43113

p 33

User Documentation for the MSK and OMS Intelligent Tutoring Systems

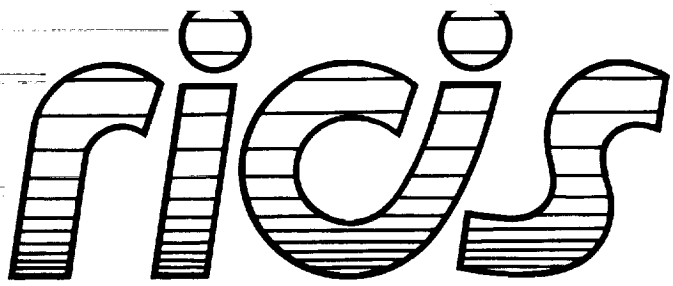
Pamela K. Fink
L. Tandy Herren
David T. Lincoln

Southwest Research Institute

May 2, 1991

Cooperative Agreement NCC 9-16
Research Activity No. ET.23

NASA Johnson Space Center
Mission Operations Directorate
Space Station Training Office



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

User Documentation for the MSK and OMS Intelligent Tutoring Systems

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Pamela K. Fink, Dr. L. Tandy Herren and David T. Lincoln of the Southwest Research Institute. Dr. Glenn Freedman, Director of the Software Engineering Professional Education Center at the University of Houston-Clear Lake, served as RICIS research coordinator.

Funding has been provided by the Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Barbara N. Pearson of the Systems/Elements Office, Space Station Training Office, Mission Operations Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

Contract No. NASA NCC-9-
Subcontract No. 060
Research Activity No. ET.
SwRI Project No. 05-3515

USER DOCUMENTATION FOR THE MSK AND OMS INTELLIGENT TUTORING SYSTEMS

Prepared By:

Pamela K. Fink, Ph.D.
L. Tandy Herren, Ph.D.
David T. Lincoln

Southwest Research Institute
San Antonio, Texas

May 2, 1991

TABLE OF CONTENTS

LIST OF FIGURES

User Documentation for the OMS and MSK Intelligent Tutoring System

1. THE USER'S GUIDE

This user's guide describes how to use the Intelligent Tutoring Systems for the Manual Select Keyboard (MSK) and the Orbital Maneuvering System (OMS) and how to use the C code that runs the Mockup version of the MSK.

2. THE MSK ITS CODE

2.1 Overview

This section describes the following aspects of the MSK Intelligent Tutoring System:

a. The directory structure and files maintained in each directory

b. How to use the MSK ITS in interpreted and executable modes

c. How to rebuild the interpreted and runtime versions of the MSK ITS using the makefiles provided. The system is compiled with the Apollo Domain X C compiler.

2.2 Directory Structure

The code for the MSK ITS exists in three sub-directories of the directory "/hrlits/mskits": graphics, tutor, and runtime. To see a listing of these directories, type "cd /hrlits/mskits" at the % prompt followed by the command "ls."

2.2.1 The Graphics Directory

The graphics directory contains the C source code and header files used to implement the graphic display of the MSK. The graphics directory also contains the object code generated by compiling the C source code. Figure 1 contains a listing of the files in the graphics directory. These files create the representation of the MSK on the computer display, blowup objects on user request, monitor student input during an exercise, collect the speed of performance, generate a quiz during the overview of the tutor, and perform many other basic utilities of the ITS.

2.2.2 The Tutor Directory

The tutor directory contains the CLIPS code that implements the tutor's instructional strategy, expert model, and student model and directs the flow of control in the ITS. Figure 2 contains a listing of the files in the tutor directory. The following files constitute the bulk of the CLIPS rules:

a) buildex.clp - these rules construct the exercises

b) cleanup.clp - these rules remove facts from the previous exercise and update information about the student prior to a new exercise

c) evaluate.clp - these rules evaluate the student's performance on an exercise and recommend a course of action

d) exercise.clp - these rules perform a search of the exercise tree to determine what type of exercises the student should perform

e) genextype.clp - these rules randomly generate exercise types based on the student's needs during unguided, speeded, and automated phases of training

```
beep.c
beep.o
beep_text.c
blowup_obj.c
blowup_obj.o
blowup_panel.c
blowup_panel.o
ckfont
cktext
clr_text.c
colors.c
colors.o
console
console.c
console.o
console.t
create_plot.c
ddd.h
ddd.t
defs.h
demo_step.c
demo_step.o
display_plot.c
display_plot.o
display_text.c
display_text.o
draw_cons.c
draw_cons.o
draw_goal_band.c
draw_goal_band.o
draw_goal_box.c
draw_leds.c
draw_leds.o
draw_obj.c
draw_obj.o
draw_panel.c
draw_panel.o
draw_plot.c
draw_plot.o
drk.h
drk.t
exer.clp
get_status.c
get_status.o
get_student_proc.c
get_student_proc.o
get_student_step.c
get_student_step.o
globals.c
globals.o
hilight.c
hilight.o
init_struc.c
init_struc.o
int_to_asc.c
int_to_asc.o
main.c
main.o
makefile
makefile.t
mbutn_update.c
mbutn_update.o
monitor.h
msk.h
msk.t
num.txt
objects.h
panels.h
plot.c
plot.h
pop_buttons.c
pop_buttons.o
position.c
position.o
quiz1.c
quiz1.o
quiz1text.c
quiz1text.o
quiz2.c
quiz2.o
quiz2text.c
quiz2text.o
quizutil.c
quizutil.o
random.c
random.o
randomtot.c
randomtot.o
select.c
select.o
smek.h
smek.t
stdalone.c
stdalone.o
stopclk.h
stopclk.t
tan2file
test.clp
testdraw.c
testplot
testtext.c
testtext.o
textstring.h
toggle.c
toggle.o
train1.c
train1.o
train1text.c
train1text.o
train2.c
train2.o
train2text.c
train2text.o
trends.c
trends.c.bak
trends.o
trends.tmp
tutor.h
tutor.t
tutorutil.c
tutorutil.o
txtlin.txt
unxclips.mak
util.c
util.o
voicekey.h
voicekey.t
windows.h
write_IO.c
write_IO.o
```

Figure 1. Contents of the MSKITS/Graphics Directory

```
assign.clp          exer.stack          load.file           speedwork.ovl
autowork.ovl        exercise.clp        parmsset            speedwork.perf
autowork.perf       facts.clp           rav                 student.model
buildex.clp         funcalls.clp        remediate.clp       tandy.ovl
cleanup.clp         genextype.clp       resetcrit           tandy.perf
cleanup.tmp         goals.clp           setparms            tanfile
console             graph.clp           skills.clp          toplevel.clp
evaluate.clp        instruct.parms      speed.ovl           train.clp
evaluate2.clp       instructparms.c     speed.perf          update.clp
```

Figure 2. Contents of the MSKITS/Tutor Directory

f) remediate.clp - these rules control remediation by backtracking through the exercise tree

g) toplevel.clp - these rules initiate startup of the MSK tutoring system

2.2.3 The Runtime Directory

The runtime directory contains the object code created by compiling the C and CLIPS files. It also contains the executable version of the MSK ITS, called MSKITS (see Figure 3). In order to run the executable version, the file "instruct.parms" must be present in the same directory and the user simply types "mskits." Instruct.parms is created by running the PARMSSET executable (see Section 2.4).

The runtime directory also contains the C source code files located in the graphics directory as well as the C source code files generated by the "rules-to-C" function (see Section 2.5).

2.3. Using the MSK ITS

2.3.1 Using the Executable Version of the MSK ITS

To use the executable version of the MSK tutor, change directories to the /hrlits/mskits/runtime directory. At the % prompt, type "mskits." Be sure that the file "instruct.parms" is in the same directory. The executable version of the tutor can be run in any directory as long as instruct.parms is in the same directory.

2.3.2 Using the Interpreted Version of the MSK ITS

The interpreted version of the MSK ITS must be run from the /hrlits/mskits/tutor directory. At the % prompt, type "console." You will then get a CLIPS prompt and type as follows:

```
clips> (load "load.file")
clips> (assert (load))
clips> (run)
```

After the rules are loaded, type:

```
clips> (reset)
clips> (run)
```

This will load the MSK ITS into memory. Once you have exited the tutor, exit CLIPS by typing "(exit)" at the CLIPS prompt. This will return you to the Domain X system prompt.

2.3.3 The Output Files Created by the MSK ITS

The MSK ITS creates two output files for each student who uses the tutor. One file contains the student model and is identified by the student's identifier and a ".ovl" extension (e.g., george.ovl). The student model consists of the entire training tree (see Figure 4) and indicators marking which of the nodes the student has completed. The student model also contains the speed-criterion and automate-criterion lists. These lists are used during speeded and automated

```
a.out
beep.c
beep.o
beep_text.c
blowup_obj.c
blowup_obj.o
blowup_panel.c
blowup_panel.o
clips.h
clr_text.c
colors.c
colors.o
console.c
console.o
create_plot.c
ddd.h
deffacts.h
defs.h
demo_step.c
demo_step.o
display_plot.c
display_plot.o
display_text.c
display_text.o
draw_cons.c
draw_cons.o
draw_goal_band.c
draw_goal_band.o
draw_goal_box.c
draw_leds.c
draw_leds.o
draw_obj.c
draw_obj.o
draw_panel.c
draw_panel.o
draw_plot.c
draw_plot.o
drk.h
get_status.c
get_status.o
get_student_proc.c
get_student_proc.o
get_student_step.c
get_student_step.o
globals.c
globals.o
hilight.c
hilight.o
init_struc.c
init_struc.o
instruct.parms
int_to_asc.c
int_to_asc.o
main.c
main.o
makefile
makefile.bak
mbutn_update.c
mbutn_update.o
monitor.h
msk.h
mskits
msktutor0.c
msktutor0.o
msktutor1.c
msktutor1.o
msktutor2.c
msktutor2.o
msktutor3.c
msktutor3.o
msktutor4.c
msktutor4.o
msktutor5.c
msktutor5.o
msktutor6.c
msktutor6.o
msktutor7.c
msktutor7.o
objects.h
panels.h
parmsset
plot.c
plot.h
pop_buttons.c
pop_buttons.o
position.c
position.o
quiz1.c
quiz1.o
quiz1text.c
quiz1text.o
quiz2.c
quiz2.o
quiz2text.c
quiz2text.o
quizutil.c
quizutil.o
random.c
random.o
randomtot.c
select.c
select.o
setup.h
smek.h
stdalone.c
stdalone.o
stopclk.h
tan3file
tandy.ovl
tandy.perf
testdraw.c
testtext.c
testtext.o
textstring.h
toggle.c
toggle.o
train1.c
train1.o
train1text.c
train1text.o
train2.c
train2.o
train2text.c
train2text.o
trends.c
trends.o
tutor.h
tutorutil.c
tutorutil.o
util.c
util.o
voicekey.h
windows.h
write_IO.c
write_IO.o
```

Figure 3. Contents of the MSKITS/Runtime Directory

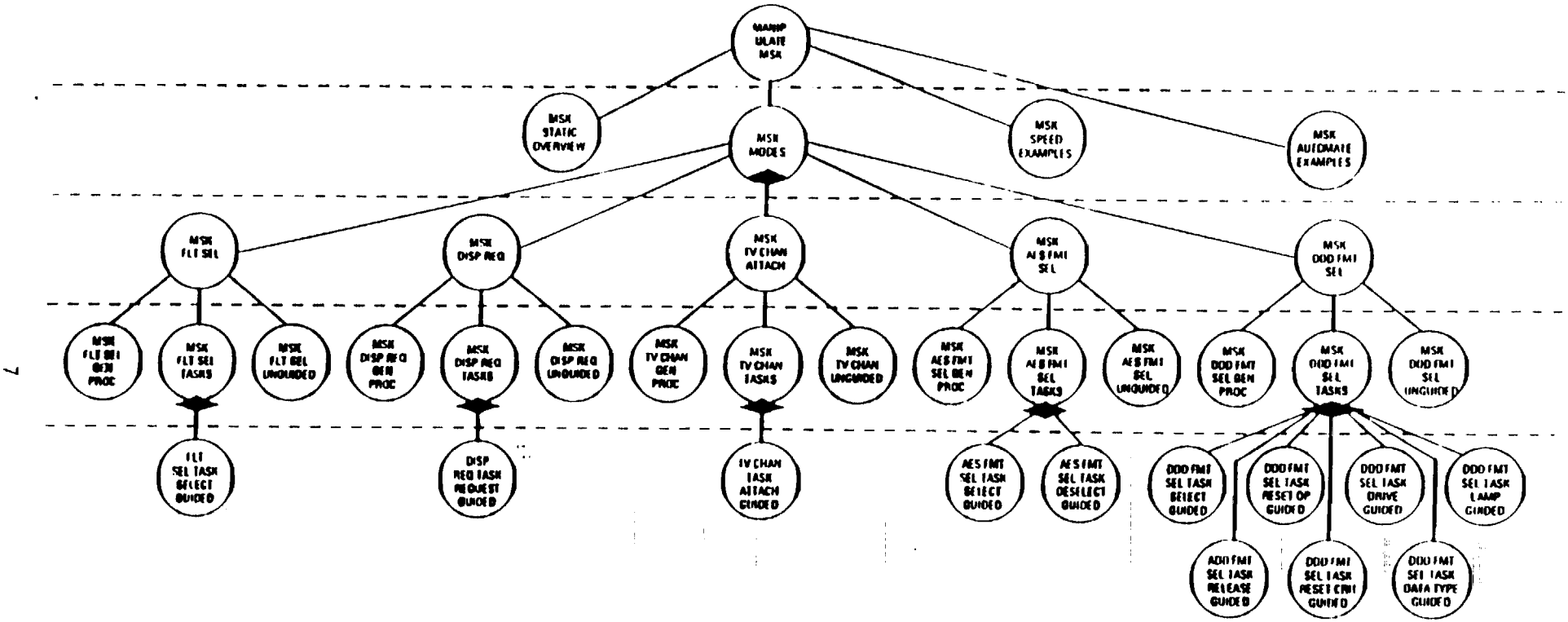


Figure 4. The MSK Training Tree

exercises to determine on which exercises the student needs to perform at the criterion set by the instructor in the file "instruct.parms."

The MSK ITS also creates a file that maintains the student's reaction times and accuracy levels from the speeded and automated exercises. This file is named using the student's identification and a ".perf" extension (e.g., george.perf). Each line in this file corresponds to a single exercise. The line format is as follows:

a) training phase: 1 = unguided or guided; 2 = speeded; 3 = automated

b) operation type: 1 = tv channel attach; 2 = display request; 3 = DDD format; 4 = AES format; 5 = flight select

c) operation sub-type: 1 = tv channel attach, display request, flight select, AES select, or DDD select; 2 = AES deselect or DDD deselect; 3 = DDD reset operational limits; 4 = DDD reset critical limits, 5 = DDD select drive; 6 = DDD select datatype; 7 = DDD select lamp test

d) speed of performance: in seconds

e) accuracy of performance: percentage of steps until error

f) a list of 1's marking the number of accurate steps in the operation; each 1 signifies an accurate step

2.4. Setting MSK ITS Parameters

Various parameters of the MSK tutoring system can be set by the user. The source code that implements parameter collection is in the file "instructparms.c" located in the tutor directory. The executable version of this file is called parmsset. To provide new parameters to the system, type "parmsset" at the % prompt. After the user provides the parameter values based on system query, the executable will create the instruct.parms file. Instruct.parms must be in the tutor directory to run the MSK ITS in interpreted mode and in the same directory as the MSKITS executable.

The parameters collected by parmsset include the following:

a) Maximum Speed. This is the speed at which a student must be performing consistently to avoid remediation.

b) Speed Criterion. The student must accurately perform two of each operation at or better than this speed in order to proceed to the automated exercises.

c) Automate Criterion. The student must accurately perform two of each operation at or better than this speed and be accurate in responding to the secondary task to end training.

d) Number of Beep Patterns. This number determines the size of the pool of possible beep patterns that will be selected from.

e) Number of Beeps in a Pattern. This number determines the length of the beep patterns.

f) Number of Target Patterns. This is the number of patterns that the student must recognize and respond to on any given exercise.

g) Percentage of Target Beep Patterns. This number determines the frequency that the system provides a target versus a distractor beep pattern to the student.

h) Latency between Beep Patterns. The amount of time that elapses between beep patterns during an exercise.

i) The system asks a series of questions to determine which of the five operations to train during the speeded and automated exercises.

2.5 Changing the Rules and Modifying the C Code for the MSK ITS

2.5.1 Changing the Rules for the MSK ITS

Changing the rules in the CLIPS files located in the tutor directory will change the operation of the interpreted version of the MSK ITS but not the executable version. To modify a rule, change directories to the tutor directory and modify the corresponding file. Then run the tutor in the interpreted mode.

If you wish to recompile the executable version of the MSK ITS following a change to the rules, perform the following steps:

```
% cd /hrlits/mskits/tutor
% console
clips> (load "load.file")
clips> (assert (load))
clips> (run)
```

After the rules are loaded, type:

```
clips> (rules-to-c "msktutor" 1)
clips> (exit)
```

This procedure produces eight source files named msktutor0.c through msktutor7.c. Copy these files to the /hrlits/mskits/runtime directory and then type:

```
%copy msktutor*.c ../runtime
% cd ../runtime
% make mskits
```

The mskits makefile will create the mskits executable.

2.5.2 Modifying the C Source Code for the MSK ITS

Any changes to a C source code file requires recompilation of that file and relinking the files to create "console." If you want to change only the interpreted version of the MSK ITS, change directories to the /hrlits/mskits/graphics directory. Type "make" at the % prompt. This will recompile the file that was changed and relink the files. After this, copy "console" to the /hrlits/mskits/tutor directory, change directories, and run the interpreted version.

If you want to modify the executable version of the MSK ITS, copy the relinked version of "console" to the /hrlits/mskits/runtime directory and change directories. At the system prompt type "make mskits." This will produce a new executable version of the MSK ITS.

3. The OMS ITS CODE

3.1 Overview

This section describes the following aspects of the OMS Intelligent Tutoring System:

a. The directory structure and files maintained in each directory

b. How to use the OMS ITS in interpreted and executable modes

c. How to rebuild the interpreted and runtime versions of the OMS ITS using the makefiles provided. The system is compiled with the Apollo Domain X C compiler.

3.2 Directory Structure

The code for the OMS ITS exists in three sub-directories of the directory "/hrlits/omsits": graphics, tutor, and runtime. To see a listing of these directories, type "cd /hrlits/omsits" at the % prompt followed by the command "ls."

3.2.1 The Graphics Directory

The graphics directory contains the C source code and header files used to implement the graphic display of the components of the shuttle orbital maneuvering system (OMS) and the VDT screens that are displayed on the propulsion console. This code also implements the static overview portion of the system. The graphics directory also contains the object code generated by compiling the C source code. Figure 5 contains a listing of the files in the graphics directory. These files create the representation of the shuttle OMS and VDT screens on the computer monitor, blowup objects on user request, monitor student input during an exercise, collect the speed of performance, generate a quiz during the overview of the tutor, and perform many other basic utilities of the OMS ITS.

3.2.2 The Tutor Directory

The tutor directory contains the CLIPS code that implements the tutor's instructional strategy, expert model, and student model and directs the flow of control in the OMS ITS. Figure 6 contains a listing of the files in the tutor directory. The following files constitute the bulk of the CLIPS rules:

a) DDD_overview.clp - these rules control the overview of the DDD panels on the propulsion console

b) Diag_evaluate.clp - these rules control remediation to MSK operations

c) diag_exercise.clp - these rules perform a search of the exercise tree to determine what type of exercises the student should perform

d) Diag_facts.clp - this file contains the facts asserted into working memory at start-up

```
DDD_overview.c
DDD_overview_quiz.c
DDD_overview_text.c
OMS_overview.c
OMS_overview_text.c
OMS_utils.c
beep.c
beep_text.c
blowup_obj.c
blowup_panel.c
build_press.c
clr_text.c
coast_mode.c
colors.c
console.c
console_reset.c
create_plot.c
demo_step.c
display1105.c
display_msk.c
display_plot.c
display_screens.c
display_temps.c
display_text.c
displayutil.c
downstream_leak.c
draw_cons.c
draw_goal_band.c
draw_goal_box.c
draw_leds.c
draw_obj.c
draw_panel.c
draw_plot.c
get_diag_answer.c
get_diagnostic_step.c
OMS.h
plot.h
failure.h
voicekey.h
objects.h
displays.h
textstring.h
get_leak.c
get_sensor.c
get_status.c
get_student_proc.c
get_student_step.c
get_which.c
globals.c
helium_system_failure.c
hilight.c
init_ddd.c
init_struc.c
int_to_asc.c
leaks.c
limits_disabled.c
main.c
mbutn_update.c
overview_quiz.c
plot.c
pop_buttons.c
popup.c
position.c
present_diagnostics.c
present_msk_tasks.c
present_vdt.c
press_sensor.c
print_copy.c
quiz1.c
quiz1text.c
quiz2.c
quiz2text.c
quizutil.c
random.c
randomtot.c
reset_crit.c
run_simulation.c
drk.h
tutor.h
msk.h
defs.h
stopclk.h
limits.h
select.c
sensor_failures.c
start_diag_sim.c
stdalone.c
system_func_text.c
tank_leak.c
temp_sensor.c
test_diag_general.c
test_helium.c
test_msk_tasks.c
testdraw.c
testtext.c
toggle.c
train1.c
train1text.c
train2.c
train2text.c
trends.c
tutorutil.c
update_1101_pt1.c
update_1103.c
update_1105.c
update_1110.c
util.c
valve_failure.c
valve_sensor.c
write_1101_pt1.c
write_1101_valve.c
write_1103.c
write_1105.c
write_1109.c
write_1110.c
write_IO.c
makefile
makerun
monitor.h
ddd.h
smek.h
helium.h
windows.h
panels.h
```

Figure 5. Contents of the OMSITS/Graphics Directory

```
diag_eval.clp
diag_exercise.clp
diag_facts.clp
diag_msk.clp
diag_update.clp
facts.clp
coast_mode.ovl2
console_prac.ovl2
ddd_display.ovl2
diag_gen.ovl2
diag_unguided.ovl2
oms_assign.clp
oms_buildex.clp
oms_cleanup.clp
oms_evaluate.clp
oms_exercise.clp
oms_funcalls.clp
gnd_prac.ovl2
gnd_unguided.ovl2
limits_dis.ovl2
no_problem.ovl2
system_failures.ovl2
oms_genextype.clp
oms_goals.clp
oms_graph.clp
oms_overview.clp
oms_remediate.clp
oms_skills.clp
valve_fail.ovl2
valve_pos.ovl2
vdt_display.ovl2
vdt_unguided.ovl2
load.file
oms_toplevel.clp
oms_train.clp
oms_update.clp
templates.clp
```

Figure 6. Contents of the OMS ITS/Tutor Directory

e) Diag_msk.clp - these rules evaluate MSK operations

f) Diag_update.clp - these rules control remediation during diagnostic training

g) OMS_buildex.clp - these rules construct the exercises

h) OMS_cleanup.clp - these rules remove facts from the previous exercise and update information about the student prior to a new exercise

i) OMS_genextype.clp - these rules randomly generate exercise types during the diagnostic phases of training based on student needs

j) OMS_overview.clp - these rules control the overview of the OMS pod components

k) OMS_remediate.clp - these rules control remediation by backtracking through the exercise tree

l) OMS_toplevel.clp - these rules initiate the startup of the OMS tutoring system

m) templates.clp - this file contains the templates (data structures) loaded at start-up

3.2.3 The Runtime Directory

The runtime directory contains the object code created by compiling the C and CLIPS files. It also contains the executable version of the OMS ITS, called OMSITS (See Figure 7).

The runtime directory also contains the C source code files located in the graphics directory as well as the C source code files generated by the "rules-to-c" function.

3.3. Using the OMS ITS

3.3.1 Using the Executable Version of the OMS ITS

To use the executable version of the OMS tutor, change directories to the /hrlits/omsits/runtime directory. At the % prompt, type "omsits."

3.3.2 Using the Interpreted Version of the OMS ITS

The interpreted version of the OMS ITS must be run from the /hrlits/omsits/tutor directory. Type:

```
% console
clips> (load "load.file")
clips> (assert (load))
clips> (run)
```

```
main.c          msktutor2.c   msktutor5.c
msktutor0.c     msktutor3.c   msktutor6.c
msktutor1.c     msktutor4.c   msktutor7.c
```

Figure 7. Contents of the OMSITS/Runtime Directory

After the rules are loaded, type:

```
clips> (reset)
clips> (run)
```

This will load the OMS ITS into memory. Once you have exited the tutor, exit CLIPS by typing "(exit)" at the clips> prompt. This will return you to the Domain X system prompt.

3.3.3 The Output Files Created by the OMS ITS

The OMS ITS creates two output files for each student who uses the tutor. One file contains the student model and is identified by the student's identifier and a ".ovl2" extension (e.g., george.ovl2). The student model consists of the entire training tree (see Figure 8) and indicators marking which of the nodes the student has completed. This training tree teaches the expert knowledge shown by Figure 9.

The OMS ITS also creates a file that maintains the student's reaction times and accuracy levels from the guided and unguided diagnostic exercises. This file is named using the student's identification and a ".perf" extension (e.g., george.perf). Each line in this file corresponds to a single exercise. The line format is as follows:

1) 1 signalling that the system only covers guided or unguided exercises

2) training level: 6 = VDT task; 7 = console reset; 8 = critical reset; 9 = diagnostic guided; 10 = diagnostic unguided

3) problem type: 1 = pressure sensor failure; 2 = temperature sensor failure; 3 = valve position sensor failure; 4 = limits disabled; 5 = dual pressure regulator failure; 6 = tank leak; 7 = downstream leak; 8 = no problem

4) failed position: 0 if irrelevant; 1 or 2 for pressure sensor; A or B for valve position sensor and dual pressure regulators

5) student's answer: same format as 3 and 4

6) diagnostic accuracy

7) diagnostic speed

8) this is followed by performance data on the MSK operations:
   - for each MSK operation, the type of operation, accuracy, and speed are recorded

The data for 3 through 7 are only recorded if the subject is in the diagnostic guided or unguided phases. The data for 5 is recorded on in the diagnostic unguided phase. MSK performance indices are recorded for all phases.

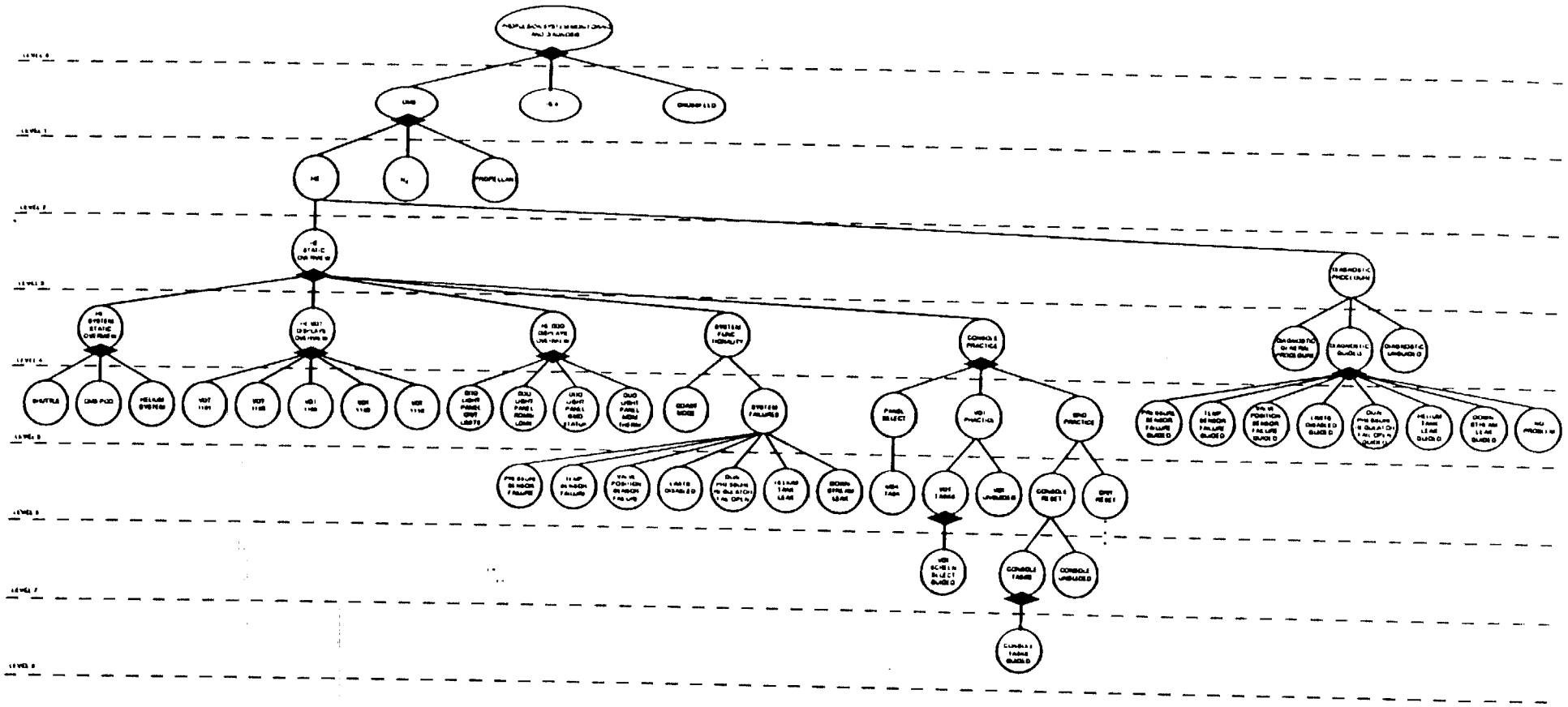


Figure 8. The OMS Training Tree

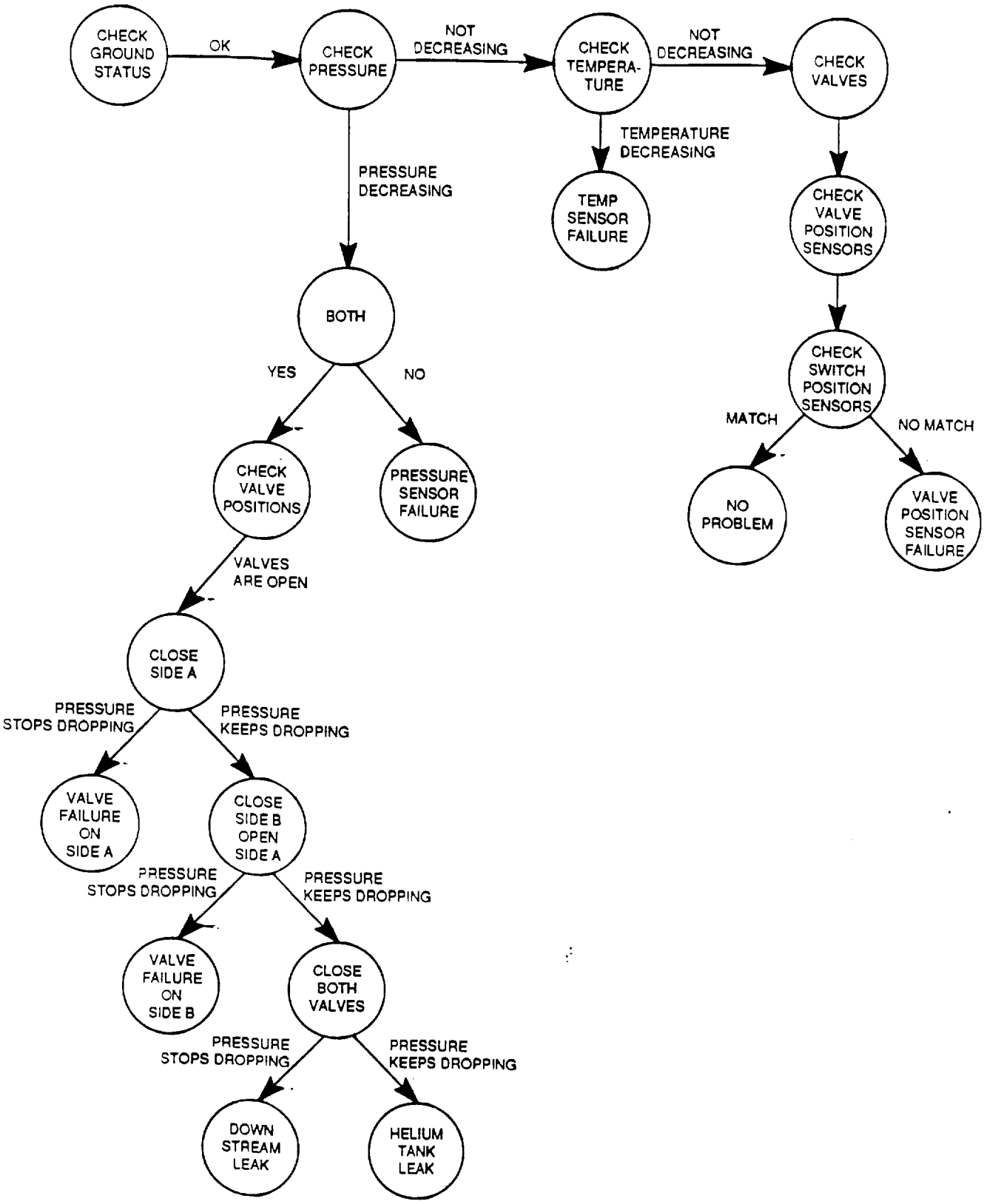


Figure 9. The OMS Expert Model

3.4 Changing the Rules and Modifying the C Code for the OMS ITS

3.4.1 Changing the Rules for the OMS ITS

Changing the rules in the CLIPS files located in the tutor directory will change the operation of the interpreted version of the OMS ITS but not the executable version. To modify a rule, change directories to the tutor directory and modify the corresponding file. Then run the tutor in the interpreted mode.

If you wish to recompile the executable version of the OMS ITS following a change to the rules, perform the following steps:

```
% cd /hrlits/omsits/tutor
% console
clips> (load "load.file")
clips> (assert (load))
clips> (run)
```

After the rules are loaded, type:

```
clips> (rules-to-c "omstutor" 1)
clips> (exit)
```

This procedure produces eight source files named omstutor0.c through omstutor7.c. Copy these files to the /hrlits/omsits/runtime directory and then type:

```
% copy omstutor*.* ../runtime
% cd ../runtime
% make omsits
```

The omsits makefile will create the omsits executable.

3.4.2 Modifying the C Source Code of the OMS ITS

Any changes to a C source code file requires recompilation of that file and relinking the files to create "console." If you want to change only the interpreted version of the OMS ITS, change directories to the /hrlits/omsits/graphics directory. Type "make" at % prompt. This will recompile the file that was changed and relink the files. After this, copy "console" to the /hrlits/omsits/tutor directory, change directories, and run the interpreted version.

If you want to modify the executable version of the OMS ITS, copy the relinked version of "console" to the /hrlits/omsits/runtime directory and change directories. At the system prompt type "make omsits." This will produce a new executable version of the OMS ITS.

4. THE MOCKUP CODE

4.1 Overview

This manual will describe the following aspects of the MSK TUTOR mockup for the PC:

1. Directory structure and the files associated with each directory
2. How to use the TUTOR in an interpreted mode and as a runtime executable
3. How to rebuild the interpreted and runtime versions via makefiles (The current version of the tutor is compiled with Microsoft C Ver. 5.10)

4.2 Directory Structure

The directory structure for the mockup and a brief description of the pertinent files is given below (See Figure 10).

4.2.1 \PROJECT\MSK\BIN

The directory \project\msk\bin contains copies of working executables and batch files described below:

console.exe - The interactive version of the used to test new rules. This is quicker than recompiling MSK TUTOR every time.

dio48.exe - This file contains routines for configuring and controlling the ICS DIO48 digital I/O card. Refer to the ICS DIO48 Reference Manual for more information.

makeall.bat - A batch file to remake all the make files in the \project\msk\src subdirectories. MSKTUTOR.EXE is the executable created.

msk.dat - This file contains the configurable parameters for the msk software.

msk.exe - Contains functions that:

1) Call the msk software to monitor panel inputs and track the elapsed time for the procedure.

2) Assert the string of student actions and elapsed time into CLIPS.

3) Drive the switch monitoring software. Monitor panel switches and compile a list of switches as they are modified.

4) Compare current input states to the previous input states to determine if a different group of switches has been modified than the previous group modified. If so, returns the group number of the current group modified. Otherwise, returns NULL.

5) Add the input state of the group modified to the list of switches which have been modified.

6) Calculate the elapsed time for the procedure.

```
Directory of C:\PROJECT\MSK\WORKING

DELETED     <DIR>
LOAD.FIL
RECORD.CLP
TOPLEVEL.CLP
ASSIGN.CLP
GENEXTYP.CLP
GOALS.CLP
MOCKUPFA.CLP
TRAIN.CLP
UPDATE.CLP
SKILLS.CLP
CLEANUP.CLP
EXERCISE.CLP
BUILDEX.CLP
EVALUATE.CLP

Directory of C:\PROJECT\MSK\SRC

CONFIG      <DIR>
DIO48       <DIR>
MAIN        <DIR>
UTIL        <DIR>
CONSOLE     <DIR>
README.TXT

Directory of C:\PROJECT\MSK\SRC\CONFIG

CONFIG
CNFGTEST.C
CONFIG.C
CNFGTEST.DAT
CNFGTEST.EXE

Directory of C:\PROJECT\MSK\SRC\DIO48

DIO48
DIO48.C
DIODRVR.C

Directory of C:\PROJECT\MSK

MSKTUTOR <DIR>
BIN <DIR>
DOC <DIR>
INCLUDE <DIR>
LIB <DIR>
WORKING <DIR>
SRC <DIR>
README.TXT

Directory of C:\PROJECT\MSK\MSKTUTOR

TUTOR0.C
TUTOR1.C
TUTOR2.C
TUTOR3.C
MSKTUTOR.C
TUTOR4.C
MSKTUTOR
TUTOR5.C
TUTOR6.C
TUTOR7.C
README.TXT
LINK.RSP
MSKTUTOR.RSP
MAIN.C
LOAD.FIL
SETUP.H

Directory of C:\PROJECT\MSK\BIN

DIO48.EXE
MAKEALL.BAT
MSK.DAT
MSKGOOD.EXE
README.TXT
CONSOLE.EXE
MSKTUTOR.EXE
MSK.EXE

Directory of C:\PROJECT\MSK\SRC\MAIN

LINK.RSP
MSK
INITPARM.C
MSKDRVR.C
MSK.C

Directory of C:\PROJECT\MSK\SRC\UTIL

UTIL.LIB
UTIL
BITOPS.C
SWITCHOP.C

Directory of C:\PROJECT\MSK\SRC\CONSOLE

DELETED     <DIR>
BEEP.C
DISPTEXT.C
DUMMIES.C
GETSTAT.C
GETPROC.C
GLOBALS.C
CONSOLE
QUIZUTIL.C
TRENDS.C
TUTRUTIL.C
MAIN.C
RANDOM.C
MODTRUNC.C
LIB.RSP
CONSOLE.C
LINK.RSP

Directory of C:\PROJECT\MSK\DOC

PARTLIST
BOARD.DRW
PANEL.DRW
SWITCHES.DRW
TITLE.WP
SWLABELS.WP
DIOCHANS
MCCONECT
SWCHANS
SWCONECT

Directory of C:\PROJECT\MSK\INCLUDE

DIO48.H
CONFIG.H
README.TXT
MSK.H
MOCKUP.H

Directory of C:\PROJECT\MSK\LIB

CONFIG.LIB
DIO48.LIB
UTIL.LIB
MSKTUTOR.LIB
CONSOLE.LIB
SD.LIB
MSK.LIB
README.TXT
```

Figure 10. The Directory Structure of the MOCKUP C Code

mskgood.exe - Test file msk driver.

msktutor.exe - The compiled version of the C code created by the rules together with the CLIPS code that runs without the CLIPS user interface.

4.2.2 \PROJECT\MSK\DOC

The directory \project\msk\doc contains the documentation on hardware configuration.

4.2.3 \PROJECT\MSK\INCLUDE

The directory \project\msk\include contains the header files for the MSK TUTOR C Code.

4.2.4 \PROJECT\MSK\LIB

The directory \project\msk\lib contains the object code libraries created by specific makefiles.

4.2.5 \PROJECT\MSK\MSKTUTOR

The directory \project\msk\msktutor contains:

a. main.c for runtime version only
b. setup.h with the runtime flag set
c. the makefile, called msktutor, which creates the MSK TUTOR executable
d. tutor0.c - tutor7.c files created in \project\msk\working

The difference between the main.c program in this directory and the main.c in the \project\msk\src\console directory is that this main.c does not include any user defined functions. It also has a different calling sequence to CLIPS. These differences are necessary when creating a runtime version of rules.

4.2.6 \PROJECT\MSK\SRC

The directory \project\msk\src contains the subdirectories with the C code for the tutor.

a. \config - contains C routines to configure parts of the control panel
b. \dio48 - contains the C functions needed to configure and control the ICS DIO48 digital I/O card.
c. \main - contains the C routines that builds msk.exe, which is described above.
d. \util - contains bitops.c which is composed of routines for bit field manipulation and switchop.c which contains routines for msk switch operations.
e. \console - contains C routines which make up the interpreted version of the tutor, called console.exe, described above. It is used to test new rules.
f. \working - directory where clips rules are tested and then compiled into C code

4.3 Using the Tutor

4.3.1 Running the Interpreted and Executable Versions

To run the tutor in an interpreted mode, go to the \project\msk\working directory and type:

```
C> console
clips> (load "load.fil")
clips> (assert (load))
clips> (run)
clips> (reset)
clips> (run)
```

This will result in all facts and rules being loaded into memory. Next, the system will prompt you for a student identification. You are then asked which of the tasks the student is to perform. You may choose any combination of the tasks. For each positive response, the student will be expected to perform four trials at 100 percent accuracy. If the student makes an error, that trial does not count as far as completion is concerned, but all trials are written to the student's record.

As soon as the last question about tasks is answered, the first exercise is displayed on the screen. Also at that time the student's output file is opened to record that student's progress. From the moment the exercise appears to the moment the student presses the return key is referred to as the "read time." Then, as each step is completed, the time for that step is recorded, and when the final return is pressed, the overall time is calculated. At this point, the tutor will display each step for the exercise and whether or not the student performed the step accurately. If all steps are correct, then the overall time for the exercise is displayed. Finally, the system will ask if the student wishes to continue.

To run the tutor as an executable, you do not need to be in any specific directory as long as the PC's path name includes the directory where the msktutor.exe resides. (Normally in \project\msk\bin). Then type:

```
C> msktutor
```

The tutor will run in the same manner described above.

4.3.2 Output Files Created by the Tutor

The tutor creates a single output file, named after the student. For example, if the student's name is Bill, then the output file is "bill." An example of a student's output is given below. In the first example, the student performed the task accurately, and in the second, an error was made. Observe in the second example, an arbitrary speed of fifty is assigned since speed is not the primary factor when the student is inaccurate.

Example 1:

```
msk_ddd_format_select
Read Instruction Time 1.64999998
student_step 1 0 function_code_thumbwheel 1 ok time 0
student_step 2 0 mode_select_pbi ddd fmt_sel ok time 4.82999992
student_step 3.0 lever_switches 4440 ok time 8.39999962
student_step 4 0 data_type_pbi sim_playback2 ok time 2.20000005
student_step 5 0 enter_switches left_mon_enter ok time 1.87
student_step 6 0 proc_done on ok time 0
accuracy_list 100
speed_list 17.35000038
end
```

Example 2:

```
msk_tv_channel_attach
Read Instruction Time 0.82999998
student_step 1 0 function_code_thumbwheel 1 ok time 0
student_step 2 0 mode_select_pbi tv_chan ok time 3.3499999
student_step 3 0 lever_switches 50 ok time 4.94000006
student_step 4 1 enter_switches right_mon_enter bad time 1.70000005
accuracy_list 57.1428566
speed_list 50
end
```

4.4 Changing the Rules and Modifying the C Code

Changing only the rules will not affect the interpreted version of the tutor and therefore is the quickest method for testing changes to the rules. Once you are satisfied that the changes work correctly, you then want to recompile the runtime version, msktutor.exe. Perform the following steps to do this:

```
C> cd \project\msk\working (this should be the directory where changes occur)
C> console
clips> (load "load.fil")
clips> (assert (load))
clips> (run)
clips> (rules-to-c "tutor" 1)
clips> (exit)
```

At this point, you should have eight source files named tutor0.c - tutor7.c. Next, copy these to \project\msk\msktutor and then type:

```
C> cd \project\msk\msktutor
C> make msktutor
```

The msktutor makefile will create the msktutor.exe and place it in the \project\msk\bin directory.

Any changes to the C code requires a recompilation of that particular file via the makefile in the directory where the C file resides. For example, if you change the msk.c file in \project\msk\src\main, you perform the changes and then type:

C> make msk (do not confuse the C source files and the make file names)

Unless you are in \project\msk\src\console directory already, type the following:

C> cd \project\msk\src\console

Then type:

C> make console

This rebuilds the entire console.exe. Then, as described above, you must rebuild msktutor.exe.