an internal data structure (COMMUNICATIONS).
MSDW312 INITIALIZE SOFTWARE DESCRIPTION

ABSTRACT: This is responsible for initialization of the internal data structures describing the Software Configuration.

MSDW3121 READ SOFTWARE DESCRIPTION - initializes the Software Configuration data structures for use by Provide Menu, Validate User Selection, and Perform Selected Function.

MSDW3122 READ HELP DESCRIPTION - initializes the Help Index data structure for use by the Help function.

MSDW3123 READ ERROR DESCRIPTION - initializes the Error Index data structure for use by the Display Status Message function.
APPENDIX B

Preliminary Design Requirements Cross-reference
APPENDIX B

Preliminary Design Requirements Cross-reference

Introduction

The purpose of this appendix is to cross-reference the requirements defined in the Requirements Definition phase with the MICROSDW functions and characteristics described in the Preliminary Design phase.

Environment Goals and Characteristics

The following requirements lists outline: the goals to be achieved through the use of an integrated software development environment; the characteristics required in a software development environment; and the characteristics required in the tools the environment is composed of. The goals and characteristics which follow have been consolidated from a number of sources (15; 11; 29; 39; 14). The primary goals of an integrated software development environment follow:

<table>
<thead>
<tr>
<th>REQ #</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>To improve the productivity of software developers in all stages of the software life-cycle.</td>
</tr>
<tr>
<td>1.1.1</td>
<td>Requirements Definition</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Preliminary Design</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Detailed Design</td>
</tr>
<tr>
<td>1.1.4</td>
<td>Implementation (Coding)</td>
</tr>
<tr>
<td>1.1.5</td>
<td>Integration</td>
</tr>
<tr>
<td>1.1.6</td>
<td>Operation &amp; Maintenance</td>
</tr>
<tr>
<td>1.1.7</td>
<td>Testing &amp; Validation</td>
</tr>
<tr>
<td>1.2</td>
<td>To produce flexible, reliable software with a minimum of errors that meets the needs (requirements) of the user (15;32,34,40).</td>
</tr>
</tbody>
</table>

B - 2
1.3 To increase the use of existing software.
1.4 To provide for smooth transitions between life-cycle phases and traceability of information from one phase to the next.
1.5 To provide automated creation and maintenance of life-cycle documents (15:35).
1.6 To reduce the time necessary to train software developers.
1.7 To increase the availability of information about software development projects to the developers and management.

Software Development Environment Characteristics. The key characteristics of the development environment are the next lower level of requirements in a "requirements hierarchy". The characteristics of the the environment indicate what character traits the environment must have to satisfy the goals of the environment. Following is a list of characteristics a software development environment must have:

<table>
<thead>
<tr>
<th>REQ #</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Must support an integrated life-cycle software development methodology (11:14; 14:50; 15:37).</td>
</tr>
<tr>
<td></td>
<td>2.1.1 Requirements Definition</td>
</tr>
<tr>
<td></td>
<td>2.1.2 Preliminary Design</td>
</tr>
<tr>
<td></td>
<td>2.1.3 Detailed Design</td>
</tr>
<tr>
<td></td>
<td>2.1.4 Implementation (Coding)</td>
</tr>
<tr>
<td></td>
<td>2.1.5 Integration</td>
</tr>
<tr>
<td></td>
<td>2.1.6 Operation &amp; Maintenance</td>
</tr>
<tr>
<td></td>
<td>2.1.7 Testing &amp; Validation</td>
</tr>
<tr>
<td>2.2</td>
<td>Must provide for automated and traceable transitions between the life-cycle phases and products (Wasl:8; 14:50).</td>
</tr>
<tr>
<td>2.3</td>
<td>Must be flexible, allowing for the addition, deletion, or modification of the components (tools) of the environment (11:24; 15:34,43).</td>
</tr>
<tr>
<td>2.4</td>
<td>Must be user friendly (11:14; 29:38).</td>
</tr>
</tbody>
</table>
2.5 Must provide on-line Help and/or Tutorial facilities (11:24; 39:232).

2.6 Must support interactive software development.

2.7 Must support on-line creation and maintenance of life-cycle products (11:8; 15:35,42).

2.7.1 Requirements Definition Document
2.7.2 Preliminary Design Document
2.7.3 Detailed Design Document
2.7.4 Implementation (Coding)
2.7.5 Integration
2.7.6 Operation & Maintenance Manuals
2.7.7 Testing & Validation Plans and Results

2.8 Must provide a means of storing and updating software development information in a central repository for project information (11:8,16; 29:39).

2.9 Must support "everyday" activities of software developers (39:232).

2.10 Must support pre-fabrication and reuse of software (29:38; 15:39).

2.11 Must provide for checking the completeness and consistency of life-cycle products (15:41).

2.12 Must provide for communication (information transfer/sharing) with other computer systems (14:51).

**Software Tool Characteristics.** The characteristics of individual software tools are the next level of requirements below the software environment characteristics and environment goals. The software tools must have these characteristics to meet the goals and required characteristics of the environment:

**REQ # DESCRIPTION**

3.1 Should provide for graphical display of information.

3.2 Must provide on-line Help and/or Tutorial facilities (14:50; 39:232).
3.3 Should be adaptable to the experienced and inexperienced user (14:50; 33:14).

3.4 Must be functionally complete (ie. support a single or small number of functions) (14:50; 11:24).

3.5 Must not require knowledge of how to use another software tool to use it.

3.6 Must provide for collection of "useful" information about its use/user (14:50; 11:25).

3.7 Must be consistent with other tools in the environment (user interface, functionality, communication protocol) (14:50; 11:14,24).

3.8 Must have adequate documentation (User's manual, Installation guide, maintenance manual).

Minimum Functions

Certain functional capabilities are necessary to meet the goals and characteristics required in a software development environment. The MICROSDW implements a subset of the capabilities of the Software Development Workbench. The capabilities chosen for the MICROSDW are designed to meet the requirements listed earlier. The functional capabilities chosen for the MICROSDW are designed to provide support for all phases of the software life-cycle with additional utility functions to support the "everyday" activities of the user. The functions of the MICROSDW environment, including references to requirement numbers follow.

User Interface (F1). This function is required to provide the user with a "friendly" interface to the software tools, information, and operating system of the MICROSDW. The user interface provides the following capabilities:

1. To execute MICROSDW tools (programs).
2. To Locate MICROSDW programs, documents, or data
files.
3. To view "Help" or "Tutorial" information the operation of the MICROSDW and the tools it is composed of.
4. To modify the configuration of the MICROSDW environment, including:
   - Addition, Deletion, and Movement of software tools
   - File manipulation (Delete, Move, Copy, View)
II7
5. To communicate with the user through interactive prompts and responses on a view screen.

Requirements fulfilled by the User Interface (F1):

<table>
<thead>
<tr>
<th>REQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Provides access to functions/programs supporting the phases of the software life cycle (1.1.1 - 1.1.7)</td>
</tr>
<tr>
<td>1.6</td>
<td>User prompting/help info. reduces training time.</td>
</tr>
<tr>
<td>2.1</td>
<td>(Same as 1.1) (2.1.1 - 2.1.7)</td>
</tr>
<tr>
<td>2.3</td>
<td>Provided by the Install function of the User Interface</td>
</tr>
<tr>
<td>2.4</td>
<td>User friendliness (TBD)</td>
</tr>
<tr>
<td>2.5</td>
<td>Help option and messages provided</td>
</tr>
<tr>
<td>2.6</td>
<td>User Interface is an interactive program</td>
</tr>
<tr>
<td>2.7</td>
<td>Supports &quot;on-line&quot; software development (2.7.1 - 2.7.7)</td>
</tr>
<tr>
<td>2.9</td>
<td>Provides built-in functions commonly used and access to other programs supporting &quot;every-day activities&quot;</td>
</tr>
<tr>
<td>3.2</td>
<td>(Same as 2.5)</td>
</tr>
<tr>
<td>3.3</td>
<td>Allows experienced users to &quot;type ahead&quot; when entering menu keywords</td>
</tr>
<tr>
<td>3.5</td>
<td>Menu prompting and help information elliminates the need to know how to run another program to run this one</td>
</tr>
<tr>
<td>3.6</td>
<td>Allows recording of the menu selections made by the user</td>
</tr>
<tr>
<td>3.8</td>
<td>(TBD)</td>
</tr>
</tbody>
</table>

Requirements Specification (F2). This function shall provide for the entry, modification, deletion, tracing, printing, and display of requirements (try to host a subset of the SREM system on the MICROSDW).

Requirements fulfilled by the Requirements Specification (F2):

<table>
<thead>
<tr>
<th>REQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1</td>
<td>Supports Automated Requirements Definition</td>
</tr>
</tbody>
</table>
1.2 Helps to reduce number of software errors through complete requirements specifications
1.5 Produces the requirements specification
1.7 Formally documents the requirements

2.1.1 Supports requirements specification
2.2 Starts the traceability of requirements through the life-cycle
2.7.1 (Same as 2.1.1)
2.7.7 Supports the development of Test Plans
2.8 Provides a means of storing requirements information
2.11 Provides the first step in documenting what the software must do, produces a baseline for checking later life-cycle products against for completeness

**Structure Charts (F3).** This function shall provide support for the Preliminary Design phase of the software life-cycle. The structure charting function shall provide the following capabilities:

1. Create structure chart nodes and data element names.
2. Delete structure chart nodes and data element names.
3. Modify structure chart nodes and data element names.
4. Display structure chart nodes and data element names.
5. Print structure chart nodes and data element names.
6. Display a parent node and up to seven child nodes.
7. Print a displayed structure chart.
8. Numbering of nodes.
9. Ability to distinguish predefined child nodes.
10. Create "skeletal" data dictionary entries.
11. Create "skeletal" PDL.

Requirements fulfilled by the Structure Charter (F3):

<table>
<thead>
<tr>
<th>BRQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2</td>
<td>Supports the Preliminary Design of software</td>
</tr>
<tr>
<td>1.1.3</td>
<td>Supports Detailed Design by providing a software structure to work from</td>
</tr>
<tr>
<td>1.1.7</td>
<td>Provides a functional breakdown of the software which can be used for testing</td>
</tr>
<tr>
<td>1.2</td>
<td>Helps to identify errors in the design</td>
</tr>
<tr>
<td>1.5</td>
<td>Provides for creation and maintenance of Preliminary Design information</td>
</tr>
<tr>
<td>1.7</td>
<td>Produces a graphical display of the software structure making information about the software more easily accessible</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Supports preliminary design</td>
</tr>
</tbody>
</table>
Data Dictionary (F4). This function provides support for all phases of the software life-cycle, particularly the Preliminary and Detailed Design phases. Data Dictionaries shall provide a central repository for data about a software project. The Data Dictionary function shall provide the following capabilities:

1. Create data dictionary entries.
2. Delete data dictionary entries.
3. Modify data dictionary entries and attributes.
4. Display data dictionary entries.
5. Print data dictionary entries.
7. Display selected data dictionary entries.
8. "Batch" print groups (ALL, RELATED, SPECIFIC list, or TYPE) of data dictionary entries in alphabetic or hierarchical order.
9. Print/Display a "who-calls" structure for a given entry (similar to listing related entries).

Requirements fulfilled by the Data Dictionary (F4):

REQ #  Fulfillment
1.1 Supports all phases of life-cycle by recording information about each phase (1.1.1 - 1.1.7)
1.2 Helps to reduce errors
1.3 Can increase the use of existing software by documenting what the software does
1.4 Provides for traceability and transitions by recording information
1.5 Supports automated documentation
1.6 Reduces training time by providing information
1.7 Increases information available to developers and managers

2.1 Supports all phases of Life-cycle (2.1.1 - 2.1.7)
2.2 (same as 1.4)
2.4 (TBD)
2.5 (TBD)
2.6 Tool is interactive
2.7 Supports creation and maintenance of Life-cycle products (2.7.1 - 2.7.7)
2.8 Provides storage and retrieval capability for software development information
2.10 By providing information about software
2.11 Provides for checking completeness and consistency of software design by storing information about the software design for analysis by other software development tools

Program Design Language (PDL) (F5). This function shall provide support for the detailed design and coding phases of the software life-cycle. The PDL function provides the following capabilities:

1. Creation of PDL files.
2. Interactive syntax directed editing of PDL files.
3. "Structured" printing of PDL files.
4. Definition of the PDL grammar.
5. Modification of the PDL grammar.

Requirements fulfilled by the Program Design Language (F5):

REQ # Fulfillment
1.1.3 Supports detailed design and following Life-cycle
Coding Support (F6). This function provides software tools to support the coding phase of the software life-cycle. This function shall provide at least one of each of the following tools: a high level language (e.g., Pascal, Fortran, C, etc.) compiler, assembler, linker, and debugger. Optionally, this function may provide cross-referencers, cross-compilers, and cross-assemblers to support code generation for other computers.

Requirements fulfilled by Coding Support (F6):
Source Code Control System (F7). This function shall provide the capability to store and retrieve multiple on-line versions of files including, but not limited to, source code, assembler code, manuals, documents, and text files (Roc75).

Requirements fulfilled by the Source Code Control System (F7):

REQ # Fulfillment
1.1 Improves productivity in all phases of software life-cycle by making available old versions of software documents with a minimal use of storage space (1.1.1 - 1.1.7).
1.2 Allows recovery of previous versions of software
1.4 Preserves information
1.5 Supports maintenance of documents
1.7 Makes more information available about software and documents (# of changes, versions, what changes were made)

2.1 Supports integrated life-cycle software development methodology (2.1.1 - 2.1.7)
2.2 Helps preserve information for tracing from one phase to the next
2.4 (TBD)
2.5 (TBD)
2.6 (TBD)
2.7 Supports creation and maintenance of life-cycle documents (2.7.1 - 2.7.7)
2.8 Provides for storage and maintenance of software
development information
2.9 Also supports storage of multiple versions of text
files such as letters or reports
2.10 Provides for storage of software to reuse
2.11 Provides information to use in checking completeness
and consistency of software products
3.4 Provides file management function
3.5 (TBD)
3.6 (TBD)
3.7 (TBD)
3.8 (TBD)

Utility Functions (P8). These functions provide support
for the miscellaneous activities of the user. The following
functions were selected because of their ability to support
one or more activities of the user, and because they have the
potential to save the user much "busy" work.

Word Processing (P8.1). This function shall
support the text editing, formatting, and printing
requirements of the user.

Requirements fulfilled by Word Processing (P8.1):

REO # Fulfillment
1.1 Can improve productivity of software developers in
all phases of software life-cycle (1.1.1 -1.1.7)
1.5 Supports automated creation and maintenance of
life-cycle documents
1.7 Documents are easily reproduced, modified, and
information extracted for inclusion in reports and
presentations
2.4 (TBD)
2.5 (TBD)
2.6 Supports interactive software development
2.7 Supports on line creation and maintenance of
life-cycle products (2.7.1 - 2.7.7)
2.9 Supports "everyday" activities of software developers
by providing an automated means of creating reports
and presentations
3.1 (TBD)
3.2 (TBD)
3.3 (TBD)
3.4 Supports wordprocessing requirements
3.5 (TBD)
Communications (F8.2). This function provides the user with the capability to communicate with remote computers through a modem over standard phone lines. The function will provide the user with the capability to send and receive files, and act as a "dumb" terminal to a remote computer.

Requirements fulfilled by Communications (F8.2):

<table>
<thead>
<tr>
<th>REQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>Can increase the use of existing software by providing access to remote computer systems</td>
</tr>
<tr>
<td>1.7</td>
<td>Provides for transfer of software, or software development reports or documentation to other developers or management</td>
</tr>
<tr>
<td>2.12</td>
<td>Provides for communication between computer systems</td>
</tr>
<tr>
<td>3.2</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.3</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.4</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.5</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.7</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.8</td>
<td>(TBD)</td>
</tr>
</tbody>
</table>

File Compare (F8.3). This function provides the user with the capability to compare two text or two binary files. The function shall display the differences in the files in a readable format. The differences of between two files shall be displayed on a display screen or written to a file.

Requirements fulfilled by File Compare (F8.3):

<table>
<thead>
<tr>
<th>REQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td>2.11</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.5</td>
<td>(TBD)</td>
</tr>
</tbody>
</table>

Database Management (F8.4). This function provides the user with a general data management capability. The Data
Base Management function provides the user with the ability to create, modify, delete, and combine database files and information. The Database Management function may be used to store information such as Software Trouble Reports, code status, or data dictionary information.

Requirements fulfilled by Database Management (F8.4):

<table>
<thead>
<tr>
<th>REQ #</th>
<th>Fulfillment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>Stores information transferred from one phase to the next</td>
</tr>
<tr>
<td>1.5</td>
<td>Provides information for inclusion in documentation and reports</td>
</tr>
<tr>
<td>1.6</td>
<td>Make development information available to new programmers</td>
</tr>
<tr>
<td>1.7</td>
<td>Increases availability of information about software development</td>
</tr>
<tr>
<td>2.5</td>
<td>(TBD)</td>
</tr>
<tr>
<td>2.7</td>
<td>Supports on-line creation and maintenance of software life-cycle products (2.7.1 - 2.7.7)</td>
</tr>
<tr>
<td>2.8</td>
<td>Stores information for update and retrieval</td>
</tr>
<tr>
<td>2.9</td>
<td>Stores &quot;everyday&quot; information for software developers</td>
</tr>
<tr>
<td>2.11</td>
<td>Provides information to be used in checking completeness and consistency of life-cycle products</td>
</tr>
<tr>
<td>3.4</td>
<td>Supplies database management function</td>
</tr>
<tr>
<td>3.5</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.6</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.7</td>
<td>(TBD)</td>
</tr>
<tr>
<td>3.8</td>
<td>(TBD)</td>
</tr>
</tbody>
</table>
**Requirements to Function(s) Cross-reference**

The following table shows which function(s) satisfy which requirements.

**FUNCTIONS**

<table>
<thead>
<tr>
<th>REQ #</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8.1</th>
<th>F8.2</th>
<th>F8.3</th>
<th>F8.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.3</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.6</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.1</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.2</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.3</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.6</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.1</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.2</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.3</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.4</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** ? indicates "To Be Determined" when actual software tool is selected and tested.
<table>
<thead>
<tr>
<th>REO #</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8.1</th>
<th>F8.2</th>
<th>F8.3</th>
<th>F8.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7.6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7.7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.9</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.10</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.11</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** ? indicates "To Be Determined" when actual software tool is selected and tested.
APPENDIX C

MICROSDW Structure Charts
APPENDIX C

MICROSDW Structure Charts

Introduction

This appendix contains the structure charts documenting the MICROSDW Preliminary and Detailed Design phases. The structure charts for the MICROSDW Install program BUILDDAT follow. The process, variable (parameter), and alias descriptions for the structure charts are located in Appendix D (Data Dictionary). The structure charts for the MICROSDW User Interface are located in Appendix C of (30).

The following Structure Chart Node Listing cross-references the process nodes to the Pascal source file the process is located in. The Node Listing also contains the page number of the structure chart which shows the subprocesses of the parent process. For processes which do not have any subprocesses the page number references the first structure chart the process appears on. Processes are listed only the first time they occur in the structure charts.

Previously defined processes are indicated by a small filled in triangle in the lower left corner of the process box.
### Structure Chart Node Listing

<table>
<thead>
<tr>
<th>Node #</th>
<th>Name</th>
<th>Location (file)</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MICROSDW</td>
<td></td>
<td>C - 4</td>
</tr>
<tr>
<td>1</td>
<td>BUILD SYSTEM DATA (BUILDDAT)</td>
<td>BUILDDAT.PAS</td>
<td>- 5</td>
</tr>
<tr>
<td>1.1</td>
<td>make_terminal</td>
<td>BUILDDAT.PAS</td>
<td>- 5</td>
</tr>
<tr>
<td>1.2</td>
<td>make_printer</td>
<td>BUILDDAT.PAS</td>
<td>- 5</td>
</tr>
<tr>
<td>1.3</td>
<td>make_keyboard (not implemented)</td>
<td></td>
<td>- 5</td>
</tr>
<tr>
<td>1.4</td>
<td>make_communications (not implemented)</td>
<td></td>
<td>- 5</td>
</tr>
<tr>
<td>1.5</td>
<td>make_menu</td>
<td>MAKENUM.PAS</td>
<td>- 7</td>
</tr>
<tr>
<td>1.5.1</td>
<td>init_menu</td>
<td>MAKENUM.PAS</td>
<td>- 9</td>
</tr>
<tr>
<td>1.5.2</td>
<td>readme</td>
<td>READMENU.PAS</td>
<td>- 10</td>
</tr>
<tr>
<td>1.5.2.1</td>
<td>get_word</td>
<td>GETWORD.PAS</td>
<td>- 10</td>
</tr>
<tr>
<td>1.5.3</td>
<td>addtomen</td>
<td>ADDTOMEN.PAS</td>
<td>- 11</td>
</tr>
<tr>
<td>1.5.3.1</td>
<td>addword</td>
<td>ADDWORD.PAS</td>
<td>- 12</td>
</tr>
<tr>
<td>1.5.3.1.1</td>
<td>btlocate</td>
<td>AVL-4.PAS</td>
<td>- 13</td>
</tr>
<tr>
<td>1.5.3.1.1.1</td>
<td>isemptyy</td>
<td>AVL-1.PAS</td>
<td>- 13</td>
</tr>
<tr>
<td>1.5.3.1.1.2</td>
<td>dataval</td>
<td>AVL-1.PAS</td>
<td>- 13</td>
</tr>
<tr>
<td>1.5.3.1.1.3</td>
<td>lchild</td>
<td>AVL-1.PAS</td>
<td>- 13</td>
</tr>
<tr>
<td>1.5.3.1.1.4</td>
<td>rchild</td>
<td>AVL-1.PAS</td>
<td>- 13</td>
</tr>
<tr>
<td>1.5.3.1.2</td>
<td>avlinsert</td>
<td>AVL-4.PAS</td>
<td>- 14</td>
</tr>
<tr>
<td>1.5.3.1.2.1</td>
<td>makebt</td>
<td>AVL-2.PAS</td>
<td>- 15</td>
</tr>
<tr>
<td>1.5.3.1.2.2</td>
<td>leftbalance</td>
<td>AVL-3.PAS</td>
<td>- 15</td>
</tr>
<tr>
<td>1.5.3.1.2.3</td>
<td>rightbalance</td>
<td>AVL-3.PAS</td>
<td>- 15</td>
</tr>
<tr>
<td>1.5.3.2</td>
<td>addkeywordtomen</td>
<td>ADDTOMEN.PAS</td>
<td>- 11</td>
</tr>
<tr>
<td>1.5.3.3</td>
<td>define_menu</td>
<td>DEFMENU.PAS</td>
<td>- 16</td>
</tr>
<tr>
<td>1.5.3.3.1</td>
<td>locate_menu_node</td>
<td>LOCMENU.PAS</td>
<td>- 17</td>
</tr>
<tr>
<td>1.5.3.3.1.1</td>
<td>copymenuword</td>
<td>LOCMENU.PAS</td>
<td>- 17</td>
</tr>
<tr>
<td>1.5.3.4</td>
<td>define_call_routine</td>
<td>DEFCALL.PAS</td>
<td>- 18</td>
</tr>
<tr>
<td>1.5.3.4.1</td>
<td>findcall</td>
<td>DEFCALL.PAS</td>
<td>- 18</td>
</tr>
<tr>
<td>1.5.3.4.2</td>
<td>addcall</td>
<td>DEFCALL.PAS</td>
<td>- 18</td>
</tr>
<tr>
<td>1.5.4</td>
<td>make_word_list</td>
<td>MAKEPROC.PAS</td>
<td>- 19</td>
</tr>
<tr>
<td>1.5.4.1</td>
<td>lnr</td>
<td>AVL-2.PAS</td>
<td>- 20</td>
</tr>
<tr>
<td>1.5.4.2</td>
<td>calc_min_chars</td>
<td>MAKEPROC.PAS</td>
<td>- 19</td>
</tr>
<tr>
<td>1.5.5</td>
<td>make_decoding_paths</td>
<td>MAKEPROC.PAS</td>
<td>- 21</td>
</tr>
<tr>
<td>1.5.5.1</td>
<td>number_calls</td>
<td>MAKEPROC.PAS</td>
<td>- 21</td>
</tr>
<tr>
<td>1.5.5.2</td>
<td>make_call_records</td>
<td>MAKEPROC.PAS</td>
<td>- 22</td>
</tr>
<tr>
<td>1.5.5.3</td>
<td>make_keyw_records</td>
<td>MAKEPROC.PAS</td>
<td>- 21</td>
</tr>
<tr>
<td>1.5.6</td>
<td>displaytree</td>
<td></td>
<td>- 23</td>
</tr>
<tr>
<td>1.5.6.1</td>
<td>printtree</td>
<td>AVL-2.PAS</td>
<td>- 24</td>
</tr>
<tr>
<td>1.5.7</td>
<td>treedispose</td>
<td>AVL-2.PAS</td>
<td>- 25</td>
</tr>
<tr>
<td>1.5.8</td>
<td>menu_print</td>
<td>MENUPRINT.PAS</td>
<td>- 8</td>
</tr>
<tr>
<td>1.5.9</td>
<td>callsdispose</td>
<td>DISPROC.PAS</td>
<td>- 8</td>
</tr>
<tr>
<td>1.5.10</td>
<td>errmsg</td>
<td>ERRORMSG.PAS</td>
<td>- 8</td>
</tr>
<tr>
<td>1.6</td>
<td>make_help</td>
<td></td>
<td>- 6</td>
</tr>
<tr>
<td>1.7</td>
<td>make_error (not implemented)</td>
<td></td>
<td>- 6</td>
</tr>
</tbody>
</table>

C - 3
PROJECT: MICROSDW

1 MICROSDW Text Files
2 MICROSDW System Files
   MICROSDW.SYS
   ERROR.SYS
   HELP.SYS

MICROSDW

BUILD
SYSTEM
DATA

1

MICROSDW
User
Interface

2
PROJECT: MICROSDW

AUTHOR: Capt Paul A. Moore  TITLE: make_menu  NODE: 1.5

1 word_tree
2 menu_tree
3 menu.txt
4 parse_buf
5 node_counter
6 call_list
7 current_menu
1 doneword
2 wordtree
3 wordpointer

PROJECT: MICROSDW

init_menu
1.5.1

addword
1.5.3.1

AUTHOR: Capt Paul A. Moore
TITLE: initialize menu
NODE: 1.5.1
1. word line
2. start_position
3. stop_position
4. keyword

read_menu

get_word

PROJECT: MICROSILW

AUTHOR: Capt Paul A. Moore
TITLE: read menu definition
NODE: 1.5.2
4.3

PROJECT: MICROSDW

1 wordtree
2 dataval
3 newnode
4 side (left, right, parent)
5 A (wordtree)
6 B (wordtree)

avlinsert
1.5.3.1.2

makebt
1.5.3.1.2.1

leftbalance
1.5.3.1.2.2

rightbalance
1.5.3.1.2.3

AUTHOR: Capt Paul A. Moore
TITLE: avlinsert (continued)
NODE: 1.5.3.1.2 (cont.)
1. word_tree
2. number_of_words
3. decode_dictionary
1. wordtree
2. wordcount
3. wordtree
4. isempty
5. lchild
6. rchild
7. dataval

PROJECT: MICROSDW

AUTHOR: Capt. Paul A. Moore
TITLE: lnr
NODE: 1.5.4.1
1. call_list
2. node_counter
3. word_tree
4. menu_tree

PROJECT: MICROSDW

AUTHOR: Capt Paul A. Moore
TITLE: make_decoding_paths
NODE: 1.5.5

make_decoding_paths
1.5.5

number_calls
1.5.5.1

make_call_records
1.5.5.2

make_keyw_records
1.5.5.3
1. DONEWORD
2. word_tree
3. doneptr
4. found

PROJECT: MICROSDW

make_call_records
1.5.5.2

btlocate
1.5.3.1.1
1. wordtree
2. lchild
3. rchild
4. isempty
5. depth
6. LR

AUTHOR: Capt Paul A. Moore  TITLE: printtree  NODE: 1.5.6.1
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

UNCLASSIFIED P. A. MOORE DEC 84 AFIT/GCS/ENG/84D-18
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A
APPENDIX D

MICROSDW System Data Dictionary
APPENDIX D
MICROSDW System Data Dictionary

Introduction

This appendix contains the MICROSDW System Data Dictionary documenting the MICROSDW Preliminary and Detailed Design phases. The appendix is arranged in six sections: constant definitions, type definitions, data file descriptions, procedure/function descriptions, parameter or variable descriptions, and alias descriptions.

Appendix D Index

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Definitions</td>
<td>D - 1</td>
</tr>
<tr>
<td>Type Definitions</td>
<td>D - 5</td>
</tr>
<tr>
<td>Data File Descriptions</td>
<td>D - 7</td>
</tr>
<tr>
<td>Procedure/Function Descriptions</td>
<td>D - 11</td>
</tr>
<tr>
<td>Parameter/Variable Descriptions</td>
<td>D - 30</td>
</tr>
<tr>
<td>Alias Descriptions</td>
<td>D - 42</td>
</tr>
</tbody>
</table>
Constant Definitions

The constants used by the MICROSDW User Interface and BUILDDAT programs are contained in two files, MSDWCONS.PAS and MENUCONS.PAS. MSDWCONS.PAS is used in both the User Interface and BUILDDAT. MENUCONS.PAS defines constants used by the MENU.TXT file processing procedures of BUILDDAT. The contents of the two files follow.

(file: msdwcons.pas)
(version: 1.1)
(date: 18 August 1984)
(description: this file contains the constant definitions for the MICROSDW routines.)

const
  (* length of array for terminal control data *)
  term_length = 95;
  (* length of array for printer control data *)
  printer_length = 50;
  ff = 12; (* decimal for form feed char *)
  wordlength = 9; (* length of word in storage *)
  num_param_group = 4;
  num_bools = 10;
  num_ints = 10;
  num_reals = 10;
  num_ptr_recs = 3;
  num_ptrs = 200;
  num_words = 75;
  num_msg_dir = 50;
  num_msg_line = 250;
  screen_width = 79;
  ENDCODE = 9999;
  DONEWORD = '$$$$$$$$$$';
const
SUCCESS = 0;
ENDMENU = 0;  { end of MENU.TXT file }
BEGINMENU = 1;
MENUDEF = 2;  { line is a Menu Definition }
CALLDEF = 3;  { line is a Call Routine Definition }
MENUWORD = 4;  { line contains a menu keyword }
ENDDEF = 5;  { End of menu keyword list }
BLANK = ' ';  { MENU.TXT comment line }
COMMENT = '/*';  { MENU.TXT assignment delimiter }
ASSIGNMENT = '=';  { MENU.TXT end of keyword delimiter }
ENDSTR = '.*';  { MENU.TXT comment line }

(* maximum # chars in a menu definition record *)
MAXMENU = 99;

(* ERROR codes *)
(* submenu for menu definition does not exist *)
MERROR1 = 1;

(* incomplete menupath *)
MERROR2 = 2;

(* extra chars following a complete path spec. *)
MERROR3 = 3;

(* a call word cannot be defined for a keyword *)
(* with a submenu *)
MERROR4 = 4;

(* redefinition of a call word for a menu option*)
(* is not allowed *)
MERROR5 = 5;
MERROR6 = 6;
MERROR7 = 7;
MERROR8 = 8;
Type Definitions

The type definitions used by the MICROSDW User Interface and BUILDDAT programs are contained in two files, MSDWTYPE.PAS and MENUTYPE.PAS. MSDWTYPE.PAS contains general type definitions used by both the User Interface and BUILDDAT. MENUTYPE.PAS contains the type definitions for the dynamic menu data structures created by the menu processing procedures in BUILDDAT.

```plaintext
(* file: msdwtype.pas  *
* version: 1.0        *
* date: 15 August 1984 *
* description: this file contains the  *
* type definitions for the MICROSDW  *
* routines.                  *
******************************************************************************)

type
  ptr_recs = array[1..num_ptr_recs] of integer;

msg = record
  loc_rec : integer;
  length : byte;
end;

dict_buffer = record
  ptrs : array[1..num_ptrs] of ptr_recs;
  words : array[0..num_words] of string[wordlength];
  abbrev : array[0..num_words] of integer;
end;

param_group = record
  bools : array[1..num_bools] of boolean;
  ints : array[1..num_ints] of integer;
  reals : array[1..num_reals] of real;
end;

msg_dat = array[1..num_msg_line] of string[screen_width];
data = record
  param : array[1..num_param_group] of param_group;
  term : array[1..term_length] of byte;
  printr : array[1..printer_length] of byte;
  msg_dir : array[1..num_msg_dir] of msg;
  decode_dict : dict_buffer;
end;

wordtype = string[wordlength];
```
type
  btptr = ^wtnode; (* pointer to a binary word tree *)
wtnode = record
    keyword : wordtype; (* menu keyword, or call routine name *)
    wordnumber : integer; (* # of the word in the "words" array, used in creating decoding paths *)
    bf : integer; (* balance factor *)
    left : btptr; (* left subtree pointer (child) *)
    right : btptr; (* pointer to right subtree *)
  end;
callptr = ^callnode; (* pointer to a "call routine" list*)
callnode= record (* node definition for call list *)
  wordptr : btptr; (* ptr to the word for this call*)
  callnumber : integer; (* # of the call *)
  nextcall : callptr; (* ptr to next node in call list*)
  end;
   mtptr = ^menunode; (* pointer to a menu tree *)
menunode= record (* node definition for menu option *)
  wordptr : btptr; (* pointer to keyword *)
  node_number : integer; (* # of the menunode, used in creating decoding paths *)
  nomatchp : mtptr; (* ptr to next keyword in this menu *)
  matchp : mtptr; (* ptr to next lower submenu *)
  endptr : callptr; (* ptr to call word for this menuoption. *)
  end;
parserecord = record (* parse buffer *)
  rectype : integer; (* menu definition record type*)
  wordstr : string[MAXMENU];(*"parsed" menu def. record*)
  end;
Data File Descriptions

Following are brief data dictionary descriptions of the data files used by the MICROSDW system.

NAME: COMM.TXT
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains the definition of the location(s) of the communication ports and port control commands.

SOURCES:
DESTINATIONS:
COMPOSITION:
PART OF: MICROSDW TEXT FILES
DATA CHARACTERISTICS: Text file
VALUE RANGE:
STORAGE TYPE: Text file

NAME: ERROR.SYS
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains Error messages reformatted from ERROR.TXT for access by the MICROSDW User Interface.

SOURCES:
DESTINATIONS: make_error
COMPOSITION:
PART OF: MICROSDW SYSTEM FILES
DATA CHARACTERISTICS:
VALUE RANGE:
STORAGE TYPE: File
NAME: ERROR.TXT
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains the text for MICROSDW error messages. Error messages are separated by lines containing "<$>" as the first 3 characters on a line. The file is terminated by "<$$>" as the first 4 characters of a line.

SOURCES:
DESTINATIONS:
COMPOSITION: Error messages separated by "<$>" delimiters.
PART OF: MICROSDW TEXT FILES
DATA CHARACTERISTICS:
VALUE RANGE:
STORAGE TYPE: Text file

NAME: HELP.SYS
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains Help messages reformatted from HELP.TXT for access by the MICROSDW User Interface.

SOURCES: make_help

DESTINATIONS:

COMPOSITION: PART OF MICROSDW SYSTEM FILES

DATA CHARACTERISTICS:

VALUE RANGE:

STORAGE TYPE: File

NAME: HELP.TXT

TYPE: DATA ITEM

ALIASES:

DESCRIPTION: This file contains the text for MICRSDW help messages. Help messages are separated by lines containing "<$>" as the first 3 characters on a line. The file is terminated by "<$$>" as the first 4 characters of a line.

SOURCES:

DESTINATIONS:

COMPOSITION: Help messages separated by "<$>" delimiters.

PART OF: MICROSDW TEXT FILES

DATA CHARACTERISTICS:

VALUE RANGE:

STORAGE TYPE: Text file

NAME: KEYBOARD.TXT

TYPE: DATA ITEM

ALIASES:

DESCRIPTION: This file contains the character sequences representing function key and cursor control keys received from the keyboard.

SOURCES:

DESTINATIONS:

COMPOSITION:

PART OF: MICROSDW TEXT FILES

DATA CHARACTERISTICS: Text file

VALUE RANGE:

STORAGE TYPE: Text file

NAME: MENU.TXT

TYPE: DATA ITEM

ALIASES:

DESCRIPTION: This file contains the text description of a menu structure used by the User Interface to prompt users for input and validate user responses. The text menu description is converted into the "decode_dictionary" by make_menu.
SOURCES:
DESTINATIONS:
COMPOSITION: See Menu Grammar Definition, Appendix E.
PART OF: MICROSDW TEXT FILES
DATA CHARACTERISTICS:
VALUE RANGE:
STORAGE TYPE: Text file

NAME: MICROSDW SYSTEM FILES
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: MICROSDW SYSTEM FILES includes three files, MICROSDW.SYS, ERROR.SYS, and HELP.SYS. These files are formatted for initialization of the User Interface.
SOURCES: MICROSDW, BUILD SYSTEM DATA (BUILDDAT)
DESTINATIONS: MICROSDW User Interface
COMPOSITION: MICROSDW.SYS
HELP.SYS
ERROR.SYS

PART OF:
DATA CHARACTERISTICS:
VALUE RANGE:
STORAGE TYPE: File

NAME: MICROSDW TEXT FILES
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: These are the text files describing the hardware and software configuration of the MICROSDW.
SOURCES: MICROSDW
DESTINATIONS: BUILD SYSTEM DATA (BUILDDAT)
COMPOSITION: TERMINAL.TXT MENU.TXT
PRINTER.TXT HELP.TXT
KEYBOARD.TXT ERROR.TXT
COMM.TXT

PART OF:
DATA CHARACTERISTICS: Text files
VALUE RANGE:
STORAGE TYPE: File

NAME: MICROSDW.SYS
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains the descriptions of the MICROSDW host system hardware and menu structure.
SOURCES:
DESTINATIONS:
COMPOSITION: data_file
PART OF: MICROSDW SYSTEM FILES
DATA CHARACTERISTICS: a binary file of one Pascal record
VALUE RANGE:
STORAGE TYPE: File

NAME: PRINTER.TXT
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains the control sequences for printer control.
SOURCES:
DESTINATIONS:
COMPOSITION:
PART OF: MICROSDW TEXT FILES
DATA CHARACTERISTICS: Text file
VALUE RANGE:
STORAGE TYPE: Text file

NAME: TERMINAL.TXT
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This file contains the control sequences for terminal control, including cursor positioning, reverse video, normal video, graphics, etc.
SOURCES:
DESTINATIONS:
COMPOSITION:
PART OF: MICROSDW TEXT FILES
DATA CHARACTERISTICS: Text file
VALUE RANGE:
STORAGE TYPE: Text file
Procedure/Function Descriptions

This section of the Data Dictionary contains the descriptions of the procedures and functions which makeup the program BUILDDAT. The descriptions match the structure charts in Appendix C. Descriptions of the functions and procedures making up the MICROSDW User Interface program are located in Par83: Appendix-C.

**procedure: addcall**

module number: 1.5.3.4.2
version: 1.0
date: 16 August 1984
description: This process adds a node containing a pointer to a call-routine name to the current call-routine list.

inputs: call_list, word_pointer
outputs: call_list, call_pointer
global variables used: none
global constants used: none
modules called: none
calling modules: define_call_routine
files read: none
files written: none

**procedure: addkeywordtomenu**

module number: 1.5.3.2
version: 1.1
date: 3 September 84
description: 1.5.3.2. ADDKEYWORDTOMENU- This process adds a keyword to the "current-menu". A node for the new menu keyword is created and added to the current_menu.

inputs: word_pointer, current_menu, menutree, node_counter
outputs: current_menu, menutree
global variables used: none
global constants used: none
modules called: none
calling modules: addtomenu
files read: none
files written: none

**procedure: addtomenu**

module number: 1.5.3
version: 1.0
date: 16 August 1984
description: 1.5.3 ADDTOMENU- This process receives pointers to
the parse buffer (parse_buf) and the menu data structures (word_tree, menu_tree, call_list).
Addtomenu calls the appropriate subprocesses based on the record type (rectype) of the parse buffer to add information in the buffer to the menu data structures.

inputs: wordtree, menutree, call_list, parse_buf, node_counter, current_menu
outputs: wordtree, menutree, call_list, parse_buf, node_counter, current_menu
global variables used: none
modules called: addword, addkeywordtomenu, define_menu, define_call

procedure: addtomenu
module number: 1.5.3.1
version: 1.0
date: 16 August 1984
description:
1.5.3.1 ADDWORD - This process adds keywords to the word_tree. It calls btlocate to determine if the keyword is already stored in word_tree. Addword retruns a pointer to the location of the keyword in word_tree.

inputs: word,wordtree, wordpointer
outputs: wordtree, wordpointer
modules called: btlocate, avlinsert
files read: addtomenu, define_call_routine
files written: none

procedure: avlinsert
module number: 1.5.3.1.2
date: 20/11/83
version: 1.3
description: Avlinsert inserts a value into a binary tree and balances the resulting tree to produce an AVL (Adelson-Velskii and Landis) height balanced tree. Avlinsert uses the procedures "leftbalance" and "rightbalance" to balance the binary tree.

inputs: element -- > a word to insert in the tree
position -> pointer to the root of the binary tree
outputs: An avl balanced tree.
newnode -> a pointer to the new node added to the tree

global variables used: LPARENT, LEFT, RIGHT

global variables changed: none.

modules called: leftbalance, rightbalance, makebt,  
lchild, rchild, isempty, dataval

calling modules: addword

files read: none.

files written: none.

---

procedure: btlocate
module number: 1.5.3.1.1

date: 8/16/84

version: 1.0

description: btlocate searches a binary tree for a value.  
If the word is located a pointer to the word is returned and the "found" flag is set to true.  
If the word is not located a pointer to the potential parent is returned and the "found" flag is set to false.

inputs: word  ->  a word to locate in the tree  

btree  ->  pointer to the root of the binary tree to search.

outputs: node_ptr  ->  a pointer to the node located in the tree  

found  ->  true if the word is located, false otherwise

---

global constants used:  none.

global variables changed: none.

modules called: lchild, rchild, isempty, dataval

calling modules: addword, make_call_records

files read: none.

files written: none.

---

procedure: builddat
module number: 1

version: 1.2

date: 11 November 84

description: This procedure is the main module for the BUILDDAT program. BUILDDAT performs "install" operations for the MICROSDW. BUILDDAT reads ASCII text files describing the microcomputer hardware and menu system, validates the files and creates the MICROSDW system files (MICROSDW.SYS, ERROR.SYS, and HELP.SYS). The system files are used by the MICROSDW User Interface to interpret user inputs, display prompts, print information, and to access "help" or "error" information.

inputs: none

outputs: none
global variables used: data_file, data_recs, i, resp, resp2, Outfile, Outflag

global constants used: (See MSDWCONS.PAS, MENUCONS.PAS)

modules called: make_param, make_terminal, make_printer, make_help, make_menu

calling modules: none

files read: PARAM.TXT, TERM.TXT, PRINT.TXT, HELP.TXT, MENU.TXT

files written: MICROSDW.SYS, HELP.SYS, BUILDDAT.OUT

procedure: calc_min_chars

module number: 1.5.4.2

version: 1.2

date: 20 October 1984

description:

1.5.4.2 CALC_MIN_CHARS- This process calculates the minimum number of characters necessary to identify a particular keyword. The procedure uses the sorted array of keywords created by LNR to compare keywords. The keywords are compared in pairs character by character until a difference is found. If the minimum length of the first word is greater than its previously calculated minimum the new minimum is saved. The minimum length of the second word is set to the number of the character where the first difference is identified.

inputs: decode_dict, number_of_words

outputs: decode_dict

procedures: callsdispose

module number: 1.5.9

version: 1.0

date: 18 August 1984

description: This procedure disposes of the linked list of call routines data structure (call_list).

inputs: call_list

outputs: call_list

D - 14
procedure: copymenuword
module number: 1.5.3.3.1.1
version: 1.0
date: 16 August 1984
description: 1.5.3.3.2 COPYMENUWORD- This process copies menu words from previously parsed menu_line(s) from MENU.TXT.

inputs: source, start, stop
outputs: start, destination
global variables used: none
global constants used: BLANK
modules called: none
calling modules: locate_menu_node
files read: none
files written: none

procedure: dataval
module number: 1.5.3.1.1.2
date: 19/11/83
version: 1.0
description: To return the data value from a node in a binary tree. A pointer to a node in a tree is input to the "data" function. If the pointer is "nil" ie. does not point to a node the value "minint" is returned as the data value.

inputs: A pointer to a node in a binary tree.
outputs: The value stored in the "information" field of the node pointed to.
global variables used: none.
global variables changed: none.
modules called: none.
calling modules: many
files read: none.
files written: none.

procedure: define_call_routine
module number: 1.5.3.4
version: 1.1
date: 3 September 84
description: 1.5.3.4 DEFINE_CALL_ROUTINE- This process defines a "call-routine" for a menu option. Only menu options which are leaves of the menu tree can have call-routines specified for them. A "call-routine" is simply the name or keyword used by the User Interface to determine which process to call to carry out the action specified by the selected menu option. First the menu path to specify the call-routine for is checked to make sure that it exists,
that it is a leaf node, and that a call-routine has not been already defined for that menu path. The name of the call-routine is then added to the keyword tree. Then the call-routine is added to the list of call routine names.

inputs: menu_tree, word_tree, call_list, parse_buf
outputs: call_list
global variables used: none
global constants used: none
modules called: copymenuword, addword, findcall, addcall, errormsg, locate_menu_node
calling modules: addtomenu
files read: none
files written: none

procedure: define_menu
module number: 1.5.3.3
version: 1.0
date: 16 August 1984
description:

1.5.3.3 DEFINE_MENU- This process sets the current_menu pointer to point to the menu node which a submenu is to be defined for. If the menu option is to use a previously defined submenu, the current_menu pointer will be set to nil and the "match pointer" for the menu option will point to the previously defined submenu. The buffer (parse_buf) contains a sequence of keywords specifying the new menu to be defined. Each keyword in the sequence represents a lower level in the menu hierarchy. A menu node must exist for each of the keywords. If parse_buf indicates that a previously defined submenu is to assigned to this menu a second sequence of keywords is contained in parse_buf specifying the predefined submenu. The previously defined submenu must also exist.

inputs: menu_tree, current_menu, parse_buf
outputs: menu_tree
global variables used: none
global constants used: none
modules called: locate_menu_node, errormsg
calling modules: addtomenu
files read: none
files written: none

procedure: Displaytree
module number: 1.5.6
date: 20/11/83
version: 2.1
description: To print a binary tree in a "readable" manner. Displaytree calls printtree to traverse the left and right subtrees of the root node printing the values of the nodes as it goes. Note: printtree is a recursive procedure.

inputs: p -->a pointer to a node in a binary tree.

outputs: a "readable" binary tree.

global constants used: LEFT, RIGHT

global variables used: Outflag, Outfile

modules called: printtree, rchild, lchild, isempty, dataval

calling modules: make_menu

files read: none.

files written: Outflag

procedure: errormsg
module number: 1.5.10
version: 1.0

date: 21 August 1984
description:

1.5.10 ERRORMSG- This procedure prints out the error messages for the menu processing procedures of BUILDDAT. It receives an error number as input and prints the corresponding message.

inputs:

error number

outputs:

global variables used: Outfile, Outflag

global constants used: error constants

modules called: none

calling modules: menu processing procedures

files read: none

files written: Outfile

procedure: findcall
module number: 1.5.3.4.1
version: 1.0

date: 16 August 1984
description:

1.5.3.4.1 FINDCALL- This process checks the call-routine list to see if a call-routine has been previously defined for another menu path. If the call-routine name is already in the list a pointer to it is returned otherwise a "nil" pointer is returned.

inputs: call_list, word_pointer

outputs: call_pointer

global variables used: none

global constants used: none

modules called: none

calling modules: define_call_routine

files read: none
files written: none

procedure: get_word
module number: 1.5.2.2
version: 1.2
date: 3 September 84
description: 
1.5.2.1 GET_WORD - This process extracts menu keywords from the MENU.TXT file records.
inputs: wordline, start, endpos
outputs: start, word
global variables used: none
modules called: none
calling modules: readmenu
files read: none
files written: none

procedure: init_menu
module number: 1.5.1
version: 1.0
date: 21 August 1984
description: 
1.5.1 INITIALIZE MENU (INITMENU) - This process initializes the pointers to the root nodes of the menu data structures, word_tree, menu_tree, and call_list.
inputs: wordtree, menutree, current_menu, call_list
outputs: wordtree, menutree, current_menu, call_list
modules called: addword
calling modules: make_menu
files read: none
files written: none

procedure: isempty
module number: 1.5.3.1.1.1
date: 19/11/83
version: 1.0
description: To return a boolean value "true" if a tree is empty and "false" if a tree is not empty.
inputs: A pointer to a binary tree.
outputs: The boolean value "true" or "false".
global variables used: none.
global variables changed: none.
modules called: none.
calling modules: many
files read: none.
files written: none.
procedure: lchild  
module number: 1.5.3.1.1.3  
date: 19/11/83  
version: 1.0  
description: To return a pointer to the left child of a node in a binary tree. lchild receives a pointer to a node as input, if that pointer is "nil" an error exists because there is not a left child of a "nil" node. In this case an error code is not returned but lchild is also set to "nil".  
inputs: A pointer to the node of which a pointer to its left child is desired.  
outputs: A pointer to the left node of a parent node.  
global variables used: none.  
global variables changed: none.  
modules called: none.  
calling modules: many  
files read: none.  
files written: none.  

procedure: leftbalance  
module number: 1.5.3.1.2.2  
date: 20/11/83  
version: 1.2  
description: To perform balancing of trees unbalanced to the left.  
inputs: A --> Pointer to last node with balance factor not equal to 0.  
        B --> Pointer to left child node of A.  
outputs: A rebalanced binary tree.  
global variables used: none.  
global variables changed: none.  
modules called: none.  
calling modules: avlinsert  

procedure: LNR  
module number: 1.5.4.1  
date: 09/11/83  
version: 1.0  
description: The function of LNR is to traverse a binary tree starting at the root of the tree in a "Left-Node-Right" manner. LNR is a recursive routine. LNR first follows the left pointers of each node until a "nil" pointer is encountered by calling itself with the left pointer of the current node. When a "nil" pointer is found the information in the current node is stored in an array and the index into the array is incremented.  
inputs: P --> Pointer to tree to traverse
wordcount --> the number of words placed in the
output array plus 1.

outputs: data_recs.decode_dict.words --> array containing
sorted values of the tree passed to LNR.

wordcount

global variables used: nil
global variables changed: none.
modules called: none.
calling modules: LNR
files read: none.
files written: none.

procedure: locate_menu_node
module number: 1.5.3.3.1
version: 1.1
date: 3 September 84
description: LOCATE_MENU_NODE- This process receives
a sequence of keywords defining a menu option
and locates the node in the menu.tree for the
last keyword. Locate_menu_node starts at the
root of the menu structure and searches for the
first keyword. If the keyword is found
then the "match pointer" for that keyword is
followed to the next lower level in the menu
hierarchy and the process is repeated for the
next keyword in the keyword sequence. If the
keyword is not found, an error flag is
returned.

inputs: menu_path, node_ptr
outputs: node_ptr, error

global variables used: none

global constants used: BLANK, SUCCESS, MERROR2, MERROR3
modules called: copymenuword
calling modules: define_menu
files read: none
files written: none

procedure: makebt
module number: 1.5.3.1.2.1
date: 20/11/83
version: 1.4
description: Makebt calls the library routine "new" to
allocate storage space for a node in a binary
tree. Each node has four fields:

info --> stores an integer value.
bf --> stores the balance factor for a node
left --> pointer to the node to the left
right --> pointer to the node to the right
Makebt initializes the fields of the new node. Makebt also "links" the parent node to the new node. Makebt is also passed a flag called "side" makebt uses the value of side to determine where to link the node into the parent. The following list shows how a new node may be added:

- side = PARENT → parent := newnode
- side = LEFT → left child of parent := newnode
- side = RIGHT → right child of parent := newnode

inputs: parent → pointer to parent of new node.
value → value to store in new node.
side → flag indicating where to insert new node.
outputs: newnode → pointer to the inserted node.
global variables used: PARENT, LEFT, RIGHT
global variables changed: none.
modules called: lchild, rchild, isempty
calling modules: addtotree, avlinsert
files read: none.
files written: none.

procedure: make_call_records
module number: 1.5.5.2
version: 1.0
date: 17 August 1984
description:
1.5.5.2 MAKE_CALL_RECORDS- This process fills in the data for the call-routine termination records in the decoding path structure.
inputs: call_list, wordtree
outputs:
global variables used: data_recs.decode_dict_ptrs
global constants used: DONEWORD
modules called: btlocate
calling modules: make_decoding_paths
files read: none
files written: none

procedure: make_comm
(module number: 1.4
version: 1.0
date: 12 November 1984
description:
1.4 INSTALL COMMUNICATIONS DEFINITION (MAKE_COMM)- (Not Implemented) This process is responsible for reading the Communication Definition file (COMM.TXT) and storing the information from the file in the "comm" array. The "comm" array is a global array of integers describing various
communications control sequences used by the
User Interface. The "comm" array is part of the
information written to MICROSDW.SYS by
BUILDDAT.

inputs:
outputs:
global variables used: Outflag, Outfile
global constants used:
modules called: none
calling modules: builddat
files read: COMM.TXT
files written: Outfile

procedure: make_decoding_paths
module number: 1.5.5
version: 1.0
date: 17 August 1984
description:
1.5.5 MAKE_DECODING_PATHS- This process calls three
subprocesses to create the decoding paths for
the menu structure. Input to
make_decoding_paths are the pointers to the
call_list, wordtree, menutree, and the number
of nodes in the menutree. Output from the
subprocesses is decode_dictionary pointer
records for the decoding paths.

inputs: call_list, wordtree, menutree,
node_counter
outputs: node_counter
global variables used: Outfile, Outflag
data_recs.decode_dict.ptrs
global constants used: none
modules called: number_calls, make_call_records,
make_keyw_records
calling modules: make_menu
files read: none
files written: Outfile

procedure: make_error (NOT IMPLEMENTED)
module number: 1.7
version: 1.0
date: 13 November 1984
description:
1.7 MAKE_ERROR- (Not Implemented) This procedure
processes the ERROR.TXT file creating an error
index and the ERROR.SYS file.

inputs:
outputs:
global variables used: Outflag, Outfile
global constants used:
modules called: none
calling modules: builddat
files read: ERROR.TXT
files written: ERROR.SYS, Outfile

procedure: make_help
module number: 1.6
version: 2.0
date: 11 November 1984
description:
1.6 MAKE_HELP- This procedure processes the HELP.TXT file creating the help index (msg_dir) stored in the MICROSDW.SYS file and the HELP.SYS file containing the text for the help messages.

inputs: none
outputs: none
global variables used: data_recs.msg_dir, Outflag
modules used: none
modules called: none
calling modules: HELP.TXT
files read: none
files written: HELP.SYS, BUILDDAT.OUT

procedure: make_keyboard (NOT IMPLEMENTED)
module number: 1.3
version: 1.0
date: 13 November 1984
description:
1.3 INSTALL KEYBOARD DEFINITION (MAKE_KEYBOARD)- (Not Implemented) This process is responsible for reading the Keyboard Definition file (KEYBOARD.TXT) and storing the information from the file in the "keyboard" array. The "keyboard" array is a global array of integers describing various keyboard control sequences used by the User Interface. The "keyboard" array is part of the information written to MICROSDW.SYS by BUILDDAT.

inputs: none
outputs: none
modules used: none
modules called: builddat
calling modules: none
files read: KEYBOARD.TXT
files written: Outfile

procedure: make_keyw_records
module number: 1.5.3
version: 1.0
date: 21 August 1984
description:  this procedure

1.5.5.3 MAKE_KEYW_RECORDS- This is a "Node-Left-Right" recursive procedure which traverses the menu tree (menu_tree) producing the decoding path record for each node in the tree. Duplicate processing of shared menus is eliminated by checking to see if the decoding path record for that node has already been created.

inputs:  menu_tree
outputs:                 

global variables used: data_recs.decode_dict.ptrs
global constants used: none
modules called: make_keyw_records
calling modules: make_decoding_paths
files read: none
files written: none

procedure: make_menu
module number: 1.5
version: 1.0
date: 16 August 1984
description:

1.5 INSTALL SOFTWARE CONFIGURATION (MAKE_MENU)- This process installs the menu system for the MICROSDW User Interface. The process reads the menu definition from the "menu.txt" file and creates the menu data structures stored in MICROSDW.SYS. First the menu data structures are initialized by init_menu, then the menu data structures are created by iteratively calling readmenu and addtomenu until the end of the MENU.TXT file is reached. The menu data structures are then converted to the data structures used to store the menu structure in the MICROSDW.SYS file.

inputs: none
outputs: none

global variables used: Outfile, Outflag
global constants used: ENDMENU, MERROR6
modules called: readmenu, add_keyword_to_menu,
make_word_list, make_decoding_paths,
displaytree, dispose_tree,
disposecalls, menu_print

calling modules: builddat
files read: menu.txt
files written: Outfile

procedure: make_printer
module number: 1.2
version: 2.0
date: 11 November 84
1.2 INSTALLPRINTER DEFINITION (MAKE_PRINTER)- This process is responsible for reading the Printer Definition file (PRINTER.TXT) and storing the information from the file in the "printr" array. The "printr" array is a global array of integers describing various printer control sequences used by the User Interface. The "printr" array is part of the information written to MICROSDW.SYS by BUILDDAT.

inputs: none
outputs: none
global variables used: data_recs.printr, Outflag
global constants used: printer_length
modules called: none
calling modules: builddat
files read: PRINT.TXT
files written: BUILDDAT.OUT

procedure: make_terminal
module number: 1.1
version: 1.0
date: 11 November 84
description:

1.1 INSTALLTERMINAL DEFINITION (MAKE_TERMINAL)- This process is responsible for reading the Terminal Definition file (TERMINAL.TXT) and storing the information from the file in the "term" array. The "term" array is a global array of integers describing various terminal control sequences used by the User Interface. The "term" array is part of the information written to MICROSDW.SYS by BUILDDAT.

inputs: none
outputs: none
global variables used: data_recs.term, Outflag
global constants used: term_length
modules called: none
calling modules: builddat
files read: TERM.TXT
files written: BUILDDAT.OUT

procedure: make_word_list
module number: 1.5.4
version: 1.0
date: 17 August 1984
description: this procedure

1.5.4 MAKE_WORD_LIST- This process creates a sorted list of the keywords and call-routine names used in the menustructure. First, the words
are sorted using the LNR procedure. The LNR procedure puts the words in ascending order into the "words" array, and assigns numbers matching the position of the word in the array to each node in the keyword tree. This number is used by make_decoding_paths to associate the appropriate "word number" with each decoding path record. The minimum number of characters necessary to identify each keyword are then calculated by the procedure calc_min_chars.

inputs: word_tree
outputs: global variables used: data_recs.decode_dict, Outfile, Outflag
global constants used: none
modules called: lnr, calc_min_chars
calling modules: make_menu
files read: none
files written: Outfile

procedure: menu_print
module number: 1.5.8
version: 1.1
date: 30 October 1984
description: 1.5.8 MENU_PRINT- This procedure traverses the menu tree (menu_tree) producing an indented listing of the menu options available in the menu structure.

inputs: menutree, level
outputs: printed menu structure
global variables used: Outflag, Outfile
global constants used: none
modules called: menu_print
calling modules: make_menu
files read: none
files written: Outfile

procedure: MICROSDW
module number: 0
version: 1.2
date: 30 October 1984
description: This is the root node of the Microcomputer Software Development Workbench (MICROSDW). The MICROSDW is composed of two programs, Build System Data (BUILDDAT) and MICROSDW User Interface (MSDW). BUILDDAT performs the "install" function for MSDW. The MICROSDW can be customized for a specific host computer by
modifying the text files describing the hardware or software and "installing" them with BUILDDAT. BUILDDAT creates three "system" files from the text files. The system files provide a description of the MICROSDW hardware and software environment to the User Interface. The User Interface provides an interface between the user and the software development tools in the MICROSDW environment.

inputs:
outputs:
global variables used: none
global constants used: none
modules called: none
calling modules: none
files read: none
files written: none

procedure: MICROSDW User Interface (MICROSDW)
module number: 2.0
version: 2.1
date: 30 October 1984
description:

2. MICROSDW User Interface (MICROSDW)- This program provides an interface between the user, the programs the MICROSDW is composed of, and the host operating system. The MICROSDW User Interface is composed of two main parts. One part is responsible for prompting the user and reading user inputs. The other part is responsible for executing functions built-into the User Interface and programs on the host microcomputer.

inputs:
outputs:
global variables used:
global constants used:
modules called: get-data, pause, select_routine
calling modules: none
files read: MICROSDW.SYS, HELP.SYS
files written:

procedure: number_calls
module number: 1.5.5.1
version: 1.0
date: 17 August 1984
description:

1.5.5.1 NUMBER_CALLS- This process assigns numbers to the nodes in the call list corresponding to their record number in the decoding paths.
procedure: printtree
module number: 1.5.6.1
version: 1.2
date: 30 October 1984
description: This procedure prints a binary tree.
inputs: node, depth, LR
outputs: global variables used: Outflag, Outfile
         global constants used: LEFT, RIGHT
modules called: printtree
calling modules: displaytree
files read: none
files written: Outfile

procedure: rchild
module number: 1.5.3.1.1.4
version: 1.0
date: 19/11/83
description: To return a pointer to the right child of a
             node in a binary tree. rchild receives a
             a pointer to a node as input, if that pointer
             is "nil" an error exists because there is not
             a right child of a "nil" node. In this case
             an error code is not returned but rchild is
             also set to "nil".
inputs: A pointer to the node of which a pointer to its
        rightchild is desired.
outputs: A pointer to the right node of a parent node.
global variables used: none.
global variables changed: none.
modules called: none.
calling modules: many
files read: none.
files written: none.

procedure: readmenu
module number: 1.5.2
version: 1.3
date: 21 October 1984
description: 1.5.2 READ MENU DEFINITION (READMENU)- This process reads
records from the MENU.TXT file and passes them into a buffer (parse_buf). The buffer is passed back to make_menu which then invokes addtomenue to build the word_tree, menu_tree, and call_list structures.

inputs: menu_text
outputs: parse_buf
global variables used: Outfile, Outflag
global constants used: COMMENT, ENDMENU, MENUDF, CALLDEF, ENDEF, MENUWORD
modules called: get_word
calling modules: make_menu
files read: menu_text
files written: Outfile

procedure: rightbalance
module number: 1.5.3.1.2.3
date: 20/11/83
version: 1.2
description: To perform balancing of trees unbalanced to the right.
inputs: A --> Pointer to last node with balance factor not equal to 0.
B --> Pointer to right child node of A.
outputs: A rebalanced binary tree.
global variables used: none.
global variables changed: none.
modules called: none.
calling modules: avlinsert

procedure: treedispose
module number: 1.5.7
date: 19/11/83
version: 1.1
description: treedispose "disposes" of a binary tree structure. If the tree is not empty treedispose traverses the tree structure in a "Right-Left-Node" (RLN) fashion calling the routine "dispose" to return leaf nodes to the system. Disposetree is a recursive procedure.
inputs: A pointer to the root of the binary tree to be disposed of.
outputs: An empty tree.
modules called: isempty, lchild, rchild, treedispose
calling modules: avl
files read: none.
files written: none.
Parameter/Variable Descriptions

Following are descriptions of the parameters and variables referenced on the structure charts in Appendix C.

NAME: A
TYPE: DATA ITEM
ALIASES: 
DESCRIPTION: This is a pointer into the binary keyword tree used by "avlinsert", "leftbalance", and "rightbalance" in height balancing a binary tree.
SOURCES: avlinsert
DESTINATIONS: leftbalance, rightbalance
COMPOSITION: 
PART OF: DATA
DATA CHARACTERISTICS: btpt (pointer to binary tree node)
VALUE RANGE: 
STORAGE TYPE: passed

NAME: B
TYPE: DATA ITEM
ALIASES: 
DESCRIPTION: This is a pointer into the binary keyword tree used by "avlinsert", "leftbalance", and "rightbalance" in height balancing a binary tree.
SOURCES: avlinsert
DESTINATIONS: leftbalance, rightbalance
COMPOSITION: 
PART OF: DATA
DATA CHARACTERISTICS: btpt (pointer to binary tree node)
VALUE RANGE: 
STORAGE TYPE: passed

NAME: call_list
TYPE: DATA ITEM
ALIASES: 
DESCRIPTION: This is a linked list of names of routines to call for different menu options. The list is composed of nodes which contain: a pointer to the call routine name stored in the binary wordtree, the number of the call routine, and a pointer to the next node in the list of call routines.
SOURCES: make_menu, addtomenu, define_call_routine, make_decoding_paths
DESTINATIONS: init_menu, addtomenu, make_decoding_paths, callsdispose, define_call_routine, findcall,
addcall, number_calls, make_call_records

COMPOSITION: callnode
PART OF: DATA
DATA CHARACTERISTICS: pointer
VALUE RANGE: passed
STORAGE TYPE: passed

NAME: call_pointer
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Call_pointer is a pointer to a node in the "call_list" representing a specified "callword".
SOURCES: define_call_routine
DESTINATIONS: findcall, addcall
COMPOSITION:
PART OF: DATA
DATA CHARACTERISTICS: callptr (pointer to call_list nodes)
VALUE RANGE: passed
STORAGE TYPE: passed

NAME: callword
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Callword is a word specified in the menu definition (MENU.TXT) representing a routine to be called by the User Interface.
SOURCES: define_call_routine
destinations: copymenuword
COMPOSITION:
PART OF: DATA
DATA CHARACTERISTICS: wordtype (string[WORDLENGTH])
VALUE RANGE: passed
STORAGE TYPE: passed

NAME: current_menu
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Current_menu points to the menu which menu keywords may be added to. Current_menu is set when a menu is defined in a MENU.TXT record (MENUDef). Current_menu is reset to "nil" when an ENDDEF record is encountered in MENU.TXT.
SOURCES: make_menu, addtomin
DESTINATIONS: addtomin, addkeywoartomenu, define_menu
COMPOSITION:
PART OF: DATA
DATA CHARACTERISTICS: pointer (mtptr) to a menu node (mtnode)
VALUE RANGE: passed
STORAGE TYPE: passed

D - 31
NAME: dataval
TYPE: DATA ITEM
ALIASES: dataval (function)
DESCRIPTION: This function is used in the same context as a
data item or parameter. Dataval returns a
keyword from the keyword tree.
SOURCES: btlocate, avlinsert, lnr
DESTINATIONS:
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: wordtype ( string[wordlength] )
VALUE RANGE:
STORAGE TYPE: function

NAME: decode_dictionary
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is the static data structure containing
the decoding paths for the keyword menu
structure, a linear array of keywords, and a
linear array of keyword abbreviations.
SOURCES: make_word_list
DESTINATIONS: calc_min_chars
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: dict_buffer
VALUE RANGE:
STORAGE TYPE: global, passed

NAME: depth
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This indicates a "depth" within the keyword
tree. Depth is used in formatting the printed
keyword tree structure.
SOURCES: displaytree, printtree
DESTINATIONS: printtree
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE:
STORAGE TYPE: passed

NAME: doneptr
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a pointer to the node containing
DONEWORD in the binary keyword tree.
SOURCES: make_call_records
DESTINATIONS: btlocate
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: btpt
VALUE RANGE:
STORAGE TYPE: passed

NAME: DONEWORD
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a special word stored in the keyword
  tree which is used to indicate termination of a
  keyword menu.
SOURCES: init_menu, make_call_records
DESTINATIONS: addword, btlocate
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: wordtype ( string[WORDLENGTH] )
VALUE RANGE: "$$$$$$$$$$
STORAGE TYPE: passed constant

NAME: error
TYPE: DATA ITEM
ALIASES: error_number
DESCRIPTION: Error is used to communicate the completion
  status of menu processing procedures.
SOURCES: make_menu, define_menu, define_callRoutine
DESTINATIONS: errormsg
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1..8
STORAGE TYPE: passed

NAME: found
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Found is a boolean flag indicating if a keyword
  is stored in the keyword tree.
SOURCES: addword, make_call_records
DESTINATIONS: btlocate
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: boolean, TRUE = keyword is in the
  word_tree, FALSE = keyword is not in
  word_tree.
VALUE RANGE: TRUE/FALSE
STORAGE TYPE: passed

NAME: isempty
TYPE: DATA ITEM
ALIASES: isempty (function)
DESCRIPTION: This is a boolean function used in the same context as a data item or parameter.
SOURCES: btlocate, avlinsert, lnr, displaytree, printtree, treedispose
DESTINATIONS:
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: boolean
VALUE RANGE: TRUE = binary tree is empty, FALSE = binary tree is not empty
STORAGE TYPE: function

NAME: keyword_pointer
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a pointer to a keyword node in the binary keyword tree. The pointer is either a pointer to an existing keyword or a newly created keyword node in the keyword tree.
SOURCES: addtomenu
DESTINATIONS: addword, addkeywordtomenu
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: pointer (btptr)
VALUE RANGE:
RELATED SADT ELEMENTS:
STORAGE TYPE: passed

NAME: lchild
TYPE: DATA ITEM
ALIASES: lchild (function)
DESCRIPTION: This function is used in the same context as a data item or parameter. Lchild returns a pointer to the left child (subtree) of a node in the binary keyword tree.
SOURCES: btlocate, avlinsert, lnr, displaytree, printtree, treedispose
DESTINATIONS:
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: btptr (pointer to a binary tree node)
VALUE RANGE:
STORAGE TYPE: function

NAME: level
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Level indicates the depth of the menu structure.
SOURCES: make_menu
DESTINATIONS: menuprint
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: positive integers
STORAGE TYPE: passed

NAME: LR
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: LR indicates to Printtree which side (left or right) of a binary tree node is currently being processed. LR affects the formatting the the binary tree for printing.
SOURCES: displaytree, printtree
DESTINATIONS: printtree
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1 = LEFT, 2 = RIGHT
STORAGE TYPE: passed

NAME: menu_path
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Menu_path is a list of keywords specifying a menu path. Each successive keyword is from a lower level in the menu hierarchy. Menu_path is copied from the parse_buf.wordstr by define_menu. Menu_path is passed to locate_menu_node which determines if the list of keywords in menu_path is a valid menu path.
SOURCES: define_menu, define_call_routine
DESTINATIONS: locate_menu_node
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: menuline (string[MAXMENU] )
VALUE RANGE:
STORAGE TYPE: passed

NAME: menu_position
TYPE: DATA ITEM
ALIASES: menu_node_pointer
DESCRIPTION: Menu_position is a pointer into the menu tree structure. Menu_position is calculated by locate_menu_node, it points to the last keyword specified in "menu_path".
SOURCES: define_menu, define_call_routine
DESTINATIONS: locate_menu_node
**COMPOSITION:**

**PART OF:**

**DATA CHARACTERISTICS:** mtptr (pointer to a menu tree node)

**VALUE RANGE:**

**STORAGE TYPE:** passed

**NAME:** menu_tree

**TYPE:** DATA ITEM

**ALIASES:** menutree

**DESCRIPTION:** Menu_tree is a dynamic data structure containing the MICROSDW menu structure. Keywords in the menu structure are stored in the "word_tree" and are referenced from the menu structure using pointers. There is a node in the menu structure for each keyword in each menu. The menu tree is not a true binary tree because two different menu nodes may point to the same submenu.

**SOURCES:** make_menu, addtomenu, make_decoding_paths

**DESTINATIONS:** init_menu, addtomenu, make_decoding_paths, menu_print, addkeywordtomenu, define_menu, define_call_routine, make_keyw_records

**COMPOSITION:** menunode

**PART OF:**

**DATA CHARACTERISTICS:** pointer

**VALUE RANGE:**

**STORAGE TYPE:** passed

**NAME:** newnode

**TYPE:** DATA ITEM

**ALIASES:**

**DESCRIPTION:** This is a pointer to a new node added to the binary keyword tree.

**SOURCES:** makebt

**DESTINATIONS:** avlinsert

**COMPOSITION:** wtnode

**PART OF:**

**DATA CHARACTERISTICS:** btptr (pointer to wtnode)

**VALUE RANGE:**

**STORAGE TYPE:** passed

**NAME:** node_counter

**TYPE:** DATA ITEM

**ALIASES:**

**DESCRIPTION:** Node_counter indicates the current number of nodes in the menu tree. Node_counter is used to assign "node_numbers" to menunodes in the menu tree.

**SOURCES:** make_menu, make_decoding_paths, addtomenu

**DESTINATIONS:** init_menu, addtomenu, make_decoding_paths,
number_calls, addkeywordtomenu

COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 0..NUM_PTRS
STORAGE TYPE: passed

NAME: number_of_words
TYPE: DATA ITEM
ALIASES: wordcount
DESCRIPTION: This is the number of words in the binary keyword tree plus one.
SOURCES: make_word_list, lnr
DESTINATIONS: lnr

COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE:
STORAGE TYPE: passed

NAME: parse_buf
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a record which contains parsed MENU.TXT lines.
SOURCES: make_menu, readmenu, addtomenu
DESTINATIONS: make_menu, addtomenu, define_menu, define_call_route

COMPOSITION: parserrecord
PART OF:
DATA CHARACTERISTICS: record
VALUE RANGE:
STORAGE TYPE: passed

NAME: rchild
TYPE: DATA ITEM
ALIASES: rchild (function)
DESCRIPTION: This function is used in the same context as a data item or parameter. Rchild returns a pointer to the right child (subtree) of a node in the binary keyword tree.
SOURCES: btlocate, avlinsert, lnr, displaytree, printtree, treedispose

DESTINATIONS:
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: btptr (pointer to a binary tree node)
VALUE RANGE:
STORAGE TYPE: function
NAME: side
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Side indicates where to link a newly created
keyword node into the binary keyword tree.
LEFT indicates as the left child, RIGHT
indicates as the right child, LPARENT indicates
that the new node is a parent node.
SOURCES:
avlinsert
DESTINATIONS: makebt
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: [LEFT, RIGHT, LPARENT]
STORAGE TYPE: passed

NAME: source
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a buffer containing a menu path.
SOURCES: locate_menu_node
DESTINATIONS: copymenuword
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: menuline (string[MAXMENU])
VALUE RANGE:
STORAGE TYPE: passed

NAME: start
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: The position in the "menu_path" to start
extracting a keyword from "menu_path".
SOURCES: locate_menu_node
DESTINATIONS: copymenuword
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1..MAXMENU
STORAGE TYPE: passed

NAME: start_position
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Start_position is the position in "menu_line"
to start looking for a keyword.
SOURCES: readmenu
DESTINATIONS: get_word
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1..MAXMENU
STORAGE TYPE: passed

NAME: stop
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: The position in the "menu_path" to stop
extracting a keyword from "menupath" at.
SOURCES: locate_menunode
DESTINATIONS: copymenuword
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1..MAXMENU
STORAGE TYPE: passed

NAME: stop_position
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: Stop_position is the length of word_line.
SOURCES: readmenu
DESTINATIONS: get_word
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: integer
VALUE RANGE: 1..MAXMENU
STORAGE TYPE: passed

NAME: word
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: A keyword.
SOURCES: addword
DESTINATIONS: btlocate, avlinsert
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: wordtype
VALUE RANGE:
STORAGE TYPE: passed

NAME: word
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a keyword extracted from a "menupath"
by copymenuword.
SOURCES: locate_menunode
DESTINATIONS: copymenuword
COMPOSITION:
PART OF:
DATA CHARACTERISTICS: wordtype (string[WORDLENGTH])
VALUE RANGE:
STORAGE TYPE: passed

NAME: word_line
TYPE: DATA ITEM
ALIASES: wordline
DESCRIPTION: Word_line is a line from the MENU.TXT file.
SOURCES: readmenu
DESTINATIONS: getword

COMPOSITION:

PART OF:
DATA CHARACTERISTICS: string[MAXMENU]
VALUE RANGE:
STORAGE TYPE: passed

NAME: wordpointer
TYPE: DATA ITEM
ALIASES:
DESCRIPTION: This is a pointer to a keyword node in the
binary keyword tree. The pointer is either a
pointer to an existing keyword or a newly
created keyword node in the keyword tree.
SOURCES: addword, define_call_routine
DESTINATIONS: init_menu, btlocate, avlinsert, findcall, addcall

COMPOSITION:

PART OF:
DATA CHARACTERISTICS: pointer (btptr)
VALUE RANGE:
STORAGE TYPE: passed

NAME: word_tree
TYPE: DATA ITEM
ALIASES: wordtree
DESCRIPTION: This is a binary tree data structure storing
the menu keywords. The tree is dynamically
created from the MENU.TXT file. Each unique
keyword appears only once in the tree.
SOURCES: make_menu, init_menu, addtomenu, addword,
btlocate, define_call_routine, make_word_list,
make_call_records, make_decoding_paths,
displaytree, printtree, treedispose
DESTINATIONS: init_menu, addtomenu, make_word_list,
make_call_records, make_decoding_paths,
displaytree, printtree, treedispose, addword,
define_call_routine, btlocate, avlinsert,
isempty, dataval, lchild, rchild, makebt, lnr

COMPOSITION: wtnode

D - 40
PART OF:
DATA CHARACTERISTICS: btptr (pointer to wtnode)
VALUE RANGE:
STORAGE TYPE: passed
**Alias Descriptions**

Following are descriptions of the aliases for parameters and variables listed in the previous Parameter/Variable Descriptions section.

**NAME:** error_number  
**TYPE:** ALIAS  
**DD TYPE:** PARAMETER (DATA ITEM)  
**SYNONYM:** error  
**ENVIRONMENT:**  
**WHERE USED:**  
**COMMENTS:** error_number is passed to the process errormsg by some processes.

**NAME:** menu_node_pointer  
**TYPE:** ALIAS  
**DD TYPE:** PARAMETER  
**SYNONYM:** menu_position  
**ENVIRONMENT:**  
**WHERE USED:** define_menu to locate_menu_node  
**COMMENTS:**

**NAME:** menutree  
**TYPE:** ALIAS  
**DD TYPE:** PARAMETER  
**SYNONYM:** menu_tree  
**ENVIRONMENT:** menutree is often used as the local name for the global menu_tree pointer which is passed to subprocesses.  
**WHERE USED:** many places  
**COMMENTS:**

**NAME:** wordcount  
**TYPE:** ALIAS  
**DD TYPE:** PARAMETER  
**SYNONYM:** number_of_words  
**ENVIRONMENT:** wordcount is the local name used in lnr for the number of words in the binary word_tree.  
**WHERE USED:** lnr  
**COMMENTS:**

**NAME:** wordline  
**TYPE:** ALIAS  
**DD TYPE:** PARAMETER  
**SYNONYM:** word_line  
**ENVIRONMENT:** wordline is just a shorter name for word_line  
**WHERE USED:**
NAME: wordtree
TYPE: ALIAS
DD TYPE: PARAMETER
SYNONYM: word_tree
ENVIRONMENT: wordtree is just a shorter name for word_tree, when word_tree is passed from process to process the shorter name is often used in the destination process.
WHERE USED: many places
COMMENTS:
APPENDIX E

Menu Language Definition
APPENDIX E

Menu Language Definition

Introduction

The purpose of this appendix is to document the language developed for specifying User Interface menu structures. The Install program (BUILDDAT) interprets the menu definition language and creates the static menu data structures stored in the MICROSDW.SYS file. The MICROSDW.SYS file is read by the User Interface to initialize the menu structure and other static data structures describing the hardware environment and help messages.

The text menu definition is stored in the file MENU.TXT. The MENU.TXT file is processed by the "make_menu" routines in BUILDDAT. The make_menu routines first read the text file, parsing the text lines as they are read from the file, convert the text menu structure to dynamic data structures, and then transform the dynamic data structures into static data structures. The static data structures are then stored in part of the MICROSDW.SYS file.

The text menu definition allows the specification of a hierarchical menu structure. The menu structure is similar to a binary tree except that a menu (list of keywords) may have more than one parent menu. This feature allows two or more keyword options to share the same submenu. Sharing of submenus reduces the amount of space necessary to store the menu structure. As discussed in Preliminary Design, space is
a critical resource in most microcomputer applications. Menu structures which do not have identical submenus will not be able to take advantage of shared submenus. The CONTROL-CAD system from which the MICROSDW User Interface descended made extensive use of this feature. The following section describes the menu definition language in terms of a menu definition grammar.

Menu Definition Grammar

\[
\text{menu} ::= \text{<menu definition> } \text{<menu word>} \mid \text{<menu word>} \mid \text{<end menu>} \mid \text{<call definition> } \text{<menu word>} = \text{<keyword>} \mid \text{<menu definition> } \text{<menu word>} = \text{<menu word>} \mid \text{<comment> } \text{<alpha>} \mid \text{<end definition>}
\]

\[
\text{<menu definition>} ::= "S"
\text{<menu word>} ::= \text{<keyword> [.<keyword>]}
\]

\[
\text{<call definition>} ::= ":."
\text{<end definition>} ::= ":."
\text{<comment>} ::= ":.;"
\text{<keyword>} ::= \text{<alpha>.[<alpha>]}
\text{<end menu>} ::= ":1"
\text{<alpha>} ::= \text{0|1|2|3|4|5|6|7|8|9} \text{<upper case>} | \text{<lower case>}
\text{<digit>} ::= \text{A|B|C|..X|Y|Z}
\text{<lower case>} ::= \text{a|b|c|..x|y|z}
\]

There are six types of input text lines: menu definition, call definition, menu word, comment, end of menu, and end of definition. There are two types of menu definitions. The first type of menu definition allows specification of a menu option followed by the keywords which make up the options under that menu option on following separate lines, terminated by and end of menu line. The
second type of menu definition allows assignment of a previously defined submenu to be the submenu of the specified menu option. The name of the procedure to call for a particular menu option is specified by a call definition line. Comment lines may occur anywhere in the text and are identified by a ';' in column 1.

Figure E-1 is a sample menu definition file. Figure E-2 shows the menu structure created from the menu definition in Figure E-1.
; Start of Example Menu Structure
; Define Root Menu
Setup
LifeCycle
Help
Exit
!
; Define Call routines for Options without submenus
Setup = Setup
Exit = STOP
!
; Define LifeCycle options
$LifeCycle
Require
Structure
PDL
Coding
Testing
Planning
!
; Define Call routines for LifeCycle options
$Lifecycle.Require = Require
$Lifecycle.Coding = Coding
$Lifecycle.Testing = Testing
$Lifecycle.Planning = Planning
!
; Define LifeCycle.PDL options
$LifeCycle.PDL
Pre-Fab
ProjA
ProjB
!
; Define Call Routines for LifeCycle.PDL
.LifeCycle.PDL.Pre-Fab = Design
.LifeCycle.PDL.ProjA = Design
.LifeCycle.PDL.ProjB = Design
!
; Define LifeCycle.Structure Options, share with PDL
$LifeCycle.Structure = LifeCycle.PDL
!
; Define Help Options
$Help
MICROSDW
LifeCycle
!
; Define Help Call Routine
.Help.MICROSDW = Help
.Help.LifeCycle = Help
;

Figure E-1: Menu Definition File Example
Figure E-2: Menu Structure Matching Figure E-1
APPENDIX F

Microcomputer Software Development Tools

And Environments Listing
APPENDIX F

Microcomputer Software Development Tools
And Environments Listing

Introduction

This appendix lists software development tools currently available for microcomputers. The list is not comprehensive as the number of software development tools for microcomputers is changing every day. The list is meant to be a "living" document which is updated as new software development tools for microcomputers are discovered.
<table>
<thead>
<tr>
<th>Tool or Environment</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Doctor</td>
<td>F - 4</td>
</tr>
<tr>
<td>Excelerator</td>
<td>F - 4</td>
</tr>
<tr>
<td>P-System</td>
<td>F - 5</td>
</tr>
<tr>
<td>Structured Program Designer</td>
<td>F - 6</td>
</tr>
<tr>
<td>STRUCTURE(S)/PC</td>
<td>F - 6</td>
</tr>
<tr>
<td>TURBO-Pascal</td>
<td>F - 7</td>
</tr>
<tr>
<td>TURBO TOOLBOX</td>
<td>F - 7</td>
</tr>
<tr>
<td>Utility Pack #1 and #2</td>
<td>F - 7</td>
</tr>
</tbody>
</table>
Name: Disk Doctor
Acronym: 
Description: A program to recover "crashed" disks.
Life-cycle Phase(s) Supported: 
Machines: 
Operating Systems: CP/M and CP/M-86
Offered By: 
SuperSoft, Inc.
P.O. Box 1628
Champaign, IL 61820
(217)359-2112
Cost (Approximate): $100

Name: Excelerator
Acronym: Excelerator
Description: Excelerator is an integrated product designed to support professional systems analysts. It provides the capabilities required to design and document software systems such as:
Graphics:
   Data Flow Diagrams
   Structure Charts
   Data Model Diagrams
   Presentation Graphs
Data Dictionary
Create Application "Screens" and Menus
Reports
Documentation: Hardcopy graphics printout of diagrams
   Data Dictionary Information
Wordprocessing
System Backup and Recovery

Life-cycle Phase(s) Supported: Requirements Specification, Preliminary Design, Detailed Design, Prototyping
Machines: IBM PC-XT, 256K RAM, MS-DOS 2.0 (or later), 10 Megabyte Harddisk
Operating Systems: MS-DOS 2.0 or later
Offered By: 
Index Technology Corporation (InTech)
Five Cambridge Center
Cambridge, MA 02142
Cost (Approximate): $9,500 including special hardware, graphics board, multifunction board (256K RAM, parallel port, calendar, clock), Mouse.
Name:  P-System
Acronym:  
Description:  Pascal, FORTRAN-77, or BASIC software development environment.

Universal Operating System With Text Processing:
  p-System
  Screen Editor, File Manager, Disc Recovery Tools Command Menu I/O Interface Turtle Graphics
  Printing Utilities Library Manager Configuration Tools File & Screen Management p-Machine Emulator Program Chaining

Integrated Languages:
  UCSD Pascal
  FORTRAN-77
  BASIC

Advanced Development Tool Kit:
  Symbolic Debugger Assembler & Linker
  Native Code Generator Program Analysis Tools

Additional Products:
  Insight Window Designer KSAM (Keyed Sequential Access Method)
  EDVANCE (editor) SofTeach (CAI for UCSD Pascal)
  XenoFile (Disk file Transfer)

Life-cycle Phase(s) Supported:  Detailed Design, Coding, Maintenance

Machines:  IBM PC, IBM PCjr, APPLE II, APPLE IIe, DEC Rainbow 100, Macintosh, etc.

Operating Systems:  CP/M, CP/M-86, MS-DOS, etc.

Offered By:
  SofTech Microsystems Inc.
  16875 W. Bernardo Drive
  San Diego, CA 92127

Cost (Approximate):  Depends on the configuration purchased. Educational Institution Discounts available.
Name: Structured Program Designer
Acronym: DEZIGN
Description: DEZIGN supports development of programs using
the Jackson Program Design Methodology. Features an
interactive editor for creating Data Structure Diagrams,
Program Structure Diagrams, and pseudo-code or
"Executable Operations". Provides a Librarian to manage
libraries of the above diagrams. Provides code
generators for Ada, C, Pascal, or PL/I.
Life-cycle Phase(s) Supported: Preliminary Design, Detailed
Design, Coding
Machines: Zenith Z89, Z90, Z100
Operating Systems: CP/M, CP/M-86, Z-DOS
Offered By:
ZEDUCOMP
P.O. Box 68
Stirling, NJ 07980
(201) 755-2262
Cost (Approximate):

Name: STRUCTURE(S)/PC
Acronym: STRUCTURE(S)/PC
Description: A data structured development workstation
available for the IBM PC.
Supports creation of:
- functional flow diagrams
- data structure diagrams (extension of Warnier/Orr)
- data base design charts
- system event diagrams
- system process structure charts
- program structure charts
- code generation (COBOL)
Life-cycle Phase(s) Supported: Requirements Definition,
Preliminary Design, Detailed Design, Coding
Machines: IBM PC/XT
Operating Systems: MS-DOS
Offered By:
Ken Orr & Associates, Inc.
1725 Gage Blvd.
Topeka, KS 66604-3379
800/255-2459, in Kansas 913/273-0653
Cost (Approximate): $1995 (1-5 copies), $1795 (6-10 copies)
Parts of the system may also be purchased separately.
Name: TURBO-Pascal
Acronym: TURBO-Pascal
Description: TURBO-Pascal provides an integrated compiler, editor, and run-time debugger.
Life-cycle Phase(s) Supported: Coding
Machines:
Operating Systems: CP/M, CP/M-86, MS-DOS
Offered By:
Borland International
4113 Scotts Valley Drive
Scotts Valley, CA 95066
Cost (Approximate): $50, discounts for orders of 5 or more copies.

Name: TURBO TOOLBOX
Acronym: TURBO TOOLBOX
Description: This tool box supplies 3 programming tools: 1) ISAM files using B+ Trees, 2) Quicksort on Disk, 3) GINST, a terminal installation program for programs developed in TURBO-Pascal.
Life-cycle Phase(s) Supported: Coding
Machines:
Operating Systems: CP/M, CP/M-86, MS-DOS
Offered By:
Borland International
4113 Scotts Valley Drive
Scotts Valley, CA 95066
Cost (Approximate): $49.95 + $5 shipping, discounts for orders of 5 or more copies.

Name: Utility Pack #1 and #2
Acronym:
Description: Two collections of general purpose routines to: Search lists of files for a string, Compare files, Erase files, Archive files, Sort files, Print files, etc
Life-cycle Phase(s) Supported:
Machines:
Operating Systems: CP/M
Offered By:
SuperSoft, Inc.
P.O. Box 1628
Champaign, IL 61820
(217) 359-2112
Cost (Approximate): $60 each
APPENDIX G

MICROSDW User's Manual
APPENDIX G
MICROSDW User's Manual

Introduction

This appendix contains a short User's Manual for the MICROSDW system. The current development provides an Installation program, BUILDDAT, and a User Interface program MICROSDW. BUILDDAT allows the user to configure the MICROSDW for a host microcomputer. MICROSDW provides the user with an interface to software development programs and to functions built into the MICROSDW. The User's Manual is divided into three sections, the first section describes how to use BUILDDAT, the second section describes how to use the MICROSDW, and the third section outlines the procedure for installing the MICROSDW on new host microcomputers.

BUILDDAT

BUILDDAT reads five text files describing the hardware environment (TERM.TXT, PRINT.TXT), help text (HELP.TXT), and the menu structure (MENU.TXT, PARAM.TXT). These text files must be present on the same disk as BUILDDAT for BUILDDAT to access them and execute correctly. BUILDDAT creates two system files, MICROSDW.SYS and HELP.SYS, and optionally a file containing a listing of what BUILDDAT did (BUILDDAT.OUT). The prompts the user receives are "yes" or "no" type questions, the default answer if the user just enters a "return" is NO. Yes/No prompts are terminated with
"(Y[N])", the "N" in brackets indicates the default if the user enters a "return". "Y"es or "N"o may be entered in either upper or lower case letters.

To execute BUILDDAT the user should first type "BUILDDAT" in response to the operating system prompt. BUILDDAT will then display an introduction message and begin prompting the user for inputs.

The user has the option to either recreate the entire MICROSDW.SYS and HELP.SYS files from the text files or to recreate the portion of the system files matching one of the text files. The user is first asked if he wants to rebuild the entire system file or selected portions of it. The user is then asked if he wants to spool the output from BUILDDAT to the file BUILDDAT.OUT.

If the user is rebuilding the entire system he will see information on the screen indicating which file is being processed. The next prompt the user will receive will ask whether he wishes to see/display the MENU structure. The next prompt the user will receive will ask whether he wishes to see/display the WORD structure.

If the user selects to rebuild portions of the system files he will then be prompted for which portions are to be rebuilt. The prompts appear in the following order:

- parameters PARAM.TXT
  - terminal TERM.TXT
  - printer PRINT.TXT
  - help messages HELP.TXT
  - menu structure MENU.TXT
If the user elects to rebuild a portion of the system file, BUILDDAT will rebuild the portion selected and then prompt for the next section to be rebuilt.

After the text file(s) have been processed by BUILDDAT the MICROSDW.SYS file is rewritten with the information updated from the text files. The HELP.SYS file is rewritten immediately after the HELP.TXT file is processed.

When the MICROSDW.SYS file has been rewritten BUILDDAT displays the message "MICROSDW installation complete" and terminates.

MICROSDW

The MICROSDW provides the user with an interface to various functions built into the MICROSDW and to programs residing on disk. The MICROSDW uses a hierarchical keyword command structure to prompt the user for inputs.

To execute the MICROSDW the user should first type: "MICROSDW" (without quotes) in response to the operating system prompt. The MICROSDW will then display the message: "Initializing the MICROSDW Menu structure and Hardware configuration". The MICROSDW takes about 1 minute to initialize. The MICROSDW will then display a "title slide", indicating that initialization is complete. The user is then prompted to enter a "carriage return" to continue.

A list of keywords is then displayed on the screen followed by the prompt: "Enter Option >"
The user may then enter one of the keywords displayed or an abbreviation for a keyword. The minimum abbreviation for each keyword is shown in bold. The user may either enter the entire keyword or an abbreviation for the keyword with at least as many characters as the minimum abbreviation. All characters entered for a keyword are validated against the currently acceptable keywords.

The user may enter more than one complete keyword or abbreviation after the "Enter Option >" prompt. Keywords and abbreviations must be separated by spaces and terminated with a carriage return. After a keyword or sequence of keywords is entered the MICROSDW checks them against the valid keywords. If the user entered an invalid keyword or abbreviation the invalid keyword is highlighted and the user is prompted to reenter a valid keyword. After a valid keyword has been entered the MICROSDW displays the keywords for the next lower level in the menu structure or executes the selected option. When then keywords are displayed for a lower level in the menu structure the previously entered keywords are displayed following "Enter Option >".

The user may "abort" a particular option while being prompted from the menu structure by entering a "$" and a "carriage return". This returns the user back to the top level menu.

If Problems Are Encountered. Following is a brief list of actions to take when problems are encountered during execution of the MICROSDW.
1. Errors during initialization - Make sure all files required by the MICROSDW reside on the same disk as the MICROSDW. Attempt to re-execute the program.

2. A "trashy" title slide appears - Attempt to re-execute the program.

   2.1. If the error occurs again and this is the first attempt to execute the MICROSDW on a new system or after installing a new terminal definition then: the terminal definition needs to be corrected (modify TERM.TXT, execute BUILDDAT, reinstall the terminal definition, copy new MICROSDW.SYS file to current disk).

   2.2. If the MICROSDW has previously been executed on the current system with no problems either the MICROSDW.SYS file has been damaged or the executable MICROSDW file has been damaged, restore the former files from a backup copy and try again.

3. If the problem is with erratic program operation try step 2 and then 2.1 or 2.2 if necessary. If the problem cannot be corrected document the problem: provide the commands used and the system response. Provide this information to the system developer for correction or attempt to modify the code and debug the problem yourself.

MICROSDW Installation

This section briefly describes a procedure for installing the MICROSDW on new host microcomputers. The installation procedure is described in the following list of steps:

1. Obtain a copy of the MICROSDW executable programs, text files, and system files for your host computer. If executable programs do not exist for your computer or the executable programs will not execute on your computer see the MICROSDW Maintenance Manual for instructions on how to create executable programs for your system.

2. Attempt to run the MICROSDW program as described in the above section. If the Initialization message appears and the "title slide" is "trashy" you need to
modify the TERM.TXT file to work with your terminal. If the Initialization message does not appear you must create executable programs for your system.

3. To install a new terminal definition follow the following steps:

3.1. Modify the TERM.TXT file to be compatible with your terminal. The most likely areas for problems are: cursor positioning and graphics characters. Your terminal must be capable of positioning the cursor to a given line and column using an "escape" sequence of characters or binary numbers.

3.2. Execute BUILDDAT.

3.3. Either rebuild the entire MICROSDW.SYS file or just the terminal definition portion. You may want try rebuilding the entire MICROSDW.SYS file first in case it was damaged in transporting it to your system.

3.4. Copy the MICROSDW.SYS file and HELP.SYS file to the same disk as the MICROSDW executable program. Try executing the MICROSDW again.
APPENDIX H

MICROSDM Programmer/Maintenance Manual

H - 1
APPENDIX H
MICROSDW Programmer/Maintenance Manual

Introduction

This appendix contains a short Maintenance manual which gives some insight into how to modify the Install program, BUILDDAT, and the User Interface program, MICROSDW. Interested individuals should also consult the TURBO-Pascal Reference Manual (42) for details about the TURBO-Pascal implementation of Pascal, and the operation of the TURBO-Pascal Compiler and Editor. The source code should also be consulted when modifying the MICROSDW or attempting to debug problems with BUILDDAT or the MICROSDW.

The current system implementation contains no overlays thus simplifying maintenance of the programs. Future expansion of the MICROSDW program or BUILDDAT may require use of overlay or chain files to conserve memory space. When this is considered, or attempted, the TURBO-Pascal Reference Manual should be consulted.

An installed version of the MICROSDW requires only the MICROSDW program and two system files, MICROSDW.SYS and HELP.SYS, for execution. Also, any programs executed by the MICROSDW should also be resident on the same disk as the MICROSDW to reduce the necessity for switching disks while executing the MICROSDW. The source files required for maintaining the MICROSDW and BUILDDAT programs reside on separate disks.
The MICROSDW and BUILDDAT programs are split up into separate source files. The source files for the MICROSDW and BUILDDAT programs are listed in Appendix I. A listing of the program source files and the procedures they contain is located at the beginning of Appendix I. The programs share two source files describing constants and data types used by both programs, MSDWCONS.PAS and MSDWTYPE.PAS respectively. If either of these files are modified both programs must be recompiled. If the programs are not compiled using the same version of these two files the MICROSDW will probably not be able to read the MICROSDW.SYS file correctly and thus will not initialize or run correctly.

Program Modifications

Maintenance of the MICROSDW and BUILDDAT programs is simplified by the features of TURBO-Pascal. The following steps may be used to modify BUILDDAT using TURBO-Pascal:

1. Execute TURBO-Pascal.
2. Change the "Logged Drive" to the disk drive containing the source files for BUILDDAT.
3. Select the "Option" to Compile the program to disk, the entire program cannot be compiled in memory on systems with 64K or less of addressable memory.
4. Specify BUILDDAT as the "Main" file.
5. Specify the source file to be modified as the "Work" file.
7. "Compile" BUILDDAT.
8. If errors in compilation occur:
   8.1. Edit the source file containing the error.
   8.2. Repeat from step 7.
9. "Run" BUILDDAT (If MS-DOS "Quit" first).
10. Check for errors, repeat from 6 if necessary.
11. "Save" the modified source file.
12. "Quit" TURBO-Pascal.
13. Run MICROSDW using the new MICROSDW.SYS and HELP.SYS files created by BUILDDAT.
14. Check for errors, repeat from 6 if necessary.

The procedure for modifying the MICROSDW program is the same as that for BUILDDAT, replace MICROSDW for BUILDDAT in the above steps and omit steps 13 and 14 (the MICROSDW.SYS and HELP.SYS files are not changed by the MICROSDW).

**Adding MICROSDW Built-in Functions**

The most likely area for further development of the MICROSDW is the addition of more built-in functions. Functions may be added to the MICROSDW using the following steps:

1. Identify the Function to be added.
2. Develop a subroutine to implement the function.
3. Modify the menu structure in MENU.TXT to include the new function.
   - 3.1. Add a Keyword for the new option to the appropriate menu(s).
   - 3.2. Define the name of the "Call Routine" for the Keyword in the modified menu(s). This is used in the procedure SELECT_ROUTINE to call the new function.
4. Run BUILDDAT to install the new menu structure.
5. Modify the procedure SELECT_ROUTINE (SELECT.PAS) in the MICROSDW following the steps for Program Modification above. Add the "Call Routine" name for the new function to the Case statement branching to different functions in SELECT_ROUTINE.
6. If the new function is not part of an existing source file, add an "include" statement for the new source file to MICROSDW.PAS (follow the Program Modification Steps above).
7. Run the MICROSDW, test the new function.

An alternative approach to installing a new function is to first create the new function and test it using a "Driver" program. Then use the above steps to install the new function in the MICROSDW program. This approach may lead to quicker
debugging of new functions as only the new function and the
driver program need to be compiled during debugging instead
of the entire MICROSDW program (2300+ lines of source code).

Another common change to the MICROSDW environment is the
addition, deletion, or modification of software development
programs (tools). If possible additional software
development programs should be implemented in TURBO-Pascal.
The programs can be created as TURBO-Pascal "Chain" files and
be chained to from the MICROSDW. Chain files are smaller
than program files and thus can load for execution faster
than program files. Using Chain files also has an advantage
over Overlay files in that they may be compiled separately
from the MICROSDW while Overlay files cannot. A program
which is Chained to by the MICROSDW should Chain back to the
MICROSDW when it completes. To add a program to the
environment use the following steps:

1. Modify the MENU.TXT file to include the new program
   as an option in the menu structure.
2. Specify the "Call Routine" for the program in the
   menu structure. This routine will execute the
   program. The "Call Routine" may be the generic "Run"
   procedure or a new procedure developed specifically
   for executing the new program.
3. Install the new menu structure using BUILDDAT.
4. Modify the SELECT_ROUTINE procedure in SELECT.PAS if
   necessary.
5. If possible copy the new program, or chain file to
   the MICROSDW system disk.
6. Execute the MICROSDW, test the new program.

To delete a program or function from the MICROSDW use
the following steps:

1. Delete references to the program/function from the
   menu structure in the MENU.TXT file.
2. Install the new menu structure using BUILDDAT.
3. If a function or a program with a special initiation procedure is being deleted then:
   3.1. Delete the appropriate "Call Routine" reference from SELECT_ROUTINE in SELECT.PAS (if it is not used by any other menu options).
   3.2. Delete the function (include statement) from the appropriate source file.
   3.3. Recompile the MICROSDW.
4. Delete the source files for the program or function.
5. Execute the MICROSDW and test the new menu structure.

Current System Limits

Limits to the current programs are specified in MSDWCONS.PAS, MSDWTYPE.PAS, MENCNS.PAS, and MENTYPET.PAS. Currently up to 75 unique keywords and up to 250 decoding path records are allowed in the menu structure. This is sufficient for a large keyword menu structure. Changes to the limits in the MSDWCONS.PAS and MSDWTYPE.PAS files require both the MICROSDW and BUILDDAT programs to be recompiled and the MICROSDW.SYS and HELP.SYS files to be completely rebuilt using the new version of BUILDDAT. Limits to the number of records in the TERM.TXT, PRINT.TXT, and the number of messages in the HELP.TXT file are also specified in the former files.
APPENDIX I

MICROSDW Program Listings
APPENDIX I
MICROSDW Program Listings

Introduction

This appendix contains the source code the MICROSDW Installation Program, BUILDDAT.COM, and User Interface, MICROSDW.COM. Listings for the text files which are read by BUILDDAT to create the MICROSDW system files MICROSDW.SYS and HELP.SYS. The text file listings are followed by the source code for BUILDDAT.COM and MICROSDW.COM respectively. The text files listed are:

- help.txt
- menu.txt
- param.txt
- print.txt
- term.txt

A cross index listing which procedures are contained in which source files follows.
Source File Contents

BUILDDAT Source Files:

BUILDDAT.PAS
  make_param
  make_terminal
  make_printer
  make_help
  builddat

ADDKEY.PAS
  addkeywordtomenu

ADDTOMEN.PAS
  addtomenu

ADDWORD.PAS
  addword

AVL-1.PAS
  create
  isempty
  lchild
  rchild
  dataval

AVL-2.PAS
  makebt
  LNR
  displaytree
    printtree
    treedispose

AVL-3.PAS
  leftbalance
  rightbalance

AVL-4.PAS
  avlinsert
  btlocate

DEFCALL.PAS
  findcall
  addcall
  define_call_routine

DEFMENU.PAS
  define_menu
BUILDDAT Source Files (continued):

DISPROC.PAS
  callsdispose
ERRORMSG.PAS
  errormsg
GETWORD.PAS
  get_word
LOCMENU.PAS
  copymenuword
  locate_menu_node
MAKEMENU.PAS
  init_menu
  make_menu
MAKEPROC.PAS
  calc_min_chars
  make_word_list
  number_calls
  make_call_records
  make_keyw_records
  make_decoding_paths
MENUPRNT.PAS
  menu_print
READMENU.PAS
  readmenu

MENUCONS.PAS  - Menu Constant Definition File
MENUTYPE.PAS  - Menu Type Definition File
MSDWCONS.PAS  - MICROSDW.SYS file constant defs
MSDWTYPE.PAS  - MICROSDW.SYS type definitions
MICROSDW User Interface Source Files:

MICROSDW.PAS
  microsdw

DISPLAYC.PAS
  displa_commandword

GETCOM.PAS
  get_cmd

GETDAT.PAS
  title_slide
  bld_stat_line
  get_data

GETINPUT.PAS
  get_inp

GETINT.PAS
  getchi
  del_1st_ch
  ck_chr
  out_int
  get_int

GETLINE.PAS
  get_line

GETSTRIN.PAS
  get_strng

HELP.PAS
  help

INSTRUCT.PAS
  instruction

MSG.PAS
  disp_line
  clear_msg
  disp_msg

OUTPUT.PAS
  out_string

PAUSE.PAS
  pause

PROCESER.PAS
  proces_error
MICROSDW User Interface Source Files (continued):

PROMPTCM.PAS
  prompt_cmd

PROMPTHE.PAS
  prompt_help

READCOM.PAS
  readcom

SELECT.PAS
  select_routine

TERMINAL.PAS
  graphics
  nographics
  highlight
  nohighlight
  gotoxy
  clear
  VideoLow
  SVideoLow
  VideoBold
  SVideoBold
  Rectangle

TRIM.PAS
  trim

UCASE.PAS
  ucase

VALNDEC.PAS
  check_word
  val_n_dec
FILE: help.txt

message 1
<$$>
message 2
<$$>
message 3
<$$> #4
<$$> #5

INVALID Keyword or Abbreviation

This is a valid command, the rest is extraneous!

<$$>
message 6
<$$>
message 7
<$$>
message 8
<$$>
message 9
<$$>
message 10
<$$>
message 11
<$$> #12
<$$> #13

Enter an <S> or <C>...or an <$$> to abort....

To exit, press <$$>; for more, press <CR>.
<$$>
message 14
<$$> #15

MICROSDW System Help Information

The MICROSDW provides an interface to software development tools
hosted on microcomputers, and access to functions built-into the
interface to allow the user access to the environment. Following is
a list of the options in the MICROSDW root menu:

Setup - allows the user to change default settings of the MICROSDW
LifeCycle - provides access to software development tools
    supporting different phases of the software LifeCycle
Wordproc - provides access to Wordprocessing programs
Commun - provides access to Communications programs
Database - provides access to Database manipulation programs
Files - provides access to File manipulation functions/programs
Run - allows the user to "Run" another program
MediaMas - provides access to the Media-Master programs for
    formatting and transferring files to/from various
disk formats.

<$> #16

COMMAND INPUT

The command input structure is made up of a hierarchy of menus
containing keywords (or command words). After a keyword is entered
it is validated against the valid keywords for the current menu. If
the keyword is valid the next lower menu of keywords is displayed or
a prompt from the selected function appears. If an invalid keyword is
entered the keyword is highlighted on the "Enter Option >" line and the
user is prompted to enter a valid keyword. If the user knows the valid
keywords at the next lower level menu for a keyword in the current menu
the user may "type ahead" keywords for lower levels in the menu structure.

Example: Files Delete
    Files Dir

    (Dir is an abbreviation for Directory)

    Abbreviations are generally the first two or three
letters of the keyword. Abbreviations are shown in bold for each command word. The Abbreviation shown in bold is the minimum number of characters necessary to identify a command word, additional characters of the command word may be entered. All characters entered for a command word will be validated against the valid command words.

If a mistake is noted prior to pressing the carriage return <CR>, just use the backspace or delete key to erase backwards to the error. Correct the error and retype the remainder of the command. If the program is prompting you for a command completion, you can only erase the characters internally back to the point that began that prompt.

At any point an input is expected from the user, (either command input or subroutine process) the user may abort the process and return to the command mode. If a separate program is being executed enter the "Exit" or "Abort" command for that program.

To abort a command enter '$' followed by a carriage return and you will be returned to the top MICROSDW menu.

Setup function is not implemented yet.

LifeCycle provides access to programs supporting various phases of the software Life-cycle. Phases supported:

Require - Requirements Definition (NO TOOLS included yet)
Structure - Structure Charting (NO TOOLS included yet)
PDL - Program Design Language (NO TOOLS included yet)
DataDict - Data Dictionary tool(s) (NO TOOLS included yet)
Coding - access to TURBO-Pascal
Testing - (NO TOOLS included yet)
Planning - Software Development Management (NO TOOLS included yet)
Help - (No Help available yet)

<$> $19
Wordproc(essing) provides access to Wordprocessing programs.
Wordprocessors accessible: (None available yet)

<$> $20
Communic(ations) provides access to Communications programs.
Communications programs available:

    Kermit -
    Modem7 -

<$> $21
Provides access to Database manipulation programs.
(NO PROGRAMS supported yet)

<$> $22
Files provides access to a number of file manipulation functions
and programs. File Options:

View - View a file on the display screen
Print - Print a file
Rename - Rename a file
Delete - Delete a file
Undelete - Undelete a previously deleted file or files.
    To Undelete a file the disk must NOT have been written
to after the file was deleted.
Restore - Same as Undelete
Directory - Provides a sorted listing of files on a disk.
LongDir - Same as Directory only file sizes are displayed.
Filesize - Displays the size of a specified file.
Space - Displays the remaining space on a disk.
Compare - Compares two files for differences.
Squeeze - "Squeezes" out repeated characters in files.
FILE: help.txt

UnSqueeze - "UnSqueezes" repeated characters in files.
Copy - Copies one file to another.

<$$> #23
Run allows the user to specify a program to run from the MICROSDW.
The user will be returned to the operating system after execution of the
program. Future enhancements will return the user to the MICROSDW.

<$$> #24
message 24
<$$>
message 25
<$$>
message 26
<$$>
message 27
<$$>
message 28
<$$>
message 29
<$$>
FILE: menu.txt

Setup
LifeCycle
Wordproc
MediaMas
Communic
Database
Files
Run
Help
Exit

; Define Call routines for Options without submenus
.Setup = Setup
.Wordproc = Wordproc
.Run = RUN
.Exit = STOP
; Define LifeCycle options
$LifeCycle
.Require
.Structure
.PDL
.DataDict
.Coding
.Testing
.Planning
.Help

; Define Call routines for LifeCycle options
.LifeCycle.Require = Require
.LifeCycle.Structure = Structure
.LifeCycle.PDL = PDL
.LifeCycle.Coding = Coding

FILE: menu.txt
FILE: menu.txt

.Lifecycle.Testing = Testing
.Lifecycle.Planning = Planning
.Lifecycle.Help = Help
; Define Lifecycle DataDictionary Options
$Lifecycle.DataDict
Pre-Fab
Project
; Define Call Routines for DataDict options
$Lifecycle.DataDict.Pre-Fab = DATADICT
$Lifecycle.DataDict.Project = DATADICT
; Define Media Master options
$MediaMas
Format
Transfer
; Define Media Master Call routines
.MediaMas.Format = MFORMAT
.MediaMas.Transfer = MMASTER
; Define Communications Options
$Communic
Kermit
Modem7
Run
; Define Communications Call Routines
.Communic.Kermit = RUN
.Communic.Modem7 = RUN
.Communic.Run = RUN
; Define Database Options
$Database
Create

FILE: menu.txt
FILE: menu.txt

Delete
Modify
Archive
List

; Define Database Call Routine
.Database.Create = DATABASE
.Database.Delete = DATABASE
.Database.Modify = DATABASE
.Database.Archive = DATABASE
.Database.List = DATABASE
; Define Files Options
$Files
View
Print
Rename
Delete
Undelete
Restore
Directory
LongDir
Filesize
Space
Compare
Squeeze
UnSqueeze
Copy

; Define Call Routines For Files Options
.Files.View = FILES
.Files.Print = FILES
.Files.Rename = FILES
FILE: menu.txt

.Files.Delete = FILES
.Files.Undelete = FILES
.Files.Restore = FILES
.Files.Directory = FILES
.Files.LongDir = FILES
.Files.Filesize = FILES
.Files.Space = FILES
.Files.Compare = RUN
.Files.Squeeze = RUN
.Files.UnSqueeze = RUN
.Files.Copy = FILES
;
Define Help Options

$Help

MICROSDW
Setup
LifeCycle
Wordproc
Communic
Database
Files
Run
MediaMas

;
Define Help Call Routine

.Help.MICROSDW = Help
.Help.Setup = Help
.Help.LifeCycle = Help
.Help.Wordproc = Help
.Help.Communic = Help
.Help.Database = Help
.Help.Files = Help
.Help.Run = Help
FILE: param.txt

1 1 Printer = false, Printer disabled
2 1 Trans = false, Transaction file disabled
3 1 Temp = false, Temporary file output disabled
4 0 Crt = true, Crt output enabled
5 1 show_abbreviation = false, not used
6 0 in_terminal = true, input from terminal enabled
7 0 stat_on = true, display status line enabled
8 1 macro_error = false
9 1 not used
10 1 not used
1 3 help_level
2 1 not used
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
1 1.0
2 -3.0
3 5.0
4 -1.0
5 0.0001
6 0.1
7 0.0
8 0.0
9 0.0
10 0.0
1 LST: ; List device name
2 TRANSACTION ; Transaction File name

FILE: param.txt
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>MACRO.INP</td>
</tr>
<tr>
<td>4</td>
<td>STRING 4</td>
</tr>
<tr>
<td>5</td>
<td>STRING 5</td>
</tr>
<tr>
<td>6</td>
<td>STRING 6</td>
</tr>
<tr>
<td>7</td>
<td>STRING 7</td>
</tr>
<tr>
<td>8</td>
<td>STRING 8</td>
</tr>
<tr>
<td>9</td>
<td>STRING 9</td>
</tr>
<tr>
<td>10</td>
<td>STRING 10</td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>1-2</td>
<td>Cursor positioning, initial char seq.</td>
</tr>
<tr>
<td>3-9</td>
<td>ESC 91 [ ] 0 0 0 0 7 1 Row offset 8 1 Column offset 9 0 10 1 intermediate character(s) 11 59 ; 12 0 13 0 14 1 terminating character(s) 15 102 f 16 0 17 0 18 0 19 6 Clear screen, Home cursor 20 27 ESC 21 91 [ 22 102 f 23 27 ESC 24 91 [ 25 74 J 26 0 27 4 Highlight (Reverse Video) 28 27 ESC 29 91 [ 30 55 7 31 109 m 32 0</td>
</tr>
</tbody>
</table>

FILE: term.txt

Page: 1
FILE: builddat.pas

[A-]  { enable recursion (CPM/80) }

**************

* file: builddat.pas
* version: 2.0
* date: 28 November 84
* description: This file contains the main program
*              BUILDDAT of the MICROSDW system.
* procedures contained: make_param, make_terminal,
*                       make_printer, make_help, builddat
* author: Paul A. Moore, Capt, USAF

**************

**************

* procedure: builddat
* module number: 1
* version: 2.1
* date: 28 November 84
* history: 9/28/84 created
* description: This procedure is the main module for the
*              BUILDDAT program.
* This program performs the
* "install" operations for the MICROSDW. BUILDDAT reads ASCII
* text files describing the host microcomputer hardware and
* menu system, validates the files and creates the MICROSDW
* system files (MICROSDW.SYS, ERROR.SYS, and HELP.SYS). The
* system files are used by the MICROSDW User Interface to
FILE: builddat.pas

* interpret user inputs, display prompts, print information, *
* and to access "help" or "error" information.
* inputs:
* outputs:
* global variables used: data_file, data_recs, i,
  resp, resp2, Outfile, Outflag
* global constants used: (See MSDWCONS.PAS, MENUCONS.PAS)
* modules called: make_param, make_terminal,
  make_printer, make_help, make_menu
* calling modules: none
* files read: PARAM.TXT, TERM.TXT, PRINT.TXT
* HELP.TXT, MENU.TXT
* files written: MICROSDW.SYS, HELP.SYS, BUILDDAT.OUT
* author: Paul A. Moore, Capt, USAF
*
******************************************************************************

program builddat;

{$I MSDWCONS.PAS} (* include MICROSDW constants *)
{$I MENUCONS.PAS}
{$I MSDWTYPE.PAS} (* include MICROSDW type definitions *)
{$I MENUTYPE.PAS}

type menuline = string[MAXMENU];

var data_file : file of data;
data_recs : data;
i : integer;
resp : char;
resp2 : char;
Outfile : Text; (* file to spool Builddat output to *)
FILE: builddat.pas

Outflag : boolean; (* true = spool output, false=don't *)

(* -------------- include builddat subroutines ---------------*)
{$I errmsg.pas}
{$I avl-1.pas}
{$I avl-2.pas}
{$I avl-3.pas}
{$I avl-4.pas}

{$I getword.pas}
{$I readmenu.pas}

{$I menuprnt.pas}

{$I addword.pas}
{$I addkey.pas}
{$I locmenu.pas}
{$I defmenu.pas}
{$I defcall.pas}
{$I addtomen.pas}
{$I makeproc.pas}
{$I disproc.pas}

{$I makemenu.pas}

[PA]
******************************************************************************
*                        *
*  procedure: make_param    *
*  module number:          *
*  version:    2.0          *
*  date:        28 November 84 *
******************************************************************************

FILE: builddat.pas
FILE: builddat.pas

* history: 9/28/84 created
* description: This procedure reads the PARAM.TXT file
  containing the parameters for the MICROSDW
  User Interface. The parameters are
  stored in data_recs.param for writing to
  MICROSDW.SYS
* inputs: none
* outputs: none
* global variables used: data_recs.param
* global constants used: num_param_group, num_bools,
  num_ints, num_reals, num_strings
* modules called: none
* calling modules: builddat
* files read: PARAM.TXT
* files written: BUILDDAT.OUT
* author: Paul A. Moore, Capt, USAF

**************************************************************************

procedure make_param;
var param_text : text;
line_number : integer;
i : integer;
j : integer;
tru_false : integer;
value : boolean;
tempstr : string[80];

procedure removeblanks(var instring : paramstring);
var bpos : integer;
begin
repeat
FILE: build.dat.pas

bpos := Pos(instring, ' ');
if bpos <> 0 then  { remove the blank }
  Delete(instring, bpos, 1);
until bpos = 0;
end;

begin
  writeln(chr(ff), 'Generating Parameters...');
  if Outflag then writeln(Outfile, chr(ff), 'Generating Parameters...');
  assign( param_text, 'param.txt' );
  reset( param_text );

  for i := 1 to num_param_group do
    begin
      if Outflag then
        writeln(Outfile, '  Generating parameter group ', i );
      writeln('  Generating parameter group ', i );

      (* ---------------- Do Booleans -------------------------- *)
      for j := 1 to num_bools do
        begin
          [$I-] readln( param_text, line_number, tru_false, tempstr );
          [$I+]
          if I0result <> 0 then
            begin
              writeln('Error reading boolean #', j:0);
              if Outflag then
                writeln(Outfile, 'Error reading boolean #', j:0);
            end
          else
            begin
              value := (tru_false = 0);
FILE: builddat.pas

data_recs.param[i].bools[j] := value;
if Outflag then
  writeln(Outfile,' ', line_number:4, value:6, tempstr );
end;
end;
if Outflag then writeln(Outfile);

(* ------------------------ Do Integers ------------------------ *)
for j := 1 to num_ints do
begin
  {$I-}
  readln( param_text, line_number,
  data_recs.param[i].ints[j] );
  {$I+}
  if IOresult <> 0 then
    begin
      writeln('Error reading integer #',j:0);
      if Outflag then
        writeln(Outfile,'Error reading integer #',j:0);
    end
  else
    if Outflag then
      writeln(Outfile,' ', line_number:4,
      data_recs.param[i].ints[j]:6,tempstr );
  end;
if Outflag then writeln(Outfile);

(* ------------------------ Do Reals ------------------------ *)
for j := 1 to num_reals do
begin
FILE: builddat.pas

{$I-}
readln( param_text, line_number,
data_recs.param[i].reals[j], tempstr );

{$I+}
if IOresult <> 0 then
begin
  writeln('Error reading real #',j:0);
  if Outflag then
    writeln(Outfile,'Error reading reals #',j:0);
  end
else if Outflag then
  writeln(Outfile,
           ',line_number:4,
data_recs.param[i].reals[j]:6,tempstr );
end;

if Outflag then writeln(Outfile);

(* ------------------------- Do Strings ------------------------- *)
for j := 1 to num_strings do
begin
  {$I-}
  readln( param_text, line_number,
data_recs.param[i].strings[j], tempstr );
  {$I+}
  removeblanks(data_recs.param[i].strings[j]);
  if IOresult <> 0 then
    begin
      writeln('Error reading string #',j:0);
      if Outflag then
        writeln(Outfile,'Error reading string #',j:0);
      end

FILE: builddat.pas
FILE: builddat.pas

    else if Outflag then
        writeln(Outfile,' ', line_number:4,
            data_recs.param[ i ].strings[ j ].tempstr );
    end;
    
end;

[..PA]
******************************************************************************
* *
*    procedure:    make_terminal
*    module number: 1.1
*    version:       1.0
*    date:          11 November 84
*    history:       9/28/84 created
*    description:
*  1.1 INSTALL TERMINAL DEFINITION (MAKE_TERMINAL)- This *
* process is responsible for reading the Terminal Definition *
* file (TERMINAL.TXT) and storing the information from the file *
* in the "term" array. The "term" array is a global array of *
* integers describing various terminal control sequences used *
* by the User Interface. The "term" array is part of the *
* information written to MICROSDW.SYS by BUILDDAT. *
* *
*    inputs: none
*    outputs: none
*    global variables used: data_recs.term, Outflag
*    global constants used: term_length
*    modules called: none
*    calling modules: builddat

FILE: builddat.pas
FILE: builddat.pas

* files read: TERM.TXT *
* files written: BUILDDAT.OUT *
* author: Paul A. Moore, Capt, USAF *

procedure make_terminal;

var i : integer;
line_number : integer;
term_dad : text;
tempstr : string[80];

begin
  writeln('Generating terminal control data...');
  if Outflag then
    writeln(Outfile,chr(ff), 'Generating terminal control data...');

  assign( term_dat, 'term.txt' );
  reset( term_dat );

  for i := 1 to term_length do
  begin
    readln( term_dat, line_number, data_recs.term[i], tempstr );
    if IOresult <> 0 then
      begin
        writeln('Error reading line #',i:0);
        if Outflag then writeln(Outfile,'Error reading line #',i:0);
      end
    else if Outflag then
      writeln(Outfile, ' ', line_number:4, data_recs.term[i]:5, tempstr );
FILE: builddat.pas

end;
end;

[..PA]

(***********************************************************************
*procedure: make_printer
*module number: 1.2
*version: 2.0
*date: 11 November 84
*history: 9/28/84 created
*description:
*1.2 INSTALL PRINTER DEFINITION (MAKE_PRINTER)- This process
*is responsible for reading the Printer Definition file
*(PRINTER.TXT) and storing the information from the file in
*the "printr" array. The "printr" array is a global array of
*integers describing various printer control sequences used by
*the User Interface. The "printr" array is part of the
*information written to MICROSDW.SYS by BUILDDAT.
*
*inputs: none
*outputs: none
*global variables used: data_recs, printr, Outflag
*global constants used: printer_length
*modules called: none
*calling modules: none
*files read: PRINT.TXT
*files written: BUILDDAT.OUT
*author: Paul A. Moore, Capt, USAF
*)

**********************************************************************************
FILE: builddat.pas
procedure make_printer;

var
  i                      : integer;
  line_number            : integer;
  print_dat              : text;
  tempstr                : string[80];

begin
  writeln('Generating printer control data...');
  if Outflag then
    writeln(Outfile, chr(ff), ' Generating printer control data...');

  assign( print_dat, 'print.txt' );
  reset( print_dat );

  for i := 1 to printer_length do
    begin
      {S1-}
      readln( print_dat, line_number, data_recs.printr[i], tempstr );{S1+}
      if IOresult <> 0 then
        begin
          writeln('Error reading line #',i:0);
          if Outflag then writeln(Outfile,'Error reading line #',i:0);
        end;
        if Outflag then
          writeln(Outfile, ' , line_number:4,data_recs.printr[i]:5,tempstr);
      end;
      {.PA}
end;

FILE:  builddat.pas
FILE: bulldat.pas

(*-------------------------------------------------------------
  procedure: make_help
  module number: 1.6
  version: 2.0
  date: 11 November 84
  history: 9/28/84 created
  description:
  1.6 MAKE_HELP- This procedure processes the HELP.TXT file
  creating the help index (msg_dir) stored in the MICROSDW.SYS
  file and the HELP.SYS file containing the text for the help
  messages.
  inputs: none
  outputs: none
  global variables used: data_recs, msg_dir, Outflag
  global constants used: none
  modules called: none
  calling modules: none
  files read: HELP.TXT
  files written: HELP.SYS, BUILDDAT.OUT
  author: Paul A. Moore, Capt, USAF
  *
  ******************************************************)

procedure make_help;

  type
    msg_buff = string[ screen_width ];

  var
    i        : integer;
    msg_num  : integer;
    msg_rec_num : integer;

FILE: bulldat.pas
FILE: builddat.pas

msg_length : integer;
msg_text : text;
one_line : string[ 100 ];
msg_file : file of msg_buff;
line_length : integer;
msg_data : msg_buff;
next_message,
end_of_messages : boolean;

begin
  writeln('Generating the message data...');
  if Outflag then
    writeln(Outfile, chr(ff), ' Generating the message data...');
  writeln;

  assign( msg_text, 'help.txt' );
  reset( msg_text );

  assign( msg_file, 'help.sys' );
  rewrite( msg_file );

  msg_num := 1;
  msg_rec_num := 1;
  end_of_messages := false;

repeat
  begin
    data_recs.msg_dir[ msg_num ].loc_rec := msg_rec_num;
    msg_length := 0;
    if Outflag then writeln(Outfile, 'Message number... ', msg_num);
    next_message := false;

FILE: builddat.pas
FILE: builddat.pas

repeat
  begin
    one_line := '';
    readln( msg_text, one_line );
    next_message := (Pos(' $', one_line) = 1);
    end_of_messages := (Pos(' $$', one_line) = 1);

    if not(next_message or end_of_messages) then
      begin
        line_length := length( one_line );
        if line_length > screen_width then line_length := screen_width;
        msg_data := copy( one_line, 1, line_length );
        write(msg_file,msg_data);
        if Outflag then writeln(Outfile, msg_rec_num:4, ' ',msg_data );
        msg_length := msg_length + 1;
        msg_rec_num := msg_rec_num + 1;
      end;
    end;
  until next_message or end_of_messages;

  data_recs.msg_dir[msg_num].length := msg_length;
  msg_num := msg_num + 1;
end;
  until end_of_messages;

writeln( 'This is the end of the messages.' );
close( msg_file );

if Outflag then
  begin
    write(Outfile, chr(ff), 'Message directory table...');
FILE: builddat.pas

   writeln(Outfile,'  Msg #  beg rec #    length');
   for i := 1 to (msg_num - 1) do
      writeln(Outfile, i:8, data_rec.msg_dir[i].loc_rec:12,
               data_rec.msg_dir[i].length:11);
   end;
end;

(* ------------------------------------------------------- *)

begin  (* the main program, builddat !!!!!!!! *)
   for i := 1 to 5 do
      writeln;
      writeln('Welcome to BUILDDAT! BUILDDAT is the installation program for ');
      writeln('the MICROSDW User Interface. BUILDDAT creates the MICROSDW.SYS ');
      writeln('file from text install files describing the host system hardware ');
      writeln('and the MICROSDW menu structure. BUILDDAT can either rebuild the ');
      writeln('entire MICROSDW.SYS file or selected portions of it.');
      writeln;
      writeln('If you want to re-create the entire system file, enter an "R" ');
      writeln;
      writeln('Or if you want to do each section selectively, enter an "S" ');
      writeln;
      write('                                                                Your choice -----> ');
      readln( resp );
      resp := UpCase(resp);
      writeln;

FILE: builddat.pas
FILE: builddat.pas

writeln('For the following questions "(Y[N])" indicates that "Y" means');
writeln('"Yes" and "N" means "No", the default answer if you enter <CR>;');
writeln('is "[N]" or "No".');
writeln;
write('Do you want to spool output to the file "BUILDOUT" (Y[N])?');
readln( resp2 );
writeln;
Outflag := (UpCase(resp2) = 'Y');
if Outflag then
  begin
    Assign(Outfile,'BUILDOUT');
    rewrite(Outfile);
  end;

assign( data_file, 'MICROSDW.SYS' );

if resp = 'R' then
  begin
    rewrite( data_file );
    make_param;
    make_terminal;
    make_printer;
    make_help;
    make_menu;
    writeln;
    writeln('Writing new system data file.');
    write( data_file, data_recs );
    close( data_file );
  end;
FILE: builddat.pas

if resp = 'S' then
begin
    reset( data_file );
    read( data_file, data_recs );

    write( 'Do you want to install new parameters (Y[N])? ' );
    readln( resp );
    if (UpperCase(resp) = 'Y') then make_param;
    writeln;

    write( 'Do you want to install new TERMINAL parameters (Y[N])? ' );
    readln( resp );
    if (UpperCase(resp) = 'Y') then make_terminal;
    writeln;

    write( 'Do you want to install new PRINTER parameters (Y[N])? ' );
    readln( resp );
    if (UpperCase(resp) = 'Y') then make_printer;
    writeln;

    write( 'Do you want to install new HELP messages (Y[N])? ' );
    readln( resp );
    if (UpperCase(resp) = 'Y') then make_help;
    writeln;

    write( 'Do you want to install a new MENU structure (Y[N])? ' );
    readln( resp );
    if (UpperCase(resp) = 'Y') then make_menu;

    (* reset data_file to beginning of the file *)
    reset( data_file );

FILE: builddat.pas
FILE:   builddat.pas

     writeln;
     writeln('Updating system data file.');
     write( data_file, data_recs );
     close( data_file );
     end;

if Outflag then close(Outfile);

     writeln;
     writeln('MICROSDW installation complete.');
     end.
FILE: addkey.pas

(*---------------------------------------------*
* file: addkey.pas *
* version: 1.1 *
* date: 3 September 84 *
* description: this file contains procedures which *
* are part of the MICROSDW BUILDDAT *
* program. *
* language: TURBO-Pascal *
* procedures contained: addkeywordtomenus *
* author: Paul A. Moore, Capt, USAF *
* *
*-----------------------------------------------*)

(*---------------------------------------------*
* procedure: addkeywordtomenus *
* module number: 1.5.3.2 *
* version: 1.1 *
* date: 3 September 84 *
* history: 8/16/84 created *
* 9/3/84 fixed "new" problem, *
* fixed linking problem *
* description: *
* 1.5.3.2. ADDKEYWORDEOMENU- This process adds a keyword *
* to the "current-menu". A node for the new menu keyword is *
* created and added to the current_menu. *
* inputs: word_pointer, current_menu, menutree, *
* node_counter *
* outputs: current_menu, menutree *
* global variables used: none *
*-----------------------------------------------*)
procedure addkeywordtomenu ( word_pointer : bt.ptr;
    var current_menu : mt.ptr;
    var menutree : mt.ptr;
    node_counter : integer);

var
    menunode : mt.ptr;
    tempmenu : mt.ptr;
begin
    (* add keyword to the end of the current menu list *)
    (* allocate a menu node for the keyword *)
    new(menunode);
    menunode^.wordptr := word_pointer;  (* initialize the node *)
    menunode^.nomatchp := nil;
    menunode^.matchp := nil;
    menunode^.endptr := nil;
    menunode^.node_number := node_counter;

    (* check for a completely empty menu structure *)
    if menutree = nil then
        menutree := menunode
    else
        if current_menu = nil then (* then adding a word to the root menu *)
            (* then adding a word to the *)
            (* root menu *)
FILE: addkey.pas

begin
  tempmenu := menutree;
  while tempmenu^.nomatchp <> nil do (* find end of *)
    tempmenu := tempmenu^.nomatchp; (* root menu *)
    tempmenu^.nomatchp := menunode; (* add the word *)
  end;
else
  begin
    if current_menu^.matchp = nil then (* first word *)
      current_menu^.matchp := menunode (* in the menu *)
    else
      begin (* locate end of the menu *)
        tempmenu := current_menu^.matchp;
        while tempmenu^.nomatchp <> nil do
          tempmenu := tempmenu^.nomatchp; (* link in the new menu word *)
        tempmenu^.nomatchp := menunode;
      end;
  end;

(* an alternate method of inserting the menu node would be to *
 link the menunode to the beginning of the nomatchp list, it *
 involves less code. But reverses the order of menu options *
 from the way they are entered in the file. *)
menunode^.nomatchp := current_menu^.matchp;
current_menu^.matchp := menunode;

end;
FILE: addword.pas

(*file: addword.pas
version: 1.0
date: 28 September 84
description: this file contains procedures which
are part of the MICROSDW BUILLDAT
program.
language: TURBO-Pascal
procedures contained: addword
author: Paul A. Moore, Capt, USAF
*)

(**********************************************************************)

(*procedure: addword
module number: 1.5.3.1
version: 1.0
date: 16 August 1984
history: 8/16/84 created
description:
1.5.3.1 ADDWORD- This process adds keywords to the*
word_tree. It calls blocate to determine if the keyword is*
already stored in word_tree. Addword returns a pointer to*
the location of the keyword in word_tree.
inputs: word, wordtree, wordpointer
outputs: wordtree, wordpointer
global variables used: none
global constants used: none
modules called: blocate, avlinsert
*)
procedure addword(word : wordtype; var wordtree : btptr;
    var wordpointer : btptr);
var
    found : boolean;
begin
    btlocate(word, wordtree, wordpointer, found);
    if not found then
        avlinsert(word, wordtree, wordpointer);
end;
FILE: avl-1.pas

{,PL60}
(***********************************************************************
* FILE NAME: avl-1.pas
* VERSION DATE: 9/20/84
* VERSION NUMBER: 2.0
* DESCRIPTION: This file is part of a system of routines created
* to manipulate binary tree data structures. The
* routines are contained in four files, AVL-{1,2,3,4}.PAS.*
* Routines are included to balance a binary tree as
* information is added to it.
* *
* FILE PROCEDURE
* AVL-1 create(root:btptr)
* AVL-1 isempty(root:btptr)
* AVL-1 lchild (root:btptr)
* AVL-1 rchild (root:btptr)
* AVL-1 dataval(root:btptr)
* AVL-2 LNR (P : btptr)
* AVL-2 displaytree (p : btptr )
* AVL-2 treedispose (p : btptr )
* AVL-3 addtotre (value : integer;var root : btptr)
* AVL-3 leftbalance (A, B : btptr )
* AVL-3 rightbalance(A, B : btptr )
* AVL-4 avlinsert (element : integer; var position : btptr )
* AVL-4 btlocate ( word : wordtype ; btree : btptr;
* var node_ptr : btptr; var found : boolean);
* *
* FILES WRITTEN: Outfile
* AUTHOR: Capt Paul A. Moore, APIT GCS-84D.
* HISTORY: Written for EE 5.86 Information Structures Course
* Fall Quarter 1983 for Dr. Lamont.

FILE: avl-1.pas
FILE: avl-1.pas

* 8/15/84 Converted to a library module for use with * 
  the MICRSDW make_menu routines * 
* 9/20/84 Converted to TURBO-Pascal, split into files * 
  AVL-1.PAS, AVL-2.PAS, AVL-3.PAS, AVL-4.PAS * 
* 
  INSTALLATION: DEC Rainbow 100, CP/M-80 * 
******************************************************************************************

const
  LEFT  = 1;  (* Insert new node to left of parent *)
  RIGHT = 2;  (* Insert new node to right of parent *)
  LPARENT = 3;  (* New node becomes the parent *)

{.PA}
******************************************************************************************

(* 
* PROCEDURE NAME: create 
* MODULE NUMBER: 
* VERSION DATE: 19/11/83 
* VERSION NUMBER: 1.0 
* 
* FUNCTION: To create a "nil" pointer to the root node of a 
* binary tree. The root of a binary tree does not 
* contain any information but points to the root 
* of a binary tree. 
* 
* INPUTS: none. 
* OUTPUTS: A pointer to the root node of a binary tree. 
* GLOBAL VARIABLES USED: none. 
* GLOBAL VARIABLES CHANGED: none. 
* MODULES CALLED: none. 
* CALLING MODULES: 
* FILES READ: none. 

FILE: avl-1.pas
procedure create(var tree : btptr);
begin
  tree := nil;
end;

function isempty(root : btptr) : boolean;
begin

FILE: avl-1.pas Page: 3
FILE: avl-1.pas

if root = nil then isempty := true
else isempty := false;
end;

FUNCTION NAME: lchild
MODULE NUMBER: 1.5.3.1.1.3
VERSION DATE: 19/11/83
VERSION NUMBER: 1.0

FUNCTION: To return a pointer to the left child of a
node in a binary tree. lchild receives a
a pointer to a node as input, if that pointer
is "nil" an error exists because there is not
a left child of a "nil" node. In this case
an error code is not returned but lchild is
also set to "nil".

INPUTS: A pointer to the node of which a pointer to its
left child is desired.
OUTPUTS: A pointer to the left node of a parent node.
GLOBAL VARIABLES USED: none.
GLOBAL VARIABLES CHANGED: none.
LIBRARY ROUTINES USED: none.
MODELS CALLED: none.
CALLING MODULES:
FILES READ: none.
FILES WRITTEN: none.
INSTALLATION: DEC Rainbow 100, CP/M-80

function lchild(root : btptr ) : btptr ;

FILE: avl-1.pas
begin
  if root = nil then lchild := nil (* error condition *)
  else
    lchild := root^.left;
end;

FUNCTION rchild

FUNCTION: To return a pointer to the right child of a
node in a binary tree. rchild receives a
a pointer to a node as input, if that pointer
is "nil" an error exists because there is not
a right child of a "nil" node. In this case
an error code is not returned but rchild is
also set to "nil".

INPUTS: A pointer to the node of which a pointer to its
right child is desired.

OUTPUTS: A pointer to the right node of a parent node.

GLOBAL VARIABLES USED: none.

GLOBAL VARIABLES CHANGED: none.

LIBRARY ROUTINES USED: none.

MODULES CALLED: none.

CALLING MODULES:

FILES READ: none.

FILES WRITTEN: none.

INSTALLATION: DEC Rainbow 100, CP/M-80
function rchild(root : btptr) : btptr;
begin
  if root = nil then rchild := nil (* error condition *)
  else rchild := root^.right;
end;

(*FUNCTION NAME: dataval*)
(*MODULE NUMBER: 1.5.3.1.1.2*)
(*VERSION DATE: 19/11/83*)
(*VERSION NUMBER: 1.0*)
(*FUNCTION: To return the data value from a node in a binary tree. A pointer to a node in a tree is input to the "data" function. If the pointer is "nil" ie. does not point to a node the value "minint" is returned as the data value.*)
(*INPUTS: A pointer to a node in a binary tree.*)
(*OUTPUTS: The value stored in the "information" field of the node pointed to.*)
(*GLOBAL VARIABLES USED: none.*)
(*GLOBAL VARIABLES CHANGED: none.*)
(*LIBRARY ROUTINES USED: none.*)
(*MODULES CALLED: none.*)
(*CALLING MODULES:*)
(*FILES READ: none.*)
(*FILES WRITTEN: none.*)
(*INSTALLATION: DEC Rainbow 100, CP/M-80*)
FILE: avl-1.pas

function dataval(root : btptr) : wordtype;
begin
  if root = nil then dataval := BLANK
else
  dataval := root^.keyword;
end;
FILE: avl-2.pas

[.PL60]

(* file: avl-2.pas *)
(* version: 1.0 *)
(* date: 28 September 84 *)
(* description: This file contains routines to manipulate *)
(* binary trees. The remaining routines are *)
(* contained in files AVL-1.PAS, AVL-3.PAS, *)
(* and AVL-4.PAS. *)
(* procedures contained: *)
(* LNR (P : btptr) *)
(* displaytree (P : btptr) *)
(* treedispose (P : btptr) *)
(* author: Paul A. Moore, Capt, USAF *)

[.PA]

(* PROCEDURE NAME: makebt *)
(* MODULE NUMBER: 1.5.3.1.2.1 *)
(* VERSION DATE: 20/11/83 *)
(* VERSION NUMBER: 1.4 *)
(* FUNCTION: Makebt calls the library routine "new" to allocate *)
(* storage space for a node in a binary tree. Each *)
(* node has four fields: *)
(* info --> stores an integer value.*)
(* bf --> stores the balance factor for a node *)
(* left --> pointer to the node to the left *)
(* right --> pointer to the node to the right *)

FILE: avl-2.pas
procedure makebt(var parent: btptr; value: wordtype);
    var newnode: btptr; side: integer);
begin
    (* Initialize the new node to be added to the binary tree *)
    new(newnode);
    newnode^.keyword := value;
end;
FILE: avl-2.pas

newnode^.wordnumber := 0;
newnode^.bf := 0; (* initial balance factor *)
newnode^.left := nil;
newnode^.right := nil;

(* "link" the new node into the tree *)
case side of
  LPARENT : parent := newnode;
  LEFT : parent^.left := newnode;
  RIGHT : parent^.right := newnode;
end; (* end makebt *)

(* PROCEDURE NAME: treetraversal *)
(* MODULE NUMBER: 1.5.7 *)
(* VERSION DATE: 19/11/83 *)
(* VERSION NUMBER: 1.1 *)
(* FUNCTION: treetraversal traverses a binary tree structure. *)
(* If the tree is empty treetraversal traverses the *)
(* tree structure in a "Right-Left-Node" (RLN) fashion *)
(* calling the routine "dispose" to return leaf nodes *)
(* to the system. Disposetree is a recursive *)
(* procedure. *)

(* INPUTS: A pointer to the root of the binary tree to be *)
(* disposed of. *)
(* OUTPUTS: An empty tree. *)
(* MODULES CALLED: isempty, lchild, rchild, treetraversal *)
(* MODULES CALLED BY: avl *)
(* LIBRARY ROUTINES USED: dispose *)
procedure treedispose( var p : btptr );
begin
  if not isempty(p) then (* make sure the tree isn't empty already *)
    begin
      if not isempty(p^.right) then
        treedispose(p^.right); (* traverse right subtree *)
      if not isempty(p^.left) then
        treedispose(p^.left); (* traverse left subtree *)
      dispose(p); (* return node to free pool *)
      p := nil;
    end;
  end; (* end treedispose *)

(* ********************************************************************* *)
(* * PROCEDURE NAME: LNR     *)
(* * MODULE NUMBER: 1.5.4.1  *)
(* * VERSION DATE: 09/11/83  *)
(* * VERSION NUMBER: 1.0     *)
(* * FUNCTION: The function of LNR is to traverse a binary tree *)
(* * starting at the root of the tree in a  *)
(* * "Left-Node-Right" manner. LNR is a recursive *)
(* * routine. LNR first follows the left pointers *)
(* * of each node until a "nil" pointer is encountered *)
(* * by calling itself with the left pointer of the *)
(* ********************************************************************* *)
FILE: avl-2.pas

(* current node. When a "nil" pointer is found *)
(* the information in the current node is stored in *)
(* an array and the index into the array is incremented *)
(* *)
(* INPUTS: P --> Pointer to tree to traverse *)
(* wordcount --> the number of words placed in the *)
(* output array plus 1. *)
(* OUTPUTS: data_recs.decode_dict.words --> array containing *)
(* sorted values of the tree passed to LNR,*)
(* wordcount *)
(* GLOBAL VARIABLES USED: nil *)
(* GLOBAL VARIABLES CHANGED: none. *)
(* LIBRARY ROUTINES USED: none. *)
(* MODULES CALLED: none. *)
(* CALLING MODULES: LNR *)
(* FILES READ: none. *)
(* FILES WRITTEN: none. *)
(* INSTALLATION: DEC Rainbow 100, CP/M-80 *)
******************************************************************************

procedure LNR(P : btptr ; var wordcount : integer );
begin
  if isempty(lchild(P)) then (* no nodes to the left *)
    begin (* so save information *)
      data_recs.decode_dict.words[wordcount] := dataval(P);
    end
  else begin
    LNR(lchild(P),wordcount); (* traverse the left subtree *)
      (* traversed all nodes to the *)
    (* left so save information *)
      (* data_recs.decode_dict.words[wordcount] := dataval(P); *)
  end

FILE: avl-2.pas Page: 5
FILE: avl-2pas Page: 6

end;

(* set wordnumber for later reference during creation of the menu *)
(* decoding paths. *)
P^.wordnumber := wordcount;
wordcount := wordcount + 1;

if not isempty(rchild(P)) then (* traverse the right subtree *)
  LNR(rchild(P),wordcount);
end; (* end LNR *)

[.PA]

******************************************************************************

(* PROCEDURE NAME: Displaytree *)
(* MODULE NUMBER: 1.5.6 *)
(* VERSION DATE: 20/11/83 *)
(* VERSION NUMBER: 2.1 *)
(* FUNCTION: To print a binary tree in a "readable" manner. *)
(* Displaytree calls printtree to traverse the *)
(* left and right subtrees of the root node printing *)
(* the values of the nodes as it goes. *)
(* Note: printtree is a recursive procedure. *)
(* INPUTS: p --> a pointer to a node in a binary tree. *)
(* OUTPUTS: a "readable" binary tree. *)
(* GLOBAL CONSTANTS USED: LEFT, RIGHT *)
(* GLOBAL VARIABLES USED: Outflag, Outfile *)
(* LIBRARY ROUTINES USED: none. *)
(* MODULES CALLED: printtree, rchild, lchild, *)
(* isempty dataval *)
(* CALLING MODULES: make_menu *)
(* FILES READ: none. *)

FILE: avl-2pas Page: 6
FILE: avl-2.pas

(* FILES WRITTEN: Outflag *)
(* INSTALLATION: DEC Rainbow 100, CP/M-80 *)

procedure displaytree(p: btpt0);
var
  i, depth : integer;
  printpoint : array [1..20] of char;

{.PA}

(* procedure: printtree  *
  * module number: 1.5.6.1  *
  * version: 1.2  *
  * date: 30 October 1984  *
  * history: 9/28/84 created  *
  * description: This procedure prints a binary tree.  *
  * inputs: node, depth, LR  *
  * outputs:  *
  * global variables used: Outflag, Outfile  *
  * global constants used: LEFT, RIGHT  *
  * modules called: printtree  *
  * calling modules: displaytree  *
  * files read: none  *
  * files written: Outfile  *
  * author: Paul A. Moore, Capt, USAF *)

procedure printtree( node: btpt0 ; depth: integer; LR: integer);
var i: integer;

begin
FILE: avl-2.pas

if node <> nil then
  begin
    printtree(lchild(node), (depth+1), LEFT);
    if LR = LEFT then
      printpoint[depth] := '|';
    if rchild(node) <> nil then
      printpoint[depth+1] := '|';

    (* print line *)
    if Outflag then
      begin
        for i := 1 to depth do
          write(Outfile, ',printpoint[i]);
        write(Outfile, '<',node^.keyword,'>');
        if isempty(lchild(node)) and isempty(rchild(node)) then
          writeln(Outfile)
        else writeln(Outfile, '<');
      end
    else
      begin
        for i := 1 to depth do
          write(',printpoint[i]);
        write('<',node^.keyword,'>');
        if isempty(lchild(node)) and isempty(rchild(node)) then
          writeln
        else writeln('<');
      end;

    if LR = RIGHT then printpoint[depth] := '|
    printtree(rchild(node), (depth+1), RIGHT);
    printpoint[depth+1] := '|
  end;

FILE: avl-2.pas
FILE:       avl-2.pas

end;    (* end printtree *)
{.PA}
(*--------------------------------------------------------------- *)
begin    (* beginning of displaytree *)
  if p <> nil then
    begin
      for i := 1 to 20 do printpoint[i] := ' ';
      (* traverse left side of tree *)
      depth := 0;
      printtree(lchild(p),(depth+1),LEFT);
      if Outflag then
        begin
          write(Outfile,'<',p^.keyword:0,'>');
          if isempty(lchild(p)) and isempty(rchild(p)) then
            writeln(Outfile)
          else writeln(Outfile,'|');
        end
      else
        begin
          write('<',p^.keyword:0,'>');</n
          if isempty(lchild(p)) and isempty(rchild(p)) then writeln
            else writeln('|');
        end;

      (* traverse right side of tree *)
      printtree(rchild(p),(depth+1),RIGHT);
    end
  end;
end;

FILE:       avl-2.pas
FILE: avl-3.pas

(* file: AVL-3.PAS *)
(* version: 1.0 *)
(* date: 28 September 84 *)
(* description: This file contains routines for manipulating *)
(* binary trees. *)
(* procedures contained: *)
(* addtotree (value: integer; var root: btptr) *)
(* leftbalance (A, B: btptr) *)
(* rightbalance (A, B: btptr) *)
(* author: Paul A. Moore, Capt, USAF *)

(* NOTE: THIS ROUTINE IS NOT USED BY THE MICROSDW SO IT IS COMMENTED OUT*)
(* OPERATION NAME: addtotree *)
(* MODULE NUMBER: *)
(* VERSION DATE: 29/11/83 *)
(* VERSION NUMBER: 1.2 *)
(* FUNCTION: Addtotree adds values to a binary tree. Values *)
(* less than or equal to the value of the current *)
(* node are stored in the left subtree of the current *)
(* node. Values greater than the value of the *)
(* current node are stored in the right subtree of *)
(* the current node. *)
(* INPUTS: value --> The value to be stored in the binary tree. *)
(* position --> The position in the tree to start looking *)
(* for the position in the tree to store the *)
(* new value. *)
(* OUTPUTS: A modified binary tree with the value added to the *)
FILE: avl-3.pas

* input tree as a leaf node.
* GLOBAL VARIABLES USED:
  * PARENT --> insert the new value as a root node.
  * LEFT  --> insert the new value as a left subtree
  * RIGHT --> insert the new value as a right subtree
* GLOBAL VARIABLES CHANGED: none.
* LIBRARY ROUTINES USED: none.
* MODULES CALLED: lchild, rchild, makebt, isempty, dataval
* CALLING MODULES:
* FILES READ: none.
* FILES WRITTEN: none.
* INSTALLATION: DEC Rainbow 100, CP/M-80

***************************************************************

{ procedure addtotree(value : wordtype ; var root : btptr );
var
  position,
  newnode : btptr ;  (* pointer to location of new node *)
  placed : integer;  (* indicates where the 'value' is *)
                         (* to be placed in the tree. *)
begin
  (* initialize for traversing the tree *)
  placed := 0;
  position := root;

  (* find the place for the new node in the tree *)
  (* pointed to by position. *)
  repeat
  if isempty(position) then  (* empty tree *)
    placed := LParent
  else  (* tree not empty *)
    if (value < position^.keyword) then
      position := position^.LChild
    else
      place := position^.RChild
  until (value = position^.keyword);  (* found the correct place *)

if isempty(1child(position)) then (* found place for value *)
    placed := LEFT
else
    position := 1child(position)
else
    if (value > position^.keyword) then
        if isempty(rchild(position)) then (* found place for value *)
            placed := RIGHT
        else
            position := rchild(position)
        else (* value = position^.keyword *)
            placed := 4;
    (* end else *)
until placed <> 0;
if placed <> 4 then
    makebt(position,value,newnode,placed); (* add "value" node *)
if placed = LPARENT then root := position; (* take care of new tree case *)
end; (* end addtotree *)

FILE: avl-3.pas
Page: 3
procedure leftbalance(var A, B : btptr);
var
  C : btptr ; (* pointer to right child of B *)
beginn
  if B^.bf = 1 then (* balance type LL *)
    begin (* debug write('LL') ; *)
      A^.left := B^.right;
      B^.right := A;
      A^.bf := 0;
      B^.bf := 0;
    end
  else
    begin (* balance type LR *)
      C := B^.right;
      B^.right := C^.left;
      A^.left := C^.right;
      C^.left := B;
      C^.right := A;
      (* ------------------------------- *)
      (* fix balance factors *)
    end
end
FILE: avl-3.pas

(* ----------------------------- *)

case C^\bf of
1:
  begin (* LRB *)
    (* debug write('LRb'); *)
    A^\bf := -1;
    B^\bf := 0;
  end;
-1:
  begin (* LRC *)
    (* debug write('LRC'); *)
    A^\bf := 0;
    B^\bf := 1;
  end;
0:
  begin (* LRA *)
    (* debug write('LRA'); *)
    A^\bf := 0;
    B^\bf := 0;
  end;
end; (* end case *)

C^\bf := 0; (* B is a new root *)
B := C;
end; (* End balance type LR *)
end; (* End left imbalance correction *)

(*PROCEDURE NAME: rightbalance*)
(*MODULE NUMBER: 1.5.3.1.2.3*)
(*VERSION DATE: 20/11/83*)
procedure rightbalance(var A, B : bptr);

var
  C : bptr ; (* Pointer to left child of B  *)

begin
  if B^.bf = -1 then
  begin
    (* balance type RR  *)
    (* debug write('RR');  *)
    A^.right := B^.left;
    B^.left := A;
    A^.bf := 0;
    B^.bf := 0;
  end
  else
  begin
    (* Balance type RL  *)
    C := B^.left;
    B^.left := C^.right;
    A^.right := C^.left;
  end
end
FILE: avl-3.pas

C^.left := A;
C^.right := B;

(* fix balance factors *)
(* fix balance factors *)
(* fix balance factors *)

case C^.bf of
  1:
    begin
      (* RLc *)
      (* debug write('RLc'); *)
      A^.bf := 0;
      B^.bf := -1;
      end;
  -1:
    begin
      (* RLb *)
      (* debug write('RLb'); *)
      A^.bf := 1;
      B^.bf := 0;
      end;
  0:
    begin
      (* RLa *)
      (* debug write('RLa'); *)
      A^.bf := 0;
      B^.bf := 0;
      end;
end; (* end case *)

C^.bf := 0;    (* B is a new root *)
B := C;
end; (* End balance type RL *)
(* End right imbalance correction *)
end; (* end rightbalance *)
FILE: avl-4.pas

[PL60]

file: AVL-4.PAS
version: 1.0
date: 28 September 84
description: This file contains routines to manipulate binary tree data structures.

procedures contained:
avlinsert (element : integer; var position : btpt;)
bttlocate (word : wordtype; btree : btpt;
var node_ptr : btpt; var found : boolean);

author: Paul A. Moore, Capt, USAF

[PA]

PROCEDURE NAME: btlocate

MODULE NUMBER: 1.5.3.1.1
VERSION DATE: 8/16/84
VERSION NUMBER: 1.0

FUNCTION: btlocate searches a binary tree for a value.
If the word is located a pointer to the word is
returned and the "found" flag is set to true.
If the word is not located a pointer to the
potential parent is returned and the "found" flag
is set to false.

INPUTS: word --> a word to locate in the tree
btree --> pointer to the root of the binary tree
to search.
procedure btlocate( word : wordtype ; btree : btptr;
var node_ptr : btptr; var found : boolean);
var P : btptr;
begin
P := btree;
found := false; (* default to word not found *)
node_ptr := nil; (* no node found yet *)

while (not isempty(P)) and (not found) do
begin
if word = dataval(P) then
begin (* word = P^.keyword *)
found := true;
node_ptr := P;
end
else
if word < dataval(P) then P := lchild(P)
else P := rchild(P) (* word > P^.keyword *)
end; (* end while *)
end; (* end btlocate *)

FILE: avl-4.pas Page: 2
procedure avlinsert(element : wordtype; var position : btptr ;
    var newnode : btptr );

var A, B,
FILE: avl-4.pas

F, P, Q : btptr;
d : integer;
done : boolean;
found : boolean;

begin
(* debug write(' Add element <',element, '> to avl tree, ');

(* find the insertion point for the element *)

if isempty(position) then (* empty avl tree *)
    makebt(position,element,newnode,LPARENT)
else
    begin
        A := position;          (* initialize pointers, root *)
        P := position;
        B := nil;
        F := nil;
        Q := nil;               (* pointer to parent of new node *)

        found := false;
        while (not isempty(P)) and (not found) do
            begin
                if P^.bf <> 0 then
                    begin
                        A := P;
                        F := Q;
                        end;

                Q := P;

                if element < dataval(P) then  P := lchild(P)
                else
                    begin
                        A := P;
                        F := Q;
                        end;

                found := true;
            end;
        end;

        if done then
            begin
                d := 0;
                done := false;
            end;
        else
            begin
                d := d + 1;
                done := true;
            end;

end;
if element > dataval(P) then  P := rchild(P)
else (* word = P^.keyword  *)
    found := true; (* Q/P points to node  *)
end; (* end while  *)

(* ----------------------------------------------------------- *)
(* Insert new node in the avl tree and rebalance.  *)
(* Duplicate "elements" are not inserted in the tree  *)
(* ----------------------------------------------------------- *)
if not found then
begin
    if element <= dataval(Q) then makebt(Q,element,newnode,LEFT)
else makebt(Q,element,newnode,RIGHT);

(* ----------------------------------------------------------- *)
(* Adjust the balance factors from A to Q.  *)
(* All nodes between A and Q will have balance factors of  *)
(* zero because A points to the most recent node with a  *)
(* balance factor of + or - 1.  *)
(* d = +1 implies that "element" is inserted in the left  *)
(* subtree of A.  *)
(* d = -1 implies that "element" is inserted in the right  *)
(* subtree of A.  *)
(* ----------------------------------------------------------- *)

if element > dataval(A) then
begin
    P := rchild(A);
    d := -1;
end
else begin

FILE:  avl-4.pas  Page: 5
FILE: avl-4.pas

    P := lchild(A);
    d := +1;
    end;

B := P;  (* save "B" node pointer for balancing *)

while P <> newnode do
    if element > dataval(P) then
        begin (* height of right subtree increases *)
            P^.bf := -1;
            P := rchild(P);
        end;
    else
        begin (* height of left subtree increases *)
            P^.bf := +1;
            P := lchild(P);
        end;
    (* end if *)
(* end while *)

(* Check to see if the tree is unbalanced *)

done := false;
if A^.bf = 0 then
    begin (* tree still balanced *)
        A^.bf := d;
        done := true;
    end;
else if (A^.bf + d) = 0 then
    begin (* tree still balanced *)
        A^.bf := 0;
    end;

FILE: avl-4.pas
FILE: avl-4.pas

done := true;
end;

if done then
  (* debug write('no balancing necessary') *)
else
  (* not done yet *)
begin
  (* debug write('balance type '); *)
  if d = 1 then
    (* Left imbalance *)
    leftbalance(A,B)
  else
    (* d = -1, Right imbalance *)
    rightbalance(A,B);
(* debug write(' performed'); *)
(* End Imbalance correction *)

(* ---------------------------------------------- *)
(* Subtree with root B has been rebalanced *)
(* and is the new subtree of F. The original *)
(* subtree of F had root A *)
(* ---------------------------------------------- *)

if F = nil then position := B
else if A = lchild(F) then F^.left := B
  else if A = rchild(F) then F^.right := B;
end; (* end if not "done" *)
end; (* end if not "found" *)
end; (* end if not an empty tree *)
FILE: avl-4.pas

(* debug writeln(' '); *)
end;  (* end avlinsert  *)
FILE: defcall.pas

{.PL60}

(file: defcall.pas
 version: 1.1
 date: 3 September 84
 description: this file contains procedures which
 are part of the MICROSDW BUILDDAT program.
 language: TURBO-Pascal
 procedures contained: define_call_routine,
 findcall, addcall
 author: Paul A. Moore, Capt, USAF

procedure: findcall
 module number: 1.5.3.4.1
 version: 1.0
 date: 16 August 1984
 history: 8/16/84 created
 description:
 1.5.3.4.1 FINDCALL- This process checks the call-routine
 list to see if a call-routine has been previously defined for
 another menu path. If the call-routine name is already in
 the list a pointer to it is returned otherwise a "nil" pointer is returned.

  inputs: call_list, word_pointer
  outputs: call_pointer

FILE: defcall.pas
procedure findcall ( call_list : callptr; word_pointer : btptr;
  var call_pointer : callptr );
begin
  call_pointer := nil; (* default to call "not found" *)
  while (call_list <> nil) and (call_pointer = nil) do
  begin
    if call_list^.wordptr = word_pointer then
      call_pointer := call_list
    else
      call_list := call_list^.nextcall;
  end;

[.PA]
(**************************************************************************
  * procedure: addcall
  * module number: 1.5.3.4.2
  * version: 1.0
  * date: 16 August 1984
  * history: 8/16/84 created
  * description:
  * 1.5.3.4.2 ADDCALL- This process adds a node containing a
***************************************************************************]
FILE: defcall.pas

* pointer to a call-routine name to the current call-routine *
* list. *
* *
* inputs: call_list, word_pointer *
* outputs: call_list, call_pointer *
* global variables used: none *
* global constants used: none *
* modules called: none *
* calling modules: define_call_routine *
* files read: none *
* files written: none *
* author: Paul A. Moore, Capt, USAF *
* *
*******************************************************************************

procedure addcall ( var call_list : callptr; word_pointer : btptr;
                  var call_pointer : callptr );
begin
  (* allocate a new "call list" node *)
  new(call_pointer);
  call_pointer^.wordptr := word_pointer;
  call_pointer^.callnumber := 0;
  call_pointer^.nextcall := call_list;  (* insert new node at *)
                              (* front of the list  *)
  call_list := call_pointer;
end;

[.PA]
*******************************************************************************
(* procedure: define_call_routine *
* module number: 1.5.3.4 *)

FILE: defcall.pas
FILE:  defcall.pas

* version:  1.1  *
* date:  3 September 84  *
* history:  8/16/84 created  *
*        9/3/84 fix calls to copy.  *
* description:  *
* 1.5.3.4 DEFINE_CALL_ROUTINE- This process defines a *
* "call-routine" for a menu option. Only menu options which *
* are leaves of the menu tree can have call-routines specified *
* for them. A "call-routine" is simply the name or keyword *
* used by the User Interface to determine which process to call *
* to carry out the action specified by the selected menu option. *
* First the menu path to specify the call-routine for is *
* checked to make sure that it exists, that it is a leaf node, *
* and that a call-routine has not been already defined for that *
* menu path. The name of the call-routine is then added to the *
* keyword tree. Then the call-routine is added to the list of *
* call routine names. *
* inputs: menu_tree, word_tree, call_list, parse_buf *
* outputs: call_list *
* global variables used: none *
* global constants used: none *
* modules called: copymenuword, addword, findcall, *
*                 addcall, errormsg, locate_menu_node *
* calling modules: addtomenu *
* files read: none *
* files written: none *
* author: Paul A. Moore, Capt, USAF *
*
**************************************************************************

procedure define_call_routine (menu_tree : mtptr; word_tree : btptr;

FILE: defcall.pas

var call_list : callptr;
parse_buf : parserecord);

label 9999;
var

menu_path : menuline;
menu_position : mtptr;
callword : wordtype;
word_pointer : btiptr;
call_pointer : callptr;
error : integer;
wordptr : btiptr;

begin
(* locate menu node to define the call routine for. *)
  menu_position := menu_tree;
  menu_path := copy(parse_buf.wordstr,1,parse_buf.startassign-1);
  locate_menu_node(menu_path,menu_position,error);
  if error <> SUCCESS then goto 9999;

(* check for errors in the menu node         *)
(* check for attempted definition of a call word for a non-leaf option *)
  if menu_position^.matchp <> nil then error := MERROR4;

(* check for redefinition of a call-word for a menu option  *)
  if menu_position^.endptr <> nil then error := MERROR5;
  if error <> SUCCESS then goto 9999;

copymenuword (parse_buf.wordstr, parse_buf.startassign, parse_buf.length, callword);

(* look for word in the keyword tree, if not found add it, return a *)
(* pointer to the keyword node.                                   *)
wordptr := word_tree;

FILE: defcall.pas

Page: 5
addword (callword, wordptr, word_pointer );

(* look for the word in the "call list" *)
findcall(call_list, word_pointer, call_pointer );

(* add the word to the "call list" if its not in the list yet *)
if call_pointer = nil then
  addcall (call_list, word_pointer, call_pointer );

(* point to the node containing the pointer to the word to call *)
menu_position^ .endptr := call_pointer;

9999: if error <> SUCCESS then errmsg(error);
end;
FILE:  defmenu.pas

{PL60}                                                                                           Page: 1

(************************************************************************************

* *
* file:       defmenu.pas *
* version:    1.0 *
* date:       16 August 1984 *
* description: this file contains procedures which are part of the MICROSDW BUILDDAT program.
* language:   TURBO-Pascal *
* procedures contained: define_menu *
* author:     Paul A. Moore, Capt, USAF *
* ************************************************************************************

(************************************************************************************

* *
* procedure:  define_menu *
* module number: 1.5.3.3 *
* version:    1.0 *
* date:       16 August 1984 *
* history:    8/16/84 created *
* description: 1.5.3.3 DEFINE_MENU- This process sets the current_menu pointer to point to the menu node which a submenu is to be defined for. If the menu option is to use a previously defined submenu, the current_menu pointer will be set to nil and the "match pointer" for the menu option will point to the previously defined submenu. The buffer (parse_buf) contains a sequence of keywords specifying the new menu to be defined. Each keyword in the sequence represents a lower level in the menu hierarchy. A menu node must exist for each of the

FILE:  defmenu.pas                                                                                          Page: 1
FILE:  defmenu.pas

* keywords.  If  parse_buf indicates that a previously defined *
* submenu is to assigned to this menu a second sequence of *
* keywords is contained in parse_buf specifying the predefined *
* submenu.  The previously defined submenu must also exist. *
* *
* inputs: menu_tree, current_menu, parse_buf
* outputs: menu_tree
* global variables used: none
* global constants used: none
* modules called: locate_menu_node, errmsg
* calling modules: addotmenu
* files read: none
* files written: none
* author:  Paul A. Moore, Capt, USAF
*
********************************************************************************

procedure define_menu(var menu_tree, current_menu : mtpttr;
                      parse.buf : parserrecord);
label  9999;  (* error exit   *)
var
  menu_path     : menuline;
  menu_position : mtpttr;
  error         : integer;
  assign_node   : mtpttr;  (* pointer to the menu node   *)
                   (* containing pointer to submenu*)
                   (* to assign as the submenu for  *)
                   (* this menu path.               *)
begin
  error := SUCCESS;
  if parse_buf.startassign = 0 then
    menu_path := copy(parse_buf.wordstr,1,parse_buf.length)
FILE: defmenu.pas

else (* assignment *)
    menu_path := copy(parse_buf.wordstr,1,parse_buf.startassign-1);

menu_position := menu_tree;
locate_menu_node (menu_path, menu_position, error);
if error <> SUCCESS then goto 9999;

if parse_buf.startassign <= 0 then (* menu words follow in menu.txt *)
    current_menu := menu_position
else
    begin (* assigning this menu the same submenu as *)
        (* another menu *)
        (* locate start of other menu *)
        menu_path := copy(parse_buf.wordstr,parse_buf.startassign,
                          parse_buf.length-parse_buf.startassign + 1);
        assign_node := menu_tree;
        locate_menu_node (menu_path,assign_node,error);
        if error <> SUCCESS then goto 9999;

        if assign_node^.matchp=nil then error := MERROR1
        else
            begin (* assign submenu to this menu also *)
                menu_position^.matchp := assign_node^.matchp;
                current_menu := nil; (* no menu being created now *)
            end;
    end;

9999: if error <> SUCCESS then errormsg(error);
end;
FILE: disproc.pas

(* *******************************************************)
(* file: disproc.pas *)
(* version: 1.0 *)
(* date: 18 August 1984 *)
(* description: this file contains procedures which *)
(* are part of the MICROSDW BUILDDAT *)
(* program. *)
(* language: TURBO-Pascal *)
(* procedures contained: callsdispose *)
(* author: Paul A. Moore, Capt, USAF *)

(* *******************************************************)
(* procedure: callsdispose *)
(* module number: 1.5.9 *)
(* version: 1.0 *)
(* date: 18 August 1984 *)
(* history: 8/18/84 created *)
(* description: This procedure disposes of the linked *)
(* list of call routines data structure *)
(* (call_list). *)

(* inputs: call_list *)
(* outputs: call_list *)
(* global variables used: none *)
(* global constants used: none *)
(* modules called: none *)
(* calling modules: make_menu *)
(* files read: none *)

FILE: disproc.pas
procedure callsdispose (var call_list : callptr );

var   temp_call       : callptr;

begin
  while call_list <> nil do
    begin
      temp_call := call_list^.nextcall;
      dispose ( call_list );
      call_list := temp_call;
    end;
  end; (* end callsdispose  *)
FILE: errormsg.pas

{PL60}

file: errormsg.pas
version: 1.0
date: 21 August 1984
description: this file contains procedures which are part of the MICROSDW BUILDDAT program.
language: TURBO-Pascal
procedures contained: errormsg
author: Paul A. Moore, Capt, USAF

1.5.10 ERRORMSG- This procedure prints out the error messages for the menu processing procedures of BUILDDAT. It receives an error number as input and prints the corresponding message.

inputs: error number
outputs:
global variables used: Outfile, Outflag
global constants used: error constants
procedure errmsg( error : integer );
begin
  write('<ERROR> ');
  case error of
    MERROR1 : writeln('submenu for menu definition does not exist.');
    MERROR2 : writeln('incomplete menupath.');
    MERROR3 : writeln('extra chars following a complete path specification.');
    MERROR4 : writeln('a call word cannot be defined for a keyword',
        'with a submenu.');</n    MERROR5 : writeln('redefinition of a call word for a menu option',
        'is not allowed.');
    MERROR6 : writeln('Error opening MENU.TXT');
    MERROR7 : writeln('Error # 7');
    MERROR8 : writeln('Error # 8');
    else writeln('Undefined error number = ',error:0);
  end;
  if Outflag then
    begin
      writeln(Outfile,'<ERROR> ');
      case error of
        MERROR1 :
          writeln(Outfile,'submenu for menu definition does not exist.');
        MERROR2 : writeln(Outfile,'incomplete menupath.'));
  end;
FILE: errmsg.pas
FILE: errormsg.pas

MERROR3 : writeln
    (Outfile,'extra chars folowing a complete path specification.');
MERROR4 : writeln(Outfile,'a call word cannot be defined for a ',
    'keyword with a submenu.');
MERROR5 : writeln(Outfile,'redefinition of a call word for a menu',
    ' option is not allowed.');
MERROR6 : writeln(Outfile,'Error opening MENU.TXT');
MERROR7 : writeln(Outfile,'Error # 7');
MERROR8 : writeln(Outfile,'Error # 8');
else     writeln (Outfile,'Undefined error number = ',error:0);
end;
end;
end;
end;
FILE: getword.pas

{.PL60}

(file: getword.pas
version: 1.2
date: 3 September 84
description: this file contains a procedure to extract keywords from menu lines.
The procedure are part of the MICROSDW BILDDAT program.
language: TURBO-Pascal
procedures contained: get_word
author: Paul A. Moore, Capt, USAF)

procedure: get_word
module number: 1.5.2.2
version: 1.2
date: 3 September 84
history: 8/16/84 created
description:
1.5.2.1 GET WORD (GETWORD) - This process extracts menu keywords from the MENU.TXT file records.

inputs: wordline, start, endpos
outputs: start, word
global variables used: none
global constants used: none
modules called: none

FILE: getword.pas
procedure get_word ( wordline : menuline; var start : integer;
               endpos : integer; var word : wordtype );
var  strend : integer;
begin
  (* skip leading blanks  *)
  while (wordline[start] = BLANK) and (start <= endpos) do
    start := start + 1;
  strend := start; (* save position of first char. of word *)

  (* find end of word *)
  while (strend <= endpos) and
         (wordline[strend] <> ENDSTR) and
         (wordline[strend] <> BLANK) do
    strend := strend + 1;

  (* copy word       *)
  word := copy(wordline, start, (strend-start) );

  (* set start to point to beginning of next potential word   *)
  start := strend + 1; (* skip terminating character     *)
end;

FILE:    getword.pas
FILE: locmenu.pas

[PL60]

file: locmenu.pas
version: 1.1
date: 3 September 1984
description: this file contains two routines which validate a menu path.
The procedures are part of the MICROSDW BUILDDAT program.
language: TURBO-Pascal
procedures contained: locate_menu_node, copymenuword
author: Paul A. Moore, Capt, USAF

procedure: copymenuword
module number: 1.5.3.3.1.1
version: 1.0
date: 16 August 1984
history: 8/16/84 created
description:
1.5.3.3.2 COPYMENUWORD- This process copies menu words from previously parsed menu_line(s) from MENU.TXT.

inputs: source, start, stop
outputs: start, destination
global variables used: none
global constants used: BLANK
procedure copymenuword( source : menuline; var start : integer; 
                      stop : integer; var destination : wordtype); 
var wordend : integer; 
begin 
 (* copy keyword from source to destination, skip leading blanks, *) 
 (* stop when a blank or end of source is encountered *) 

  while (source[start] = BLANK) and (start < stop) do 
     start := start + 1; 

  wordend := start; 
  while (source[wordend] <> BLANK) and (wordend < stop) do 
     wordend := wordend + 1; 

  destination := copy(source, start, (wordend - start)); 
  start := wordend; (* update start location for next "copymenuword" *) 
end;
FILE: locmenu.pas

* date: 3 September 84
* history: 8/16/84 created
* 9/3/84 debug location problems
* description:
* 1.5.3.3.1 LOCATE_MENU_NODE- This process receives a
* sequence of keywords defining a menu option and locates the
* node in the menu.tree for the last keyword. Locate_menu_node
* starts at the root of the menu structure and searches for the
* first keyword. If the keyword is found then the "match
* pointer" for that keyword is followed to the next lower level
* in the menu hierarchy and the process is repeated for the
* next keyword int he keyword sequence. If the keyword is not
* found, an error flag is returned.
* *
* inputs: menu_path, node_ptr
* outputs: node_ptr, error
* global variables used: none
* global constants used: BLANK, SUCCESS, MERROR2, MERROR3
* modules called: copymenuword
* calling modules: define_menu
* files read: none
* files written: none
* author: Paul A. Moore, Capt, USAF
*
********************************************

procedure locate_menu_node ( menu_path : menuline; var nodeptr : mtptr;
                          var error : integer );

var
    pathlen : integer;
    start : integer;
    lastword,

FILE: locmenu.pas
FILE: locmenu.pas

word : wordtype;
found, done : boolean;
lastnode : mtptr;

begin

fillchar(lastword,wordlength,BLANK);
error := SUCCESS; (* no errors yet ! *)
start := 1;
pathlen := length(menu_path);
done := false; (* true = looked at complete menu_path *)
found := false; (* word matches keyword *)

(* debug writeln('locate_menu_node, menu_path = ',menu_path,''); *)
repeat
  copymenuword(menu_path,start,pathlen,word);
  if length(word) = 0 then
    done := true (* looked at all words in menu_path *)
  else
    begin
      found := false;
      repeat
        if (word <> nodeptr^.wordptr^.keyword) then
          nodeptr := nodeptr^.nomatchp (* try next word *)
        else
          begin
            lastword := word;
            found := true;
            lastnode := nodeptr;
            if nodeptr^.matchp = nil then
              done := true (* no lower levels to menu structure *)
          end
        end
      end
    end
end

FILE: locmenu.pas

nodeptr := nodeptr^.matchp;
end;
until found or (nodeptr = nil);
end;
until done or (not found);

nodeptr := lastnode;

(* check for extra characters in menu_path *)
while start <= pathlen do begin
  if menu_path[start] <> BLANK then error := MERROR3;
  if error <> SUCCESS then start := pathlen + 1
  else start := start + 1;
end;

if (nodeptr^.wordptr^.keyword <> lastword) then error := MERROR2;
end;

FILE: locmenu.pas
FILE: makemenu.pas

[.PL60]

file: makemenu.pas
version: 2.1
date: 28 November 84
description: this file contains the main routine for installing a new menu structure for the MICROSDW. The procedures are part of the MICROSDW BUDDAT program.
language: TURBO-Pascal
procedures contained: make_menu, init_menu
author: Paul A. Moore, Capt, USAF

procedure: init_menu
module number: 1.5.1
version: 1.0
date: 21 August 1984
history: 8/21/84 created
description:
1.5.1 INITIALIZE MENU (INIT_MENU)—This process initializes the pointers to the root nodes of the menu data structures, word_tree, menu_tree, and call_list.
inputs: wordtree, menutree, current_menu, call_list
outputs: wordtree, menutree, current_menu, call_list
global variables used: none
FILE: makemenu.pas

(* global constants used: DONEWORD *)
(* modules called: addword *)
(* calling modules: make_menu *)
(* files read: none *)
(* files written: none *)
(* author: Paul A. Moore, Capt, USAF *)

******************************************

procedure init_menu( var wordtree : btpt; var menutree : mtpt;
   var current_menu : mtpt; var call_list : callptr );
var wordpointer : btpt; (* pointer to location of "DONEWORD" *)
begin
(* initialize menu structures *)
   wordtree := nil;
   menutree := nil;
   current_menu := nil;
   call_list := nil;

(* insert word in tree used for terminating menus *)
   addword(DONEWORD, wordtree, wordpointer);
end;

{ PA }

******************************************
FILE: makemenu.pas

* 1.5 INSTALL SOFTWARE CONFIGURATION (MAKE_MENU) - This *
* process installs the menu system for the MICROSDW User *
* Interface. The process reads the menu definition from the *
* "menu.txt" file and creates the menu data structures stored *
* in MICROSDW.SYS. First the menu data structures are *
* initialized by init_menu, then the menu data structures are *
* created by iteratively calling readmenu and addtomenue until *
* the end of the MENU.TXT file is reached. The menu data *
* structures are then converted to the data structures used to *
* store the menu structure in the MICROSDW.SYS file.
* *
* inputs: none
* outputs: none
* global variables used: Outfile, Outflag
* global constants used: ENDMENU, MERROR6
* modules called: readmenu, add_keyword_to_menu,
* make_word_list, make_decoding_paths,
* displaytree, dispostree,
* disposecalls, menu_print
* calling modules: builddat
* files read: menu.txt
* files written: Outfile
* author: Paul A. Moore, Capt, USAF
*
* *******************************************************************************

procedure make_menu;
var
    wordtree : btptr;  (* pointer to root of keyword tree *)
    menutree : mtpttr; (* pointer to root of menu tree *)
    call_list : callptr; (* pointer to root of call routine *)
    (* list *)

FILE: makemenu.pas
FILE:  makemenu.pas

current_menu : mtptr;  (* ptr to the menu to which *)
                   (* keywords may be added *)
node_counter : integer;  (* counter for nodes (records) in *)
                   (* the menu structure *)
menu_text : text;  (* text file containing the menu *)
                   (* definition *)
result : integer;
parse_buf : parserecord;(* record containing "parsed" menu*)
                   (* definition record *)
level : integer;  (* used for printing menu_tree *)
resp : char;

begin
  writeln(' Beginning to process MENU.TXT');

  (* initialize menu structures    *)
  init_menu( wordtree, menu_tree, current_menu, call_list );

  (* initialize input file        *)
  assign(menu_text,'menu.txt');
  [$I-]  reset(menu_text);  [$I+]
  if IORESULT <> 0 then
    errmsg(MERROR6)
  else
    begin
      node_counter := 0;
      repeat
        readmenu(menu_text,parserecord);
        if parserecord.rectype <> ENDMENU then
          addtonode(wordtree, menu_tree, call_list,
                  parse_buf, node_counter, current_menu );
        until parse_buf.rectype = ENDMENU;

FILE:  makemenu.pas
FILE: makeproc.pas

{PL60}

file: makeproc.pas
version: 1.2
date: 20 October 1984
description: this file contains the routines which transform the dynamic menu data structures to static structures. The procedures are part of the MICROSDW BUILDDAT program.
language: TURBO-Pascal
procedures contained: make_word_list, calc_min_chars
make_decoding_paths, number_calls,
make_call_records, make_keyw_records
author: Paul A. Moore, Capt, USAF

procedure: calc_min_chars
module number: 1.5.4.2
version: 1.2
date: 20 October 1984
history: 9/26/84 created
description:
1.5.4.2 CALC_MIN_CHARS- This process calculates the minimum number of characters necessary to identify a particular keyword. The procedure uses the sorted array of keywords created by LNR to compare keywords. The keywords are compared in pairs character by character until a difference

FILE: makeproc.pas
FILE: makeproc.pas

* is found. If the minimum length of the first word is greater *
* than its previously calculated minimum the new minimum is *
* saved. The minimum length of the second word is set to the *
* number of the character where the first difference is *
* identified.
*
* inputs:    decode_dict, number_of_words
* outputs:   decode_dict
* global variables used: none
* global constants used: none
* modules called:  none
* calling modules: make_word_list
* files read: none
* files written: none
* author: Paul A. Moore, Capt, USAF
*
**************************************************************************

procedure calc_min_chars(var decode_dict : dict_buffer;
                         number_of_words : integer );

var
    word,
    First_len, Second_len,
    min_len, i    : integer;

begin
    with decode_dict do
    begin
        abbrev[0] := 0;  (* initialize length of first abbreviation *)
        word     := 0;   (* start with first word *)
        repeat
            FILE: makeproc.pas
FILE:    makeproc.pas

First_len := length(words[word]);
Second_len := length(words[word+1]);
min_len := First_len;
if min_len > Second_len then min_len := Second_len;
i := 1;    (* initialize character index *)
while (i <= min_len) and (words[word][i] = words[word+1][i]) do
  i := i + 1;

(* set the minimum number of characters for the current word *)
(* and the next word *)
if i > Second_len then abbrev[word+1] := Second_len
else
  abbrev[word+1] := i;

if i > First_len then abbrev[word] := First_len
else
  if i > abbrev[word] then abbrev[word] := i;

word := word + 1;    (* check out the next pair of words *)

until (word = number_of_words);
end;    (* end with *)
end;    (* end calc_min_chars *)

*******
*
*
* procedure: make_word_list
*
* module number: 1.5.4
* version: 1.0
* date: 17 August 1984
* history: 8/17/84 created
*
FILE:    makeproc.pas
**FILE: makeproc.pas**

```
* description: this procedure
* 1.5.4 MAKE_WORD_LIST- This process creates a sorted list of
* the keywords and call-routine names used in the menu
* structure. First, the words are sorted using the LNR
* procedure. The LNR procedure puts the words in ascending
* order into the "words" array, and assigns numbers matching
* the position of the word in the array to each node in the
* keyword tree. This number is used by make_decoding_paths to
* associate the appropriate "word number" with each decoding
* path record. The minimum number of characters necessary to
* identify each keyword are then calculated by the procedure
* calc_min_chars.
* *
* inputs: word_tree
* outputs:
* global variables used: data_recs.decode_dict, Outfile,
*                      Outflag
* global constants used: none
* modules called: lnr, calc_min_chars
* calling modules: make_menu
* files read: none
* files written: Outfile
* author: Paul A. Moore, Capt, USAF
*
* ****************************************
```

```pascal
procedure make_word_list(word_tree : btptr);
var
    number_of_words, i : integer;
begin
    writeln;
    writeln('Sort the Keywords.');
```

**FILE: makeproc.pas**
FILE: makeproc.pas

(* sort word_tree into the WORDS array, and number the words *)
number_of_words := 0;
lnr(word_tree, number_of_words);

(* calculate minimum characters per keyword *)
writeln('Calculate Minimum Characters per Keyword.');
calc_min_chars(data_recs.decode_dict, number_of_words-1);

(* print listing of Words *)
if Outflag then
  begin
    writeln(Outfile, chr(ff), ' List of Unique Words in Menu Structure');
    writeln(Outfile, ' word# keyword abbrev');
    with data_recs.decode_dict do
      for i := 0 to number_of_words-1 do
        writeln(Outfile, ' ', i:4, ',words[i]:wordlength, ',abbrev[i]:4);
  end;
end;

[.PA]

******************************************************************************
******************************************************************************
*C
* procedure: number_calls
* module number: 1.5.5.1
* version: 1.0
* date: 17 August 1984
* history: 8/17/84 created
* description:
* 1.5.5.1 NUMBER_CALLS- This process assigns numbers to the *
* nodes in the call list corresponding to their record number *

FILE: makeproc.pas
```pascal
FILE: makeproc.pas

* in the decoding paths.
* * inputs: call_list, node_counter
* * outputs: node_counter
* * global variables used: none
* * global constants used: none
* * modules called: none
* * calling modules: make_decodin..paths
* * files read: none
* * files written: none
* * author: Paul A. Moore, Capt, USAF
*

******************************************************************************************************************************************

procedure number_calls(call_list : callptr ; var node_counter : integer);
begin
(* add "call numbers" to the call_list nodes *)
(* these are the "DONE" records in the decoding paths *)

while call_list <> nil do
begin
node_counter := node_counter + 1;
call_list^.callnumber := node_counter;
call_list := call_list^.nextcall;
end;
end;

{.PA}

******************************************************************************************************************************************
* *
* procedure: make_call_records
* * module number: 1.5.5.2
*
```
FILE: makeproc.pas

* version: 1.0
* date: 17 August 1984
* history: 8/17/84 created
* description:
* 1.5.5.2 MAKE_CALL_RECORDS- This process fills in the data
* for the call-routine termination records in the decoding path
* structure.
*
* inputs: call_list, word_tree
* outputs:
* global variables used: data_recs.decode_dict,ptrs
* global constants used: DONEWORD
* modules called: btlocate
* calling modules: make_decoding_paths
* files read: none
* files written: none
* author: Paul A. Moore, Capt, USAF
*
********************************************************************

procedure make_call_records( call_list : callptr ; word_tree : btptr);
const
  wordnum = 1;
  match = 2;
  nomatch = 3;

var
  doneptr : btptr;
  found : boolean;

begin (* locate the "done word" for the decoding paths  *)
  btlocate(DONEWORD,word_tree,doneptr,found);

FILE: makeproc.pas
FILE: makeproc.pas

(* process all the call routines. *)
while call_list <> nil do
  begin
    with data_recs.decode_dict do
      begin
        ptrs[call_list^.callnumber,wordnum] :=
          doneptr^.wordnumber;
        ptrs[call_list^.callnumber,match] :=
          call_list^.wordptr^.wordnumber;
        ptrs[call_list^.callnumber,nomatch] := ENDCODE;
      end;
    call_list := call_list^.nextcall;
  end;
end;

{PA}

*******************************************************************************
* *
* procedure:    make_keyw_records
* *
* module number: 1.5.5.3
* *
* version: 1.0
* *
* date: 21 August 1984
* *
* history: 8/17/84 created
* *
* history: 8/21/84 deleted extraneous logic
* *
* description: this procedure
* 1.5.5.3 MAKE_KEYW_RECORDS- This is a "Node-Left-Right"
* recursive procedure which traverses the menu tree (menu_tree)
* producing the decoding path record for each node in the tree.
* Duplicate processing of shared menus is eliminated by
* checking to see if the decoding path record for that node has
* already been created.
**FILE: makeproc.pas**

* inputs: menu_tree
* outputs: *
* global variables used: data_recs.decode_dict.ptrs *
* global constants used: none *
* modules called: make_keyw_records *
* calling modules: make_decoding_paths *
* files read: none *
* files written: none *
* author: Paul A. Moore, Capt, USAF *
*
************************************************************

procedure make_keyw_records(menu_tree : mtptr );

const
  wordnum = 1;
  match = 2;
  nomatch = 3;

begin
 (* recursive procedure to define all of the decoding path *)
 (* records, in the decode_dictionary *)

  with data_recs.decode_dict do
    if (menu_tree <> nil) and
       (ptrs[menu_tree^.node_number,wordnum] < 0) then
    begin
      ptrs[menu_tree^.node_number,wordnum] :=
        menu_tree^.wordptr^.wordnumber;

      if menu_tree^.matchp <> nil then (* there is a submenu *)
        ptrs[menu_tree^.node_number,match] :=
          menu_tree^.matchp^.node_number
FILE: makeproc.pas

else (* no submenu(s), so *)
(* point to a call record *)
  ptrs[menu_tree^.node_number,match] :=
  menu_tree^.endptr^.callnumber;
if menu_tree^.nomatchp <> nil then (* more options in this menu *)
  ptrs[menu_tree^.node_number,nomatch] :=
  menu_tree^.nomatchp^.node_number
else (* no more options in menu *)
  ptrs[menu_tree^.node_number,nomatch] := ENDCODE;

(* do the rest of this menu *)
make_keyw_records(menu_tree^.nomatchp);

(* do the submenu for this menu option *)
make_keyw_records(menu_tree^.matchp);
end;

[.PA]
(***********************************************************************
*  *  procedure:      make_decoding_paths   *  *
*  *  module number:  1.5.5  *  *
*  *  version:        1.0  *  *
*  *  date:           17 August 1984  *  *
*  *  history:        8/17/84 created  *  *
*  *  description:    *  *
* 1.5.5 MAKE_DECODING_PATHS- This process calls three *  *
* subprocesses to create the decoding paths for the menu *  *
* structure. In put to make_decoding_paths are the pointers to *  *
* the call_list, wordtree, menutree, and the number of nodes in *  
***********************************************************************

FILE: makeproc.pas
FILE: makeproc.pas

* the menutree. Output from the subprocesses is *
* decode_dictionary pointer records for the decoding paths. *
* *
* inputs: call_list, wordtree, menutree, *
* node_counter *
* outputs: node_counter *
* global variables used: Outfile, Outflag *
* data_recs.decode_dict.ptrs *
* global constants used: none *
* modules called: number_calls, make_call_records, *
* make_keyw_records *
* calling modules: make_menu *
* files read: none *
* files written: Outfile *
* author: Paul A. Moore, Capt, USAF *
* *
***********************************************************************

procedure make_decoding_paths(call_list : callptr; wordtree : btptr;
menutree : mt ptr; var node_counter : integer);
var
   i, wordp : integer;
begin
   (* add "call numbers" to the call routines in the call_list *)
   number_calls(call_list, node_counter);
   (* make the decoding records for the call routines *)
   (* initialize the wordpointers so shared menu paths are only *)
   (* processed once by make_keyw_records. *)
   for i := 1 to node_counter do
      data_recs.decode_dict.ptrs[i, 1] := -1;
      make_call_records(call_list, wordtree);
FILE: makeproc.pas

(* make the decoding records for the menu options *)
make_keyw_records(menu_tree);

(* print out the decoding paths *)
if Outflag then
begin
  writeln(Outfile,chr(ff));
  writeln(Outfile,'Decoding paths:');
  writeln(Outfile,'rec# word word# matchp nomatchp');
  for i := 1 to node_counter do
    with data_recs.decode_dict do
      begin
        wordp := ptrs[i,1];
        writeln(Outfile,i:4, ' ', words[wordp]:wordlength,
                 ptrs[i,1]:6, ptrs[i,2]:6, ptrs[i,3]:6);
      end; (* end with, end for *)
  end;
end;

FILE: makeproc.pas
FILE: menucons.pas

(*--------------------------------------------------------------------------*
* file: menucons.h
* version: 1.0
* date: 15 August 1984
* description: this file contains the constant definitions for the make_menu
* routines.
* author: Paul A. Moore, Capt, USAF
*)

(*--------------------------------------------------------------------------*
* menu constant declarations
*)

const
SUCCESS = 0;
ENDMENU = 0;
BEGINMENU = 1;
MENUDEF = 2;
CALLDEF = 3;
MENUWORD = 4;
ENDDEF = 5;

BLANK = ' ';  
COMMENT = ' ';  
ASSIGNMENT = '=';  
ENDSTR = '.';

MAXMENU = 99;  (* maximum # chars in a menu definition record *)

FILE: menucons.pas
FILE: menucons.pas

(* ERROR codes *)
MERROR1 = 1; (* submenu for menu definition does not exist *)
MERROR2 = 2; (* incomplete menupath *)
MERROR3 = 3; (* extra chars following a complete path spec. *)
MERROR4 = 4; (* a call word cannot be defined for a keyword *)
MERROR5 = 5; (* with a submenu *)
MERROR5 = 5; (* redefinition of a call word for a menu option *)
MERROR6 = 6;
MERROR7 = 7;
MERROR8 = 8;
FILE: menuprnt.pas

[PL60]

************

* file: menuprnt.pas
* version: 1.1
* date: 30 October 1984
* description: this file contains procedures which
* are part of the MICROSDW BUILDDAT
* program.
* language: TURBO-Pascal
* procedures contained: menu_print
* author: Paul A. Moore, Capt, USAF

************

************

* procedure: menu_print
* module number: 1.5.8
* version: 1.1
* date: 30 October 1984
* history: 9/3/84 created
* description: this procedure
* 1.5.8 MENU_PRINT- This procedure traverses the menu tree
* (menu_tree) producing an indented listing of the menu options
* available in the menu structure.
* inputs: menutree, level
* outputs: printed menu structure
* global variables used: Outflag, Outfile
* global constants used: none

FILE: menuprnt.pas
procedure menu_print(menutree : mtptr; var level : integer );
var   i : integer;

begin /* indent */
  if Outflag then
    begin
      for i := 1 to level do write(Outfile, ' ');
      writeln(Outfile, menutree^.wordptr^.keyword);
    end
  else
    begin
      for i := 1 to level do write(' ');
      writeln(menutree^.wordptr^.keyword);
    end;

/* do submenu for current menu */
  if menutree^.matchp <> nil then
    begin
      level := level + 1;
      menu_print(menutree^.matchp, level);
      level := level - 1;
    end;
FILE: menuprnt.pas

(* do next option *)
if menutree^.nomatchp <> nil then
  menu_print(menutree^.nomatchp, level);
end;

FILE: menuprnt.pas
FILE: menutype.pas

(* ************************************************** *
* file: menutype.h                                   *
* version: 1.1                                      *
* date: 18 August 1984                              *
* description: this file contains the type definitions for the make_menu routines. *
* author: Paul A. Moore, Capt, USAF                 *
* ************************************************** *)

(* ************************************************** *
* menu type definitions                              *
* ************************************************** *)

type

  btptr = ^wtnode;   (* pointer to a binary word tree *)
  wtnode = record
    keyword : wordtype;   (* menu keyword, or call *)
                      (* routine name. *)
    wordnumber : integer; (* # of the word in the "words" *)
                      (* array, used in creating *)
                      (* decoding paths *)
    bf : integer;        (* balance factor *)
    left : btptr;        (* pointer to left subtree (child)*)
    right : btptr;       (* pointer to right subtree (child)*)
  end;

  callptr = ^callnode; (* pointer to a "call routine" list *)

FILE: menutype.pas
FILE: menutype.pas

begin instance...

(******************************...
(* end of menu type definitions *)
(******************************...

FILE: menutype.pas
EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

UNCLASSIFIED P A MOORE DEC 84 AFIT/GCS/ENG/84D-10
FILE: msdwcons.pas

(* file: msdwcons.pas
* version: 1.2
* date: 28 November 84
* description: this file contains the constant
  definitions for the MICROSDW
  routines.
* author: Paul A. Moore, Capt, USAF
*)

const
  term_length = 95; { length of array for terminal control data }
  printer_length = 50; { length of array for printer control data }
  ff = 12; { decimal for form feed char }
  wordlength = 9; { length of word in storage }
  num_param_group = 1;
  num_bools = 10;
  num_ints = 10;
  num_reals = 10;
  num_strings = 10;
  num_ptr_recs = 3;
  num_ptrs = 200;
  num_words = 75;
  num_msg_dir = 50;
  num_msg_line = 250;
  screen_width = 79;

ENDCODE = 9999;
DONEWORD = '$$$$$$$$$'

FILE: msdwcons.pas
FILE: msdwtpe.pas

(*-------------------------------------------------------------*
  * file:          msdwtpe.pas  *
  * version:       1.2  *
  * date:          28 November 84  *
  * description:   this file contains the  *
                  type definitions for the MICROSDW  *
                  routines.  *
  * author:        Paul A. Moore, Capt, USAF  *
  *------------------------------------------------------------------*)

type
  ptr_recs   = array[ 1..num_ptr_recs ] of integer;
  paramstring = string[14];

  msg
    loc_rec : integer;
  length : byte;
end;

dict_buffer = record
  ptrs    : array[ 1..num_ptrs ] of ptr_recs;
  words   : array[ 0..num_words ] of string[ wordlength ];
  abbrev  : array[ 0..num_words ] of integer;
end;

param_group = record
  bools   : array[ 1..num_bools ] of boolean;
  ints    : array[ 1..num_ints ] of integer;
  reals   : array[ 1..num_reals ] of real;

FILE: msdwtpe.pas
FILE: msdwtype.pas

strings : array[1..num_strings ] of paramstring;
end;

msg_dat = array[1..num_msg_line ] of string[screen_width];
data = record
  param : array[1..num_param_group ] of param_group;
term : array[1..term_length ] of byte;
printr : array[1..printer_length ] of byte;
msg_dir : array[1..num_msg_dir ] of msg;
decode_dict : dict_buffer;
end;

wordtype = string[wordlength];
FILE: readmenu.pas

(*PL60*)
(*-----------------------------------------------------------------------------*
* * file: readmenu.pas *
* version: 1.3 *
* date: 3 September 84 *
* description: this file contains procedures which are part of the MICROSDW BUILDDAT program. *
* language: TURBO-Pascal *
* procedures contained: readmenu *
* author: Paul A. Moore, Capt, USAF *
*-----------------------------------------------------------------------------*)

(*-----------------------------------------------------------------------------*
* * procedure: readmenu *
* module number: 1.5.2 *
* version: 1.3 *
* date: 21 October 1984 *
* history: 8/16/84 created *
* 8/21/84 deleted variable "result" *
* 9/3/84 remove debug stmts *
* 10/21/84 change Input/Output *
* description: this procedure *
* 1.5.2 READ MENU DEFINITION (READMENU) - This process reads *
* records from the MENU.TXT file and passes them into a buffer *
* (parse_buf). The buffer is passed back to make_menu which *
* then invokes addtomenu to build the word_tree, menu_tree, and *
* call_list structures. *
FILE: readmenu.pas

* * inputs: menu_text
* * outputs: parse_buf
* * global variables used: Outfile, Outflag
* * global constants used: COMMENT, ENDMENU, MENUDEF,
* * CALLDEF, ENDDF, MENUWORD
* * modules called: get_word
* * calling modules: make_menu
* * files read: menu_text
* * files written: Outfile
* * author: Paul A. Moore, Capt, USAF
*
******************************************************************************

procedure readmenu ( var menu_text : text; var parse_buf : parse_record);
const
   NOTHING = '';

var
   word_line : menuline;
   word : wordtype;
   start_position : integer; (* character position to start at *)
   assignloc : integer; (* location in word_line of *)
                   (* assignment character *)
   buffpos : integer; (* location to start word in wordstr *)
   stop_position : integer; (* pos. of last char in word_line *)
   IOstatus : integer;

begin
   {$I-} { Turn off Automatic Input/Output Error checking }
   repeat
      readln(menu_text, word_line);
      IOstatus := IOresult;
      If Outflag and (IOstatus = 0) then
FILE: readmenu.pas

WRITEln(Outfile, word_line);
WRITEln(word_line); (* aid in locating errors *)
until (word_line[1] <> COMMENT) or (IOstatus <> 0);
{$I+}

if IOstatus <> 0 then (* error reading menu text file *)
    parse_buf.rectype := ENDMENU
else begin
    case word_line[1] of
        ':' : parse_buf.rectype := ENDMENU;
        '$' : parse_buf.rectype := MENUDEF;
        '.' : parse_buf.rectype := CALLDEF;
        '!' : parse_buf.rectype := ENDDIF;
    else parse_buf.rectype := MENUWORD;
    end; (* end case *)

    stop_position := length(word_line);
    parse_buf.wordstr := copy(NOTHING, 1, 0);

    (* debug *)
    (* WRITEln('parse_buf.rectype = ', parse_buf.rectype); *)

    if parse_buf.rectype <> ENDMENU then
        if parse_buf.rectype = MENUWORD then
            begin
                start_position := 1;
                get_word (word_line, start_position, stop_position, word);
                parse_buf.wordstr := word;
                parse_buf.startassign := 0; (* no assignment *)
                if start_position < stop_position then
                    WRITEln('>> extra characters after menu word <<');

FILE: readmenu.pas
end
else (* process menu definitions and call definitions *)
begin
parse_buf.startassign := 0;
start_position := 2; (* skip leading character *)
while start_position < stop_position do
begin
get_word(word_line,start_position,stop_position,word);
if word = ASSIGNMENT then
(* set index into wordstr where assignment *)
(* word(s) begin *)
parse_buf.startassign :=
length(parse_buf.wordstr) + 1
else
parse_buf.wordstr :=
concat(parse_buf.wordstr,word,BLANK);
end; (* end while *)
end; (* end processing of menu & call definitions *)

(* end if = MENUWORD *)
(* end if <> ENDMENU *)

parse_buf.length := length(parse_buf.wordstr);
end; (* end if IORESULT <> 0 else *)
FILE: addtomen.pas

(*--------------------------------------------------------------------------*
* file: addtomen.pas
* version: 1.0
* date: 16 August 1984
* description: this file contains procedures which
* are part of the MICROSDW BUILDDAT
* program.
* language: TURBO-Pascal
* procedures contained: addtomen
* author: Paul A. Moore, Capt, USAF

(*--------------------------------------------------------------------------*)

(*--------------------------------------------------------------------------*
* procedure: addtomen
* module number: 1.5.3
* version: 1.0
* date: 16 August 1984
* history: 8/16/84 created
* description: 1.5.3 ADDTOMENU- This process receives pointers to the
* parse buffer (parse_buf) and the menu data structures
* (word_tree, menu_tree, call_list). Addtomen calls the
* appropriate subprocesses based on the record type (rectype)
* of the parse buffer to add information in the buffer to the
* menu data structures.
* inputs: wordtree, menutree, call_list, parse_buf,
* node_counter, current_menu
* outputs: wordtree, menutree, call_list, parse_buf,

FILE: addtomen.pas
FILE: addtomen.pas

* node_counter, current_menu
* global variables used: none
* global constants used: MENUWORD, MENUDEF, CALLDEF,
  ENDDEF
* modules called: addword, addkeywordtomenu,
  define_menu, define_call
* calling modules: make_menu
* files read: none
* files written: none
* author: Paul A. Moore, Capt, USAF

*****************************************************************************

procedure addtomenu ( var wordtree : btptr; var menutree : mtptr;
  var call_list : callptr;
  var parse_buf : parserecord;
  var node_counter : integer;
  var current_menu : mtptr );

var
  word     : wordtype;
  keyword_pointer : btptr;
  level     : integer; (* debug *)
begin
  case parse_buf.rectype of
    MENUWORD : begin (* process menu word *)
      (* get keyword from parse_buf *)
      word := parse_buf.wordstr;
      addword(word, wordtree, keyword_pointer);
      node_counter := node_counter + 1;
      addkeywordtomenu(keyword_pointer, current_menu,
        menutree, node_counter);

FILE: addtomen.pas

Page: 2
FILE: addtomen.pas

end;

MENUDEF : begin
    define_menu(menutree, current_menu, parse_buf );
end;

CALLDEF : define_call Routine(menutree, wordtree, call_list,
    parse_buf );

ENDDEF : current_menu := nil

end; (* end case *)

end;

FILE: addtomen.pas
FILE: microsdw.pas

{SR+}  { enable range checking }  
(********************************************************************
*)  *
*) program:  MICROSDW  *
*) version:  5.10  *
*) date:  11/28/84  *
*) description: this program provides a flexible user interface for software development or other applications.  *
*) modules called: get_cmd  *
*) get_data  *
*) select_routine  *
*) history:  9/20/84 Converted to TURBO-Pascal  *
*) author:  vincent m. parisi ii, capt., usaf  *
*) modifier:  Paul A. Moore, Capt, USAF  *
*)  
)********************************************************************)

program MICROSDW;

{SI msdwcons.pas}
{SI msdwttype.pas}

const
  wordsize    = 12;
  buffersize  = 6;
  stat_line_width = 77;
  crt_only    = 'c';
  terminal_only = 't';
  as_assigned = 'a';
  backspace   = 8;
  del         = 127;

FILE: microsdw.pas
FILE: microsdw.pas

yes = true;
no = false;
abort_str = '$';

type
term_array = array[1..term_length] of byte;
print_array = array[1..printer_length] of byte;
msg_array = array[1..num_msg_dir] of msg;
buffer = array[1..buffersize] of string[wordsize];
cmdword = string[wordsize];
msg_line = string[screen_width];
dictionary = record
dictword : cmdword;
matchp : integer;
nomatchp : integer;
abbrev : byte; (* minimum length of abbreviation *)
end;

var

cmdbuffer : buffer; (* buffer of command words*)
blanks : string[screen_width];
status_line : string[stat_line_width];
call_routine : string;
abort_command : boolean;
trans : boolean;
printer : boolean;
temp : boolean;
crt : boolean;
macro_error : boolean;

FILE: microsdw.pas
FILE: microsdw.pas

show_abbreviation : boolean;
in_terminal : boolean;
stat_on : boolean;
macro_file : text;
trans_file : text;
list_dev : text;
temp_file : text;
real_error : byte;
help_level : byte;
term : term_array;
print : print_array;
msg_dir : msg_array;
decode_dict : dict_buffer;
msg_txt : file of msg_line;
string : msg_line;
decode : dictionary;
list_dev_name : paramstring;
trans_file_name : paramstring;
macro_file_name : paramstring;
number_of_commands : integer;

(******************************************************************************
* Include the sources for the routines called by MICROSDW *
******************************************************************************)

[${I ucase.pas }
[${I terminal.pas }
[${I output.pas }
[${I pause.pas }
[${I getdat.pas }

[${I msg.pas }

FILE: microsdw.pas
FILE: microsdw.pas

{$I instruct.pas}
{$I getline.pas}
{$I prompt.pas}
{$I promptcm.pas}

{$I trim.pas}
{$I display.pas}

{$I getinput.pas}
{$I getint.pas}
{$I getstring.pas}
{$I readcom.pas}

{$I processor.pas}
{$I valdec.pas}

{$I getcom.pas}
{$I help.pas}
{$I select.pas}

(* ******************************************************************************
 * main program code
 * *******************************************************************************)

begin    (* begin main program *)

(* ******************************************************************************
 * initialize the program; read in all the initializing
 * parameters, the command syntax data structure. put up
 * *******************************************************************************)

FILE: microsdw.pas
FILE: microsdw.pas

* title slide to show CRT interface is working and give *
* user something to look at. *
* Also initialize all files used by the MICROSDW. *
* ***********************************************

get_data( term, print, msg_dir, decode_dict, printer, 
trans, temp, crt, show_abbreviation, in_terminal, 
stat_on, macro_error, help_level, 
list_dev_name, trans_file_name, macro_file_name );

pause;

;/* begin main logic statements */
/* ***********************************************

repeat
    get_cmd( cmdbuffer, call_routine, number_of_commands );
    select_routine( call_routine, cmdbuffer, number_of_commands );
until (call_routine = 'STOP');

end. (* end of main program *)

FILE: microsdw.pas
FILE: displayc.pas

(*------------------------------------------------------------*)
(* module: display_commandword                               *)
(* version: 1.1                                               *)
(* date: 30 June 1984                                         *)
(* description: this module displays the commandword          *)
(* pointed to by word_num.                                   *)
(* procedures contained: display_commandword                 *)
(* author: vincent m. parisi ii, capt., usaf                 *)
(*------------------------------------------------------------*)

(*------------------------------------------------------------*)
(* procedure: displa_commandword                             *)
(* version: 1.1                                               *)
(* date: 30 June 1984                                         *)
(* description: this procedure displays the commandword       *)
(* pointed to by word_num.                                   *)
(* global variables used: none                               *)
(* global constants used: wordsize, buffersize               *)
(* modules called: outstring                                *)
(* called by: get_com                                        *)
(* author: vincent m. parisi ii, capt., usaf                 *)
(* modifier: Paul A. Moore, Capt, USAF                       *)
(*------------------------------------------------------------*)

procedure displa_commandword(cmdbuffer : buffer; word_num : integer );

var   cmd_word    : cmdword;

FILE: displayc.pas
begin
  cmd_word := cmdbuf[ word_num ];
  trim( cmd_word );
  out_string( cmd_word, 'a' );
  out_string( ' ', 'a' );
end;

FILE: displayc.pas
FILE: getcom.pas

(*---------------------------------------------------------------*
* module: get_com                                              *
* version: 3.1                                                 *
* date: 16 august 1983                                         *
* description: this module contains the procedure              *
* get_cmd which handles all processing                        *
* associated with getting a valid command from the user.      *
* it is called by the program and operation is maintained     *
* here until a decoded and validid com-mand is entered.       *
* procedures contained: get_cmd                                *
* author: vincent m. parisi ii, capt., usaf                   *
*---------------------------------------------------------------*)

(*---------------------------------------------------------------*
* declarations                                                *
*---------------------------------------------------------------*)

const
pr_cmd_row = 11;
pr_cmd_col = 5;
pr_hlp_row = 5;
cmd_row = 11;
cmd_col = 20;
instr_row = 2;
instr_col = 5;

(*---------------------------------------------------------------*
* FILE: getcom.pas                                              *
*---------------------------------------------------------------*)
procedure get_cmd
version: 3.1
date: 16 august 1983
description: this procedure handles all processing associated with getting a valid command from the user. It is called by the program and operation is maintained here until a decoded and validated command is entered.
global variables used: help_level
modules called: gotoxy
readcom
get_line
valnsec
prompt_help
display_commandword
prompt_cmd
instruction
process_error,
clear
called by: main
author: vincent m. parisii, capt., usaf

******************************************************************************

procedure get_cmd( var cmdbuffer : buffer; var call_routine : cmdword;
var num_of_commands : integer );

var i : integer;
level : integer;
rec_num : integer;
abort_command : boolean;

FILE: getcom.pas
FILE: getcom.pas

bufferpointer : integer;
error_code : char;

begin

abort_command := yes;  (* initialize so first level of command
words are displayed. *)
error_code := 'a';

repeat

if abort_command = yes then
begin
bufferpointer := 1;  (* set bufferpointer on initial entry and when
user wanted to abort previous command. *)
rec_num := 1;
get_line( decode, rec_num );  (* get first dictionary entry *)
level := 1;  (* initialize for instruction *)
(* and valdec *)
end;
clear;  (* clear the screen of everything *)

if help_level = 3 then
(* issue instructions based on help level *)
instruction( level, instr_row, instr_col );

if help_level > 1 then
prompt_help( rec_num, pr_hlp_row );  (* display the appropriate command *)
(* words based on pointer(s) in decode *)
prompt_cmd( pr_cmd_row, pr_cmd_col );  (* put up the logo *)
FILE: getcom.pas

(* display the contents of the command buffer, if it is empty, nothing *)
(* will be displayed, but if there are some commands in it form an unre-*
(* solved command, then display them.

goxy( cmd_row, cmd_col );        (* position for command  *)

for i := 1 to ( bufferpointer - 1 ) do
    displa_commandword( cmdbuffer, i );

abort_command := no;     (* set abort command for read command  *)

(* get command from user  *)
num_of_commands := 0;
readcom( cmdbuffer, bufferpointer, abort_command );

if (( abort_command = no ) and ( bufferpointer > 1 )) then
    begin
        (* set parameters for entry into vali-  *)
        (* date and decode (val_n_dec)  *)
        num_of_commands := ( bufferpointer - 1 );
        rec_num := 1;
        level := 1;     (* enter in with first commandword  *)
        error_code := 'a';     (* initial code, not really an error  *)

        (* now decode the entered command  *)
        val_n_dec( level, rec_num, error_code, num_of_commands, cmdbuffer,  
              call Routine );

        if ( error_code = 'b' ) then
            begin
FILE: getcom.pas

if help_level > 1 then
  begin
    gotoxy( cmd_row, cmd_col );
    proces_error( error_code, level, cmdbuffer, bufferpointer );
  end;
  bufferpointer := level;   (* get word that is bad   *)
end;

end;

until (error_code = 'n') or (error_code = 'c');
end;
FILE:   getdat.pas

{.PL60}

(file: getdat.pas

procedure contained: get_data, bld_stat_line, title_slide

version:  1.1

date:     28 November 84

description: this module reads the data file from the disk

and inserts it into the appropriate tables

in the main program. It also initializes the

flags and help_level. Displays title slide.

global variables used: none

global variables changed: none

global constants used: none

modules called: none

called by: main

author: vincent m. parisi ii, capt., usaf

******************************************************************************

(*****************************************************************************

procedure: title_slide

version:  2.0

date:     18 oct 83

description: this procedure displays the system title

slide. By this, it demonstrates that at

least the terminal control codes have been

initialized properly, and probably the

rest of the parameters.

global variables used: none

global constants used: crt_only

******************************************************************************

FILE:   getdat.pas

Page: 1
FILE: getdat.pas

* procedure called: clear, gotoxy, highlight, nohighlight *
  * Rectangle, outstring *
  * called by: get_dat *
  * author: vincent m. parisi ii, capt., usaf *
  *
*******************************************************************************

procedure title_slide( var term_dat : term_array );
var row : integer;
begin
  clear;
  Rectangle(5,5,71,8); (* draw title box *)
  Rectangle(16,40,31,6); (* draw author box *)

  highlight;
  for row := 6 to 11 do
    begin
      gotoxy( row, 7 );
      out_string( copy(blanks, 1,67 ), crt_only );
      end;

  gotoxy( 8, 18 );
  out_string( 'Microcomputer Software Development Workbench', crt_only );

  nohighlight;

  gotoxy( 17, 42 );
  out_string( 'Written by:', crt_only );
  gotoxy( 18, 44 );
  out_string( 'Paul A. Moore', crt_only );
  gotoxy( 19, 44 );
FILE: getdat.pas

out_string( 'advisor: Professor', crt_only );
gotoxy( 20, 44 );
out_string( 'Gary B. Lamont', crt_only );
end;

(*.PA*)

********************************************************************************
procedure:  bld_stat_line
version: 1.2
date: 18 oct 83
description: this module builds the status line from
initialization data from disk storage.
data is in param group one.
global variables used: stat_line
global variables changed: stat_line
global constants used: none
procedure called: none
called by: get_dat
author: vincent m. parisii, capt., usaf
********************************************************************************

procedure bld_stat_line
( help_level : integer; temp : boolean; printer : boolean;
  trans : boolean );
var help : char;
on_off : string[ 3 ];
begin
  status_line := concat( status_line, ' Help level = ' );
  if help_level = 3 then help := '3'
  status_line := concat( status_line, help );
end;
FILE: getdat.pas

else
  if help_level = 2 then help := '2'
else
  help := '1';
status_line := concat( status_line, help );

status_line := concat( status_line, ' Echo Print - ' );
if printer then on_off := 'ON ' 
else
  on_off := 'OFF';
status_line := concat( status_line, on_off );

status_line := concat( status_line, ' Transaction copy - ' );
if trans then on_off := 'ON ' 
else
  on_off := 'OFF';
status_line := concat( status_line, on_off );

status_line := concat( status_line, ' Temporary - ' );
if temp then on_off := 'ON ' 
else
  on_off := 'OFF';
status_line := concat( status_line, on_off );
end;

(*.PA*)

(*-------:-------------------------------------------------------------------*)
* 
*       procedure:           get_data
*       version:            2.0
*       date:               28 Nov 84
*       description:        this procedure reads the data.dat file and

FILE: getdat.pas
FILE: getdat.pas

* initializes the program variables passed to it. Also calls title_slide and build_status_line.
* global variables used: blanks, call_routine, status_line
* global variables changed: blanks, call_routine, status_line
* global constants used: term_length, screen_width, printer_length, num_msg_dir, num_prts, num_words
* procedure called: clear, title_slide, bld_stat_line
* called by: main
* author: vincent m. parisi ii, capt., usaf
*
********************************************************************************

procedure get_data( var term_dat : term_array;
var print_dat : print_array;
var msg_dir : msg_array;
var decode_dict : dict_buffer;
var printer : boolean;
var trans : boolean;
var temp : boolean;
var crt : boolean;
var show_abbreviation : boolean;
var in_terminal : boolean;
var stat_on : boolean;
var macro_error : boolean;
var help_level : byte;
var list_dev_name : paramstring;
var trans_file_name : paramstring;
var macro_file_name : paramstring);

type datarecord = record

FILE: getdat.pas
FILE: getdat.pas

var
data_file : file of data;
data_recs : dataptr;

begin
  writeln('Initializing the MICROSDW Menu Structure',
          'and Hardware configuration');

  assign( data_file, 'MICROSDW.sys' );
  reset( data_file );

  new(data_recs);
  read( data_file, data_recs^.tdata );
  close( data_file );

  move( data_recs^.tdata, term, term_dat, ( term_length + term_length ));

  blanks := '';
  callRoutine := '';

  for i := 1 to screen_width do
    blanks := concat( blanks, '' );

  status_line := '';
  stat_on := false;
FILE:      getdat.pas

title_slide( term_dat );

(* ----------------- Transfer the rest of the data to Global Storage areas *)
for i := 1 to printer_length do
  print_dat[ i ] := data_recs^.tdata.pintr[ i ];

for i := 1 to num_msg_dir do
  begin
    msg_dir[ i ].loc_rec := data_recs^.tdata.msg_dir[ i ].loc_rec;
    msg_dir[ i ].length := data_recs^.tdata.msg_dir[ i ].length;
  end;

for i := 1 to num_ptrs do
  begin
    decode_dict.ptrs[ i, 1 ] := data_recs^.tdata.decode_dict.ptrs[ i, 1 ];
    decode_dict.ptrs[ i, 2 ] := data_recs^.tdata.decode_dict.ptrs[ i, 2 ];
    decode_dict.ptrs[ i, 3 ] := data_recs^.tdata.decode_dict.ptrs[ i, 3 ];
  end;

for i := 0 to num_words do
  begin
    decode_dict.words[ i ] := data_recs^.tdata.decode_dict.words[ i ];
    decode_dict.abbrev[ i ] := data_recs^.tdata.decode_dict.abbrev[ i ];
  end;

with data_recs^.tdata do
  begin
    printer := param[1].bools[1];
    trans := param[1].bools[2];
    temp := param[1].bools[3];
    crt := param[1].bools[4];
  end;

FILE:      getdat.pas
FILE: getdat.pas

show_abbreviation := param[1].bools[5];
in_terminal := param[1].bools[6];
stat_on := param[1].bools[7];
macro_error := param[1].bools[8];
help_level := param[1].ints[1];
list_dev_name := param[1].strings[1];
trans_file_name := param[1].strings[2];
macro_file_name := param[1].strings[3];
end;

(* dispose of the temporary "data_recs" storage *)
dispose(data_recs);

(* build the status line to be shown after every pause and clear*)
bld_stat_line( help_level, temp, printer, trans );

[*******************************************************************************
* now open other files used in this system so they are ready for use.
*******************************************************************************]
assign( list_dev, list_dev_name );
rewrite( list_dev );

assign( trans_file, trans_file_name );
if trans then rewrite( trans_file );

rewrite( temp_file );

assign( macro_file, macro_file_name );
reset( macro_file );

FILE: getdat.pas
FILE: getdat.pas

assign( msg_txt, 'help.sys' );
reset( msg_txt );

end;
FILE: getinput.pas

(* *******************************************************
 * module: get_in
 * procedure: get_inp
 * version: 2.0
 * date: 28 july 1983
 * description: gets ascii input from keyboard and puts it in a string called cmd. if an abort command character is entered the routine exits with abort_command set true.
 * global variables used: none
 * global constants used: none
 * modules called: none
 * called by: readcom
 * author: vincent m. parisi ii, capt., usaf
 *
******************************************************* *)

procedure get_inp( var cmd : msg_line; var abort_command : boolean );
label exit;

var ch : char;

begin
abort_command := false; (* initialize for this input *)
while abort_command = false do
begin
read( ch );
if eoln = true then goto exit;
if (( ch <> '$' ) and ( ord(ch) > 31 )) then
FILE: getinput.pas

    cmd := concat(cmd, ch )
    else
        if ch = '$' then abort_command := true;
        end;
    exit:
    end;
end;
FILE: getint.pas

(* ------------------------------------------------------------- *
  * module: getint.mod                                         *
  * procedure contained: del_1st_chr                           *
  *                ck_chr                                        *
  *                out_int                                       *
  * function contained: getchi                                *
  * version: 1.2                                               *
  * date: 18 Oct 83                                            *
  * description: this module contains procedures that handle I/O for integers. Since normal program operation does not handle integer error exceptions very well, these routines do the job. *
  * author: vincent m. parisi ii, capt., usaf                 *
  * ------------------------------------------------------------- *)

(* ------------------------------------------------------------- *
  * procedure: getchi                                          *
  * version: 1.0                                               *
  * date: 18 Oct 83                                            *
  * description: this program gets one character from a string and returns it to the read function for conversion. Routine gotten from the Pascal MT+ instruction manual section on redirected I/O. *
  * global variables used: strng                               *
  * global variables changed: strng                            *
  * global constants used: none                                *
  * modules called: none                                       *
  * called by: get_int                                         *
  * get_real                                                  *
  * ------------------------------------------------------------- *)
function getchi : char;
begin
    if length( strng ) > 0 then
        begin
            getchi := strng[ 1 ];
            delete( strng, 1, 1 );
        end
    else
        getchi := ' ';
end;

procedure del_lst_ch;
begin
    write( chr( backspace ) );
    write( ' ' );
    write( chr( backspace ) );
end;

*****************************************************************************

procedure:    del_lst_ch
version:       1.0
date:          18 oct 83
description:   this procedure deletes the last character from the CRT*

global constants used:  backspace
modules called:  none
FILE: getint.pas

* called by: ck_chr
* author: vincent m. parisi ii, capt., usaf
*
***********************************************************************)

procedure ck_chr( ch : char; var strng : msg_line );
begin
  if ((( ord( ch ) = del ) or ( ord( ch ) = backspace )) and ( length( strng ) > 0 )) th
    delete( strnq, length( strng ), 1 );
  if ord( ch ) = del then
    del_1st_ch;
end;

***********************************************************************)
* *
* procedure: ck_chr
* version: 1.2
* date: 18 oct 83
* description: this procedure checks each character input to see if it is a delete or backspace. if it is, the screen is updated appropriately and the destination string is changed.
* *
* global variables used: strng
* global variables changed: strng
* global constants used: del, backspace
* procedures called: del_1st_ch
* called by: get_int
* author: vincent m. parisi ii, capt., usaf
***********************************************************************)
procedure out_int( number : integer; field : integer; dest : char );

begin

  case dest of
    'C', 'c' : write( number:field );
    'P', 'p' : writeln( number:field );
    'B', 'b' :
      begin
        write( number:field );
        writeln( number:field );
      end;
    'A', 'a' :
      begin
        if crt then write( number:field );
        if trans then writeln( trans_file, number:field );
  end;

FILE:    getint.pas
FILE:  getint.pas

if printer then
  writeln( list_dev, number:field );
if temp then
  writeln( temp_file, number:field );
end;
end;

******************************************************************************

* procedure:       get_int
* version:         1.3
* date:            18 oct 83
* description:     this procedure handles the input of integers.
                    Normal program integer input does not have edit
                    capability to exclude inputs such as letters for
                    integers. It just burps and asks for the number
                    again which messes up a formatted display.
                    This procedure does not accept
                    anything but valid integer constructs.
* global variables used: string, abort_command
* global variables changed: string, abort_command
* global constants used: as_assigned
* procedure called:    get_string, getchi
* called by:           get_tf
* author:              vincent m. parisi ii, capt., usaf

******************************************************************************

(* forward reference to get_string *)

procedure get_string(var string : msg_line; var abort_command : boolean;
in_dev : char; chr1, chr2 : char); forward;

FILE:  getint.pas
FILE: getint.pas

procedure get_int( var number : integer; var abort_command : boolean );
var   ch : char;
    result : integer;
begin

  number := 0;
  get_strng( strng, abort_command, as_assigned, '0', '9' );

  (* Val converts a string to a number, result = 0 if no errors, else *)
  (* result is the position of the invalid character in the string *)
  if not abort_command then
    Val(strng,number,result);

end;
FILE: getline.pas

procedure getline;
begin
  version: 3.0
  date: 17 July 1983
  description: this procedure builds a decoded entry from the record pointed to by the first pointer of dict
  global variables used: dict_entry, dict, buffer, wordsize
  global constants used: buffersize, dict-file (external), capt, usaf

    * * * * *

files used:
  dict-file (external)
  capt, usaf

*** FILE: getline.pas ***

module contained: getline
version: 3.1
date: 31 July 1984
description: this module contains the procedure that builds a decoded entry from the record pointed to by the first pointer of dict

*** FILE: getline.pas ***

Page: 1
procedure get_line( var decode : dictionary; rec_num : integer );
var dlen : integer;
begin
  decode.dictword := decode_dict.words[ decode_dict.ptrs[ rec_num, 1 ] ];
  decode.abbrev := decode_dict.abbrev[decode_dict.ptrs[ rec_num, 1 ] ];
  decode.matchp := decode_dict.ptrs[ rec_num, 2 ];
  decode.nomatchp := decode_dict.ptrs[ rec_num, 3 ];

  (* blank pad "decode.dictword*"
  dlen := length(decode.dictword);
  Insert(blanks,decode.dictword,dlen+1);
end;
FILE: getstrin.pas

* modules called: none
* called by: readcom
* author: vincent m. parisi ii, capt., usaf
*
*******************************************************************************)

procedure get_strng;
(* arguments are specified in an earlier "forward" reference
 ( var strng : msg_line; var abort_command : boolean;
   in_dev : char; chr1, chr2 : char );
*)
begin
  strng := ''; (* clear the string *)
  abort_command := false;

  case in_dev of
  'A','a': if in_terminal then
    begin
      BufLen := screen_width;
      read( strng );
    end
  else
    begin
      BufLen := screen_width;
      read( macro_file, strng );
    end
  'M','m': read( macro_file, strng );
  'T','t': begin
    BufLen := screen_width;
    read( strng );
  end
  else
    begin
      BufLen := screen_width;
    end
  end;
end;

FILE: getstrin.pas
FILE: getstrin.pas

read( strng );
end;
end; (* end case *)

if (length(strng) = 1) and (strng[1] = abort_str) then
abort_command := true;

end;
FILE: help.pas

(******************************************************************************* *
* module: help *
* procedure contained: help *
* version: 1.2 *
* date: 12 sept 1983 *
* description: this module contains the procedure that handles the logic for providing on line help. *
* global variables used: none *
* global constants used: buffersize, wordsize *
* author: vincent m. parisi ii, capt., usaf *
*******************************************************************************/)

(******************************************************************************* *
* procedure: help *
* version: 1.2 *
* date: 12 sept 1983 *
* description: this procedure handles the logic for providing on line help. The valid command is scanned to determine what help is requested. The display message routine is then called with the number of the help message requested. *
* global variables used: none *
* global constants used: wordsize, buffersize *
* modules called: pause, clear, disp_msg, trim *
* called by: select *
* author: vincent m. parisi ii, capt., usaf *
*******************************************************************************/)
FILE:  help.pas

procedure help( var cmdbufuffer : buffer; wordnumber : integer );

const
    system_msg    = 15;
    command_msg   = 16;
    lifecycle_msg = 18;
    wordproc_msg  = 19;
    database_msg  = 20;
    files_msg     = 21;
    run_msg       = 22;
    mediamsg      = 24;

var
    help_obj : cmdword;

begin
    help_obj := cmdbufuffer[ wordnumber ];
    if help_obj = 'MICROSDM' then disp_msg( system_msg )
    else if help_obj = 'Setup' then disp_msg( command_msg )
    else if help_obj = 'COMANDS' then disp_msg( lifecycle_msg )
    else if help_obj = 'LifeCycle' then disp_msg( wordproc_msg )
    else if help_obj = 'Wordproc' then disp_msg( files_msg )
FILE: help.pas

else
  if help_obj = 'Communic' then disp_msg( Communic_msg )
else
  if help_obj = 'Database' then disp_msg( Database_msg )
else
  if help_obj = 'Files' then disp_msg( Files_msg )
else
  if help_obj = 'Run' then disp_msg( Run_msg )
else
  if help_obj = 'MediaMas' then disp_msg( MediaMas_msg );

pause;
clear;
end;

FILE: help.pas

Page: 3
procedure instruction
    ( level : integer; instr_row : integer; instr_col : integer );

begin

    if level = 1 then
        out_string( 'Enter one of the following initial command words...', 'c' )
    else
        out_string
            ( 'Now enter one of these commandwords, or "$" to abort...', 'c' );

end;
procedure disp_line( rec_num : integer );
FILE: msg.pas

var
  print_line : string[screen_width];
begin
  seek( msg_txt, rec_num );
  read( msg_txt, print_line );
  out_string( print_line, as_assigned );
  writeln;
end;

(*--------------------------------------------------------------------------

  procedure:  clear_msg
  version:    1.3
  date:       18 oct 83
  description: this procedure clears the message indicated by
               msg_dnum the screen. It is the programmer's
               responsibility to position the cursor prior
               to calling this routine.
               cursor is positioned at the beginning of the line
               where the message is.
               for writing the message.
  global variables used:  msg_dir, blanks
  global variables changed: none
  global constants used:  crt_only
  procedure called:  out_string
  called by:          
  author:             vincent m. parisi ii, capt., usaf

--------------------------------------------------------------------------*)
procedure clear_msg( msg_num : integer );

var  i   :  integer;
    length :  integer;

begin

    length := msg_dir[ msg_num ].length;
    if length > 23 then
        clear
    else
        for i := 1 to length do
            begin
                out_string( blanks, rt_only );
                writeln;
            end;

end;

(*--------------------------------------------------------------------------
 * procedure: disp_msg
 * version: 2.0
 * date: 18 oct 83
 * description: this procedure displays the message pointed to by
 * parameter passed in, msg_num. The message is displayed at the current
 * cursor position. If the message length is longer that 23 lines, the display
 * stops after showing 22 lines and waits for the user to
 * indicate 'continue' with a <CR>. If a '$' is rcvd
 * the procedure is exited and return to the calling
 * routine.
 * global variables used: msg_dir,
*)
FILE: msg.pas

* global constants used: none
* procedure called: disp_line, gotoxy, clear, clear_msg
* called by: vincent m. parisi ii, capt., usaf

-----------------------------------------------
procedure disp_msg( msg_num : integer );
label exit;
var i : integer;
length : integer;
rec_num : integer;
remain_lines : integer;
resp : char;

begin

length := msg_dir[ msg_num ].length;
rec_num := msg_dir[ msg_num ].loc_rec - 1;

if length < 23 then
  for i := 0 to ( length - 1 ) do
    disp_line(( rec_num + i ))
else
  begin
    remain_lines := length;
    while remain_lines > 21 do
      

FILE: msg.pas

begin
  clear;
gotoxy( 0, 0 );
for i := 0 to 21 do
  disp_line( (rec_num + i) );
remain_lines := remain_lines - 22;
rec_num := rec_num + 21;
gotoxy( 22, 0 );
disp_msg( 13 );
read( resp );
gotoxy( 22, 0 );
clear_msg( 13 );
if resp = '$' then goto exit;
gotoxy( 22, 0 );
end;
for i := 0 to remain_lines do
  disp_line( (rec_num + i) ); (* records on disk begin at 0 *)
end;
exit:
end;
FILE: output.pas

(******************************************************************************
 * module:          output
 * procedure contained:  out_string
 * version:          1.0
 * date:             01 august 1983
 * description:      this module contains the procedure that
                     handles all output to user.
 * global variables used:  trans
 *                        printer
 *                        temp
 * global variables changed:  none
 * global constants used:   none
 * author:            vincent m. parisi ii, capt., usaf

******************************************************************************)

(******************************************************************************
 * procedure:        out_string
 * version:          1.0
 * date:             01 august 1983
 * description:      this procedure handles all string
                     output for the program. whenever
                     system output is required, this
                     module is called. the output is direc-
                     ted to the appropriate device.
 * the devices that can accept output are:
 *   crt-- generally all interaction
 *   printer-- for hard copy either dyn-
 *              amically or selectively.
 *   transaction file--file which contains

FILE: output.pas

Page: 1
FILE: output.pas

* all user interactions for session*
* tracing.
* temporary file--so user can review
* work just accomplished.
* output can be directed specifically
* to the crt only, the printer only, both
* or as indicated by the boolean switches
* discussed below.
* files used: trans_file
* temp_file,
* list_dev
* global variables used: trans  -- boolean which indi-
* cates if output should go
* to transaction file.
* printer  -- boolean which indicates
* that all output is echoed to
* the printer.
* temp  -- boolean which indi-
* cates if output should go to
* temporary file for quick rev-
* iew purposes.
* note: all file accesses are
* done globally.
* global variables changed: none
* global constants used: none
* modules called: none
* called by: quite a few!
* author: vincent m. parisi ii, capt., usaf

******************

procedure out_string( ostring : msg_line; dest : char );

FILE: output.pas
FILE: output.pas

begin

case dest of
  'C', 'c' : write( ostring );
  'P', 'p' : writeln( list_dev, ostring );
  'B', 'b' : begin
              write( ostring );
              writeln( list_dev, ostring );
            end;
  'A', 'a' : begin
              if crt then write( ostring );
              if trans then writeln( trans_file, ostring );
              if printer then writeln( list_dev, ostring );
              if temp then writeln( temp_file, ostring );
            end;
end;
end;
end;
FILE: pause.pas

******************************************************************************
* file:          pause.pas *
* version:       1.3 *
* date:          29 October 1984 *
* description:   this module contains the procedure *
*                that waits for user response to continue *
*                anytime there is a stop in the program. *
*                author:    vincent m. parisi ii, capt., usaf *
******************************************************************************

******************************************************************************
* procedure:     pause *
* version:       1.1 *
* date:          29 October 1984 *
* description:   this procedure waits for user response *
*                to continue anytime there is a stop in *
*                the program. If the user has selected *
*                the status line on, then it is displayed *
*                otherwise it is not. *
* global variables used:    blanks, *
*                           status_line_on *
*                           stat_on *
* global constants used:    screen_width, crt_only, *
*                           stat_line_width *
* procedures called:    gotoxy, highlight, nohighlight, *
*                        out_string *
* called by:             *
* author:                vincent m. parisi ii, capt., usaf *
* modifier:              Paul A. Moore, Capt, USAF
procedure pause;
var resp : char;
begin
  gotoxy( 22, 0);
  out_string(blanks, crt_only);
  gotoxy( 22, 20);
  highlight;
  out_string( ' >>>> Press <CR> Key to continue... <<< ' , crt_only );
  nohighlight;
  read( resp );
  if stat_on then
    begin
      gotoxy(22,0);
      out_string(status_line, crt_only);
    end
  else
    out_string( blanks, crt_only );
end;
FILE: proceser.pas

(*---------------------------------------------------------------*
 * module: proceser
 * procedure: proces_error
 * version: 1.1
 * date: 16 august 1983
 * description: this module contains the procedure that handles
 * command decoding errors. it prompts the user for
 * proper action to take for error correction.
 * global variables used: help_level,
 *                         blanks
 * global constants used: screen_width,
 * author: vincent m. parisi ii, capt., usaf
 *---------------------------------------------------------------*)

(*---------------------------------------------------------------*
 * procedure: proces_error
 * version: 1.1
 * date: 16 august 1983
 * description: this procedure handles command decoding errors. it
 * prompts the user for proper action to take for error
 * correction.
 * global variables used: help_level,
 *                         blanks
 * global constants used: screen_width,
 * modules called: gotoxy,
 *                  pause,
 *                  display_commandword,
 *                  highlight,
 *                  nohighlight,
 *---------------------------------------------------------------*)
FILE: proceser.pas

* disp_msg,
* clear_msg
* called by: get_cmd
* author: vincent m. parisi ii, capt., usaf

******************************************************************************

procedure proces_error( error_code : char; level : integer;
cmdbuffer : buffer; bufferpointer : integer );

var i : integer;

begin
  if help_level > 1 then
    begin
      case error_code of
        'B', 'b' : begin
        for i := 1 to ( level - 1 ) do
          displa_commandword( cmdbuffer, i );
        highlight;
        displa_commandword( cmdbuffer, level );
        nohighlight;
        for i := ( level + 1 ) to bufferpointer do
          displa_commandword( cmdbuffer, i );
        end;
        'C', 'c' : begin
        highlight;
        for i := 1 to ( level - 1 ) do
          displa_commandword( cmdbuffer, i );
        nohighlight;
        for i := level to bufferpointer do
          displa_commandword( cmdbuffer, i );
        end;
      end;
    end;
  end;
end;
FILE: proceser.pas

    display_commandword( cmdbuffer, i );
    end;
end;

if help_level > 2 then
    begin
        gotoxy( 20, 0 );
        case error_code of
            'B', 'b' : begin
                disp_msg( 4 );
                pause;
            end;

            'C', 'c' : begin
                disp_msg( 5 );
                pause;
            end;
        end;
    end;
end;

FILE: proceser.pas
procedure prompt_cmd( row : integer; col : integer );
const  logo = 'Enter Option >';
FILE: promptcm.pas

begin
  gotoxy( row, 0 );
  out_string( blanks, crt_only );
  gotoxy( row, col );
  highlight;
  out_string( logo, as_assigned );
  nohighlight;
end;
FILE: prompthel.pas

(*------------------------------------------------------------------*)
(*
* file:            prompthel.pas
* contains procedure: prompthel_helper
* version: 2.2
* date: 27 September 84
* description: this module contains the procedures that
*               display acceptable command words based
*               upon the words already entered.
* global variables used: none
* global constants used:
*   prompt_col_offset, ENCODE, DONEWORD, crt_only
* author: vincent m. parisi ii, capt., usaf
*)

const prompt_col_offset = 5;

(*------------------------------------------------------------------*)
(*
* procedure: prompthel_helper
* version: 2.1
* date: 20 October 1983
* description: this procedure displays the acceptable
*               command words based upon the words
*               already entered.
* global variables used: none
* global constants used: prompt_col_offset,
*   ENCODE, DONEWORD, crt_only
* modules called: get_line, gotoxy, out_string
* called by: SVideoLow, SVideoLow
*)

FILE: prompthel.pas
FILE: prompthe.pas

* author: vincent m. parisi ii, capt., usaf *
*
******************************************************************************)

procedure prompt_help( rec_num : integer; row : integer );
var
  row_count : integer;
  j : integer;
  decode : dictionary;
  displayword : msg_line;
begin
  row_count := 0;
  repeat
    gotoxy( ( row + row_count ), prompt_col_offset );
    out_string( ' ', crt_only );
    j := 1;
    repeat
      get_line( decode, rec_num );
      if decode.dictword <> DONEWORD then
        begin
          displayword := decode.dictword;
          SVideoLow( displayword, decode.abbrev+1 );
          SVideoBold( displayword, 1 );
          out_string( displayword, crt_only );
          j := j + 1;
        end;
      rec_num := decode.nomatchp;
      until (( j = 6 ) or ( decode.nomatchp = ENDCODE ));
    row_count := row_count + 1;
end.
until (( row_count = 6 ) or ( decode.nomatchp = ENDCODE ));
end;

```
FILE: readcom.pas

(*-----------------------------------------------*
* module: rd_cmd
* procedure contained: readcom
* version: 1.1
* date: 28 August 1983
* description: this module contains the procedure
* that gets a command line from the user.
* global variables used: cmd- input command
* cmdbuffer- buffer of command-words
* bufferpointer- position in
* command buffer.
* global variables changed: same as used
* modules called: trim,
* ucase,
* called by: get_cmd,
* author: Vincent M. Parisi II, Capt., USAF
* *)

(*-----------------------------------------------*)

procedure: readcom
* version: 1.1
* date: 28 August 1983
* description: reads command from user and splits it
* into individual words in the command
* buffer.
* global variables used: cmd- input command
* cmdbuffer- buffer of command-words
* *)

FILE: readcom.pas
procedure readcom(var cmdbuffer : buffer; var bufferpointer : integer;
    var abort_command : boolean);

var i    : integer;
j      : integer;
tword  : msg_line;
lencmd : integer;

begin    (* main body of procedure readcom *)

for i := bufferpointer to buffersize do         (* clear cmdbuffer not used to *)
    cmdbuffer[ i ] := copy( blanks, l, wordsize );  (* spaces *)

(* get the command from either the macro file or the terminal as assigned
if there is an error while in macro, then get input from terminal *)

    if macro_error then
        get_strng( strng, abort_command, terminal_only, ' ', ' ' )
    else
        get_strng( strng, abort_command, as_assigned, ' ', ' ' );

(* if there is no abort command in the string, process the command string.

FILE:     readcom.pas

Page: 2
FILE: readcom.pas

* break up the command string into individual words and put each word in
* the command buffer, left justified with right filled spaces. return with
* bufferpointer at next free command buffer position. *)

if abort_command = no then
begin
  ucase ( strng ); (* change entire command to upper case *)
  lencmd := length( strng );
  i := 1;
  repeat
    while ((strng[i] = ' ') and ( i <= lencmd )) do
      i := i + 1;
    if (( strng[i] > ' ') and ( i <= lencmd )) then
      begin
        j := 0;
        while (( strng[j + i] <> ' ') and ( (i + j) <= lencmd)) do
          j := j + 1;
        tword := copy( strng, i, j );
        tword := concat(tword, ' '); { 
        cmdbuffer[bufferpointer] := copy(tword,1,12);
        bufferpointer := bufferpointer + 1;
        i := i + j;
      end;
      until i >= lencmd;
    end;
  end; /* end of procedure readcom */
FILE: select.pas

(************************************************************
 * file: select.pas
 * procedure contained: select_routine
 * version: 2.0
 * date: 30 October 1984
 * description: this module contains the procedure that
  receives the name of the routine to call
  and calls it.
 * global variables used: none
 * global variables changed: none
 * global constants used: wordsize, buffesize
 * modules called: trim, help
 * called by: main
 * author: Paul A. Moore, Capt, USAF
 ************************************************************)

(************************************************************
 * procedure select_routine
 * version: 2.0
 * date: 30 October 1984
 * description: this procedure receives the name of the
  routine to call and calls it.
 * global variables used: none
 * global variables changed: none
 * global constants used: wordsize, buffesize
 * modules called: trim, help
 * called by: main

FILE: select.pas

Page: 1
FILE: select.pas

* author: Paul A. Moore, Capt, USAF *

*****************************************************************

procedure select_routine(var call_routine: cmdword;
                        var cmdbuffer: buffer;
                        number_of_commands: integer);

var i: integer;

begin

  trim(call_routine);

  if call_routine = 'Help' then
    help(cmdbuffer, number_of_commands)
  else if call_routine <> 'STOP' then
    begin
      (* put in a case statement to call the appropriate routines in *
       the MICROSDW system *)
      writeln;
      writeln('SELECT: ', call_routine);
      i := 1;
      while (cmdbuffer[i] <> ' ') and (i<=buffersize) do
        begin
          write(cmdbuffer[i]);
          i := i + 1;
        end;

      pause;
    end;
end;

FILE: select.pas
FILE: terminal.pas

(*********************************************************************)
*
* module:         terminal
* procedure contained: clear, gotoxy, highlight,
*                   nohighlight, graphics, nographics,
*                   VideoLow, SVideoLow, VideoBold, SVideoBold
* version:        2.0
* date:           21 oct 83
* description:    this file contains the procedures that
*                 interface with the terminal.
* global variables used: term,
*                        stat_on,
*                        stat_line
* global constants used: term_length,
*                        stat_line_width
* author:          vincent m. parisi ii, capt., usaf

*********************************************************************)

(FILE: terminal.pas

(*********************************************************************)
*
* procedure:      graphics
* version:        2.0
* date:           21 oct 83
* description:    this procedure places the terminal in graphics
*                 mode.
* global variables used: term,
* global constants used: term_length,
* procedures called: none
* called by: many
* author:         vincent m. parisi ii, capt., usaf

*********************************************************************)
procedure graphics;

var i : integer;

begin
  for i := 41 to ( term[ 40 ] + 40 ) do
    write( chr( term[ i ] ) );
end;

procedure nographics;

var i : integer;

begin
  for i := 48 to ( term[ 47 ] + 47 ) do
    write( chr( term[ i ] ) );

FILE: terminal.pas
FILE: terminal.pas

end;

****************************************************************************
*
* procedure:            highlight
* version:              2.0
* date:                 21 Oct 83
* description:          this procedure puts the screen in reverse video
* global variables used: term,
* global constants used: term_length,
* procedures called:    none
* called by:            many
* author:               Vincent M. Parisi II, Capt., USAF
****************************************************************************

procedure highlight;

var i: integer;

begin
  for i := 28 to (term[27]+27) do
    write(chr(term[i]));
end;

****************************************************************************
*  
* procedure:            nohighlight
* version:              2.0
* date:                 21 Oct 83
* description:          this procedure takes the screen out of reverse video

FILE: terminal.pas
procedure nohighlight;

var i : integer;

begin
  for i := 35 to (term[34] + 34) do
    write(chr(term[i]));
end;

(* ***********************************************

  procedure:    gotoxy
  version:      2.0
  date:         21 oct 83
  description:  this procedure places the cursor at the x and y
                coordinates passed to it. It is capable of
                sending an initial string of characters, either
                the row or column (whichever is required first)
                then an intermediate string of characters, the
                other address, and finally a trailing string
                of characters if required. Offsets if any
                are added prior to sending the row/column.

  global variables used:    term,
  global constants used:    term_length,

  ***********************************************
*)
procedure gotoxy( row : integer; col : integer );

var   i   : integer;

procedure ttype(switch:integer; wchar : integer); (* terminal type *)
begin
  if switch = 0 then
    write ( chr(wchar) ) (* H19/29 terminal *)
  else
    write( wchar ) (* VT100 terminal *)
end;

begin

  row := row + term[ 7 ]; (* add row and col offsets *)
  col := col + term[ 8 ];

  (* send out initial string *)
  for i := 2 to (term[ 1 ] + 1) do
    write( chr( term[ i ] ));

  if term[ 9 ] = 0 then (* if = 0 then row goes first, else *)
    ttype(term[90],row)
FILE:    terminal.pas

else
  ttype(term[90],col);

(* send out intermediate *)
for i := 11 to ( term[10] + 10 ) do
  write( chr( term[ i ] ) );
  (* string if any *)

if term[9] = 0 then
  ttype(term[90],col)
else
  ttype(term[90],row);

(* now send out ending string if any *)
for i := 15 to ( term[14] + 14 ) do
  write( chr( term[ i ] ) );
end;

***************************************************************************
 ***
 *** procedure:      clear
 *** version:         2.0
 *** date:            21 oct 83
 *** description:     this procedure clears the screen and homes the
 ***                   cursor. if status line is visible (stat_on =
 ***                   true ) then the status line is displayed.
 ***
 *** global variables used:       term,
 ***                                 stat_on,
 ***                                 stat_line
 ***
 *** global constants used:        term_length,
 ***                                 stat_line_width
 ***
 *** procedures called:            gotoxy
 *** called by:                     many

FILE:    terminal.pas
FILE: terminal.pas

* author: vincent m. parisi ii, capt., usaf
*
***************************************************************************

procedure clear;

var i : integer;

begin
  for i := 1 to term[ 19 ] do
    write( chr( term[ 19 + i ] ) );

  if stat_on then
    begin
      gotoxy( 22,0 );
      write( status_line );
      gotoxy( 0, 0 );
    end;
end;

***************************************************************************

(*procedure: VideoLow
version: 1.0
date: 27 Sep 84
description: this procedure puts the screen into low video

*global variables used: term,
*global constants used: term_length,
*procedures called: none
*called by: many
*author: Paul A. Moore, capt., usaf
*
FILE: terminal.pas*)
procedure VideoLow;
var i : integer;

begin
  i := 71 to ( term[ 70 ] + 70 ) do
    write( chr( term[ i ] ) );
end;

procedure SVideolo nied
version: S1.0
date: 27 Sep 84
description: this procedure inserts the character string to put
the screen into low video into the input string at the given
position and returns the modified string
global variables used: term,
global constants used: term_length,
procedures called: none
called by: many
author: Paul A. Moore, capt., usaf

{SV-} { Turn off checking of string lengths }
procedure SVideolo nied(var Instring : msg_line; pos : integer );

var i : integer;
tempstr : string[10];
FILE: terminal.pas

begin
  tempstr := ''; (* null string *)
  for i := 71 to (term[70] + 70) do
    tempstr := concat(tempstr, chr(term[i]));
  Insert(tempstr, Instring, pos);
end;

(* *******************************************************
 * Procedure: VideoBold
 * Version: 1.0
 * Date: 27 Sep 84
 * Description: this procedure writes the character string to put
 * the screen into bold video
 * Global variables used: term,
 * Global constants used: term_length,
 * Procedures called: none
 * Called by: many
 * Author: Paul A. Moore, capt., usaf
 *******************************************************)

{$V+}  { Turn on checking of string lengths  }

procedure VideoBold;

var i : integer;

begin
  for i := 77 to (term[76] + 76) do
    write( chr(term[i]) );
end;

FILE: terminal.pas
FILE: terminal.pas

*  * procedure: SVVideoBold  *
  * version: 1.0  *
  * date: 27 Sep 84  *
  * description: this procedure inserts the character string to put  *
  * the screen into bold video into the input string at the given  *
  * position and returns the modified string  *
  * global variables used: term,  *
  * global constants used: term_length,  *
  * procedures called: none  *
  * called by: many  *
  * author: Paul A. Moore, capt., usaf  *

{SV-}  { Turn off checking of string lengths  }
procedure SVVideoBold(var Instring : msg_line; pos : integer );

var i : integer;
  tempstr : string[10];

begin
  tempstr := '';  (* null string *)
  for i := 77 to ( term[ 76 ] + 76 ) do
    tempstr := concat(tempstr, chr(term[i]) );
  Insert(tempstr,Instring,pos);
end;
{SV+}  { Turn on checking of string lengths }

(FILE: terminal.pas)
FILE: terminal.pas

* date: 27 Sep 84
* description: this procedure draws a rectangle of the given dimensions
  on the video screen.
* global variables used: term
* global constants used:
* procedures called: graphics, gotoxy, nographics
* called by: many
* author: Paul A. Moore, capt., usaf

procedure Rectangle(line,column,width,height : integer);
var
  i, Ll, Cl : integer;
begin
  graphics;
  Ll := line + height - 1;
  Cl := column + width - 1;
  gotoxy(line,column);
  write(chr(term[64]));
  write(chr(term[64]));
  for i := column+1 to Cl-1 do
    write(chr(term[55]));
  write(chr(term[61]));
  (* upper left corner of Rectangle *)
  (* upper left corner *)
  (* top of Rectangle *)
  (* upper right corner *)
  (* columns of Rectangle *)
  for i := line+1 to Ll-1 do
    begin
      gotoxy(i,column); write(chr(term[54]));
      gotoxy(i,Cl); write(chr(term[54]));
    end;

FILE: terminal.pas
FILE: terminal.pas

```pascal
  gotoxy(L1,column);  (* lower left corner of Rectangle *)
  write(chr(term[63]));  (* lower left corner *)
  for i := column+1 to C1-1 do  (* bottom of Rectangle *)
    write(chr(term[55]));
  write(chr(term[62]));  (* lower right corner *)

  nographics;

  end;
```

FILE: terminal.pas
procedure trim( var cmdword : cmdword );

var i : integer;

FILE: trim.pas

Page: 1
begin
   (* trim trailing blanks from command word strings, I hope this *)
   (* is what it is supposed to do!!! *)
   i := length(cmdword);
   while (cmdword[i] = ' ') do
     i := i - 1;
     cmdword := copy(cmdword, 1, i);
   end;
procedure ucase( var instring : msg_line);
FILE: ucase.pas

var i : integer;

begin (* main body of procedure ucase *)
  for i := 1 to length(instring) do
    instring[i] := upCase(instring[i]);
end; (* end of procedure trim *)
FILE: valndec.pas

(* ****************************************************** *)
(* file: valndec.pas                              *)
(* version: 1.7                                    *)
(* date: 29 October 1984                           *)
(* description: this module contains the procedures that *)
(* validate and decode the user input              *)
(* command line.                                    *)
(* global variables used: none                     *)
(* global constants used: wordsize, buffersize,    *)
(* end_code, abbrev_code                           *)
(* modules called: val_n_dec, getline              *)
(* called by: get_com                              *)
(* author: vincent m. parisi ii, capt., usaf       *)
(* ****************************************************** *)

(* ****************************************************** *)
(* function: check_word                            *)
(* version: 1.1                                     *)
(* date: 29 October 1984                            *)
(* description:                                     *)
(* global variables used: none                      *)
(* global constants used:                           *)
(* modules called: none                             *)
(* called by: val_n_dec                             *)
(* author: Paul A. Moore, capt., usaf              *)
(* ****************************************************** *)
function check_word(decode : dictionary; command : cmdword) : boolean;

(* this function returns "true" if there is a match between the 
dictionary word and the command word. The function takes into 
account abbreviations of command words.*)

var
  dword : cmdword;
  d_len : integer;
  cmd_len : integer;
  i : integer;

begin
  dword := decode.dictword;  (* get rid of trailing blanks *)
  trim(dword);
  trim(command);

  check_word := false;  (* default to no match *)
  if command = dword then
    check_word := true
  else
    begin
      d_len := length(dword);
      cmd_len := length(command);

      (* make sure the command isn't too long or short *)
      if (cmd_len >= decode.abbrev) and (cmd_len <= d_len) then
        begin  (* compare characters *)
          i := 1;
          while (i <= cmd_len) and
            ( UpCase(command[i]) = UpCase(dword[i]) ) do
            i := i + 1;
          if ( i = cmd_len+1 ) then check_word := true;
FILE: valndec.pas

end;
end;
end; (* end check_word *)

(*----------------------------------------------------------*
 * procedure: val_n_dec
 * version: 1.6
 * date: 16 August 1983
 * description: this procedure validates and decodes
 * the command line input by the user. the
 * process begins by recovering record 1
 * from the syntax table, using get_line. *
 * a comparison is made with the first
 * word in the command buffer. *
 * If there is no *
 * match, the next record pointed to by nomatchp *
 * is retrieved with get_line. *
 * the comparisons continue until a match *
 * is found and the routine goes to the *
 * next level. if at any level no match *
 * is found after exhausting all the *
 * possibilities for that level then the *
 * error code is set to 'b' and the rou-
 * tine is exited. if a valid command *
 * is decoded but there are still some *
 * words present in the command buffer, the *
 * error code is set to 'c'. the error *
 * code is set to 'n' for a valid command. *
 * global variables used: none *
 * global constants used: wordsize, *
 * buffersize, *

FILE: valndec.pas
FILE: valndec.pas

* modules called: abbrev_code  
* check_word, getline
* called by: get_com
* author: vincent m. parisi ii, capt., usaf
*
***************

procedure val_n_dec( var level : integer; var rec_num : integer;
  var error_code : char; num_of_commands : integer;
  var cmdbuffer : buffer; var call Routine : cmdword );

var last_rec_num : integer;
cmd : cmdword;

begin
  (* "last_rec_num" points to the beginning of a list of options *)
  last_rec_num := rec_num;  (* save rec num*)

  (* If the word does not match, get the next
record which is pointed to by nomatchp. repeat this until we
run out of words indicated by an "ENDCODE" in nomatchp or there
is a match. *)

  while ((error_code = 'a') and (level <= (num_of_commands + 1))) do
    begin
      (* get the syntax line for entry rec_num *)
      get_line( decode, rec_num );
      cmd := decode.dictword;
    end

FILE: valndec.pas
trim(cmd);
if cmd = DONEWORD then
begin
  if level = (num_of_commands + 1) then
    begin
      error_code := 'n';
      call_routine := decode_dict.words[decode.matchp];
    end
  else
    begin
      error_code := 'c';
      rec_num := last_rec_num;
    end;
  end
else
  if cmdbuffer[level] = ' ' then
    error_code := 'd'
else
  if check_word(decode,cmdbuffer[level]) then
    begin
      cmdbuffer[level] := decode.dictword; (* replace possible abbr.*]
      level := level + 1;
      rec_num := decode.matchp;
      last_rec_num := rec_num;
    end
  else
    if decode.nomatchp = ENDCODE then
      begin
        error_code := 'b';
        rec_num := last_rec_num;
      end
    else
      (* try next option *)

FILE: valndec.pas
FILE: valdec.pas

rec_num := decode.nomatch;
(* end while *)
end;
end;
Bibliography


EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

UNCLASSIFIED  P R MOORE DEC 84 AFIT/GCS/ENG/84D-18  F/G 9/2


BIB - 3


VITA

Captain Paul A. Moore was born 13 May 1957 in Eugene, Oregon. He graduated from high school in Florence, Oregon, in 1975. He attended Oregon State University on an Air Force Reserve Officers Training Corps (AFROTC) Four-Year Scholarship. He served as Cadet Corps Commander during his senior year. He graduated from Oregon State University in June 1979, receiving the degree of Bachelor of Science in Computer Science. He was also a Distinguished Graduate of AFROTC. Upon graduation, he received a commission in the United States Air Force through the ROTC program. He then served as a software engineer for the 1000th Satellite Operations Group, Offutt AFB, Nebraska, until entering the School of Engineering, Air Force Institute of Technology, in June 1983.

Permanent Address: 555 E 50th Ave
Eugene, Oregon 97405
REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; Distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
APIT/GCS/ENG/84D-18

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL (if applicable)
APIT/EN

6c. ADDRESS (City, State and ZIP Code)
Wright-Patterson AFB, Ohio 45433

7a. NAME OF MONITORING ORGANIZATION

7b. ADDRESS (City, State and ZIP Code)

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (if applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

10. SOURCE OF FUNDING NOS.

11. TITLE (Include Security Classification)
See Box 19

12. PERSONAL AUTHOR(S)
Paul A. Moore, B.S., Capt, USAF

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM TO 1984 December

14. DATE OF REPORT (Yr., No. Day)

15. PAGE COUNT
490

16. SUPPLEMENTARY NOTATION

17a. COSSATI CODES

17b. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Software Development, Software Engineering, Software Development Environment, Compiler, Programming Environment

18. ABSTRACT (Continue on reverse if necessary and identify by block number)
Title: EXTENSION OF THE SOFTWARE DEVELOPMENT WORKBENCH TO INCLUDE MICROCOMPUTER WORKSTATIONS

Thesis Chairman: Dr. Gary B. Lamont


19. ABSTRACT SECURITIES CLASSIFICATION
UNCLASSIFIED

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED SAME AS RPT. DTC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Gary B. Lamont

22b. TELEPHONE NUMBER
513-255-3450

22c. OFFICE SYMBOL
APIT/ENG
19. Abstract:

The purpose of this investigation is to 1) design an integrated and automated software development environment for microcomputers, 2) to implement a portion of that environment and, 3) to modify the VAX-11/780 VMS Software Development Workbench to support microcomputer workstations.

The central portion of the initial microcomputer software development environment is a flexible user interface adapted from a previous microcomputer software development effort. The user interface is enhanced by an extensive installation program which allows 1) terminal definition, 2) help message specification and, 3) reconfiguration of the keyword menu system.

The result of the investigation is a prototype software development environment hosted on DEC Rainbow 100 microcomputer under three operating systems: CP/M-80, CP/M-86, and MS-DOS. The environment includes a flexible user interface, an installation program, a communications program, and an initial library of reusable software.

Supplied keywords include: "evaluation."