CDS -- C.A.I. Design System
System Documentation

James A. Schuyler, PhD
University of the Pacific
Division of Respiratory Therapy
2155 Webster Street
San Francisco, CA 94115

Creation of CDS was supported
as a part of the UoP Division
of Respiratory Therapy, funded
by Allied Health Professions
Special Project Grants 1 D12 AH00823-01
and 2 D12 AH00823-02, Public Health Service,
Department of Health, Education and Welfare.

This document was produced in
fulfillment of a contract with
the National Library of Medicine,
National Institutes of Health,
Department of Health, Education and
Welfare. (LO-468292-07)

Index

Section Title

1.0 Introduction

- 1.1 Rationale -- basic goals
  - 1.1.1 The Authoring Process
  - 1.1.2 Courseware Transfer
- 1.2 Design objectives
- 1.3 System overview

for learners

2.0 USING CDS Courseware

- 2.1 Signon
  - 2.1.1 Connecting to the computer
  - 2.1.2 Sign-on
  - 2.1.3 Selecting a lesson
  - 2.1.4 Correcting typing mistakes
- 2.2 Sending comments to the author of a courseware module
- 2.3 Sending messages
- 2.4 Student performance records
- 2.5 Computerized testing
- 2.6 Choosing an access-code
  - 2.6.1 Changing an access-code

for authors

3.0 CREATING CDS Courseware

- 3.1 New lesson files
- 3.2 Old lesson files
- 3.3 Using prototypes
- 3.4 Creating sign-on records
- 3.5 Deleting frames, signons, files

4.0 Core CAI Design Language (CCDL)

- 4.1 Introduction
- 4.2 "program" structure
- 4.3 arithmetic expressions
- 4.4 function descriptions
- 4.5 Prototype level functions

for programmers

5.0 Programmer use of CDS

- 5.1 Editing
- 5.2 Constructing prototypes
- 5.3 Internal variables for standard prototypes *DRILL and *LDL

6.0 Using the translator

- 6.1 Translation utilizing existing rule-sets
- 6.2 Creating and testing rule-sets

for potential implementers

7.0 PDP-11 CDS System Construction

7.1 System overview
7.2 Intermodule communication
7.3 System security
7.4 Module linkage
7.5 Terminal interface
7.6 Internal courseware format
7.7 Instruction format
7.8 External courseware storage
7.9 Runtime memory usage
7.10 Required resources
7.10.1 Operating-system requirements
7.10.2 Hardware requirements
7.11 Constraints imposed by PDP-11 hardware and system

8.0 CDS internal structure

8.1 Introduction
8.2 HELLO -- module description and flowchart
8.3 CDS system variables, tables and user-related areas
8.4 CDS--Module descriptions & structured flowcharts

9.0 CPP (printout creation) internal structure

9.1 Module descriptions and structured flowcharts

10.0 CTT (translator) internal structure

10.1 Module descriptions and structured flowcharts

--- installation & maintenance

11.0 Installation guide

11.1 RSX11m
11.2 RSTS
11.3 RT11

12.0 An Advanced Interaction-oriented Language

12.1 Implementation levels
12.2 TUTOR-IV-compatible graphics
12.2.1 Primitive Graphic Functions
12.2.2 CCDL Graphics

1.0 Introduction

1.1 Rationale -- basic goals

There are two very pressing problems addressed by the CAI Design System described in this document: 1) the augmentation of the CAI courseware creation process by a sophisticated authoring system, and 2) the need to transfer completed courseware to as many student-oriented systems as possible, in order to decrease unit cost of utilization.

1.1.1 The Authoring Process

The creation of Computer Aided Instruction (CAI) courseware has for too long been viewed as just another PROGRAMMING application of the computer. This view is amply illustrated by the PLATO-IV system (Smith and Sherwood, 1976), with its highly sophisticated language, TUTOR-IV (Sherwood, 1975; Avner, 1975) capable of expressing any function a courseware author could require. This extremely flexible and versatile language has an easy-to-learn core of commands which may be used by novice courseware authors, but most of the sophisticated and highly interactive courseware is written by a small group of virtuoso programmers which has developed among those constantly using the system. Courseware for PLATO must always be transformed into programming terms before it is put on the computer.

For a number of reasons the programming approach seems undesirable for many potential CAI courseware authors. It would be far more desirable to have CAI authoring systems which interact more directly with the EDUCATIONAL needs of courseware authors, rather than requiring either an author who is also a programmer or requiring the sometimes-difficult communication between author and virtuoso programmer.

As a first step in an experiment to determine whether the interaction can be carried out at a higher level, the CAI Design System (CDS) attempts to raise the level of communication between author and computer above the programming-language level.

1.1.2 Courseware Transfer

Once courseware has been created, it must be shared. Authoring can take 40 to 200 hours (author time) per completed student-hour of material. The cost of this time can only be justified if the finished material is widely distributed and used. Currently there are two major approaches to courseware sharing (or "transport") : 1) the central-computer system; a single computer system with thousands of computer terminals connected, and 2) the physical shipment of programs from one system to another. In the central-computer approach, courseware can be instantly made available to all users of the computer when it is completed. Modifications (and, of course, any programming errors so-introduced) are similarly available to other system users.

The alternative, physical shipment, is fraught with difficulties -- language incompatibilities, differences in language dialects, the inability of the original author to update all versions of courseware previously distributed, etc.

The ideas contained in this paper relevant to courseware transfer will only be applicable to systems utilizing physical transport methods.

1.2 Design objectives

The two overall objectives of CDS, sophisticated authoring facilities and courseware transport, were outlined in the previous section. In the current section, specific design objectives to be used during system implementation and testing will be described.

1. The CDS is to provide courseware-authoring capabilities ABOVE the CAI programming-language level. A CAI language will also be provided for those who wish to use it.

2. A library of instructional strategies, modifiable for individual authors' use, is to be provided.

3. A rule-driven translator will convert CDS courseware into other CAI languages for large-scale student use on other computing systems.

4. CDS is to be primarily an AUTHORING system, but will serve peer-reviewers and small student groups for the purposes of courseware testing and validation.

5. Student-data recording need not be supported on CDS computer systems, but CDS and its translator will produce courseware capable of utilizing data-recording facilities of target CAI systems.

6. The system will be implemented compactly, thus requiring minimal hardware for its use.

7. "System" functions, beginning with sign-on bulletins, message switching and examination of student-data from courseware, are to be delegated to CCDL* modules, to reduce the bulk of code in the CDS program itself.

8. CCDL* is to contain a small core of functions (commands) with a rich repertoire of syntax-dependent options for each.

---

* CCDL: the "CAI-programming" language used within CDS.

1.2.1 Advanced-system goals

The goals just described were prepared to guide the construction of the CDS as it now exists (1977). A second set of goals has been developed to guide the construction of the "advanced" CAI Design System (now designated the Learning Design Language). These goals are not currently implemented in operational programs.

A. Full TUTOR capabilities, (except for functions required for mechanical or structural reasons (Catchup, Background, Return, Lesin and some others).

B. "Transparent" creation and modification of graphics; no digitizing is to be done by authors. (Includes PLATO's insert-draw and see-draw features.) These graphic functions are specified in chapter 12.2 .

C. Instant switch between RUN and EDIT modes, on a frame-by-frame basis.

D. TICCIT-like systems which separate content and strategy are to be implementable, and may coexist with TUTOR-like programming systems. TICCIT-like content and strategies to be supported include:

1-- map & objective structures,
2-- student records for map and objectives
3-- advisor for map structures, (requires data-logging)
4-- learner-control module,
5-- advisor at learner-control level,
6-- file-structure for learner-control,
7-- editing of files,
8-- special-purpose (programming) modules.

E. Color graphics and videodisk control.

1.3 System Overview

In spite of the fact that CDS includes several computer programs, the user sees it as a unified whole. The user is more likely to view it as consisting of its two major operational modes:

a) the RUNNING (or execution) of CDS courseware modules, and
b) the EDITING (creation and modification) of courseware modules.

An individual requesting access to CDS identifies himself to CDS by "name" and "user-group". An "access-code" (password) is used to verify the individual's claimed identity. Other user-specific information is retrieved by CDS, including a "user-type" (student, author, programmer, advisor, coordinator or visitor) and some accounting information. Once identity is validated, s/he either RUNS courseware as a "student" or EDITS as an "author" might. In RUNNING, CDS uses stored instructions (or COURSEWARE) in the Core CAI Design Language, to carry out an interaction with the computer user. The interaction may be an actual lesson in some subject area, or it may be a "utility" program such as the one used to send messages from one user to another, or such as the student-data reporting program. In the EDIT mode, CDS allows changes to be made to existing courseware, and the creation of new courseware.

The EDITING mode is subdivided into two major LEVELS of operation. The more basic is the PROGRAMMING level, at which the user can insert, delete or modify lines of code in the Core CAI Design Language within each piece of courseware. Above that level is the PROTOTYPE level, at which the author may interact with small previously-stored prototype frames -- the interaction is one in which the computer asks questions and the author answers them. The answers to these questions supply a choice of instructional strategy and the content of frames.
"fills in" the blanks in formats previously defined and tested.

At the higher level of interaction, the ultimate goal is still the creation of a courseware module at the lower (or programming-language) level. All of this ultimate programming is done in a subset of CDS's language, the "Core CAI Design Language".

When courseware has been adequately tested by its author, by peers and students, it is passed through the CDS rule-driven translator to be converted to another CAI language suitable for use on a particular target (student) system. Because rules can be created or modified at any time, translated courseware may be adjusted to suit the local dialect of a CAI language, or new language-targets may be chosen whenever desired. The translator also makes possible the direct use of other CAI languages by authors on the Design System, utilizing "auto-translation".

The text of all courseware is protected by change-codes and inspect-codes chosen by original author(s).

Because many "system" functions are actually implemented as CCDL programs rather than as additional overlays to the CDS program itself, such operations as message switching may be tailored quickly to local requirements. This also removes much code which would otherwise have been implemented in assembly language.

CCDL courseware may also access user-specific information such as a person's user-type to validate access to privileged data or operations. For instance, student data are available only to advisors, and to the student himself. The REPORT program thus verifies an individual's user-type upon entry to determine the extent of his or her access to available information.

2.0 USING CDS Courseware

No matter what the user-type of a particular individual may be -- student, advisor, programmer, administrator, author or visitor -- s/he has access to both of the two basic modes of operation, RUNNING and EDITING.

This section of documentation describes functions performed in the RUNNING mode of CDS. It includes the "sign-on" procedure, the process of requesting a courseware module, special facilities supported during the running of courseware, and "sign-off" at completion of interaction.

2.1 Sign-on

All users of CDS on a particular computer system are identified to CDS by NAME and USER-GROUP. Associated with each name in a user-group there may also be an access-coude, a user-type and a router-lesson name.

Name:
This is a 1 to 8-character word (usually the last name of the student or user). If spaces are used, they are significant (such as between a first initial and last name), though multiple spaces are compressed to a single space, and punctuation marks are changed to spaces before the name is interpreted. For example, some valid sign-on names include:

Jimmy
Gabriel
Aaron S
S.Smith

Note that S.Smith could also type S Smith to signon, but that SSmith is not equivalent because there's no space in it.

User-group: (optional, but always assigned at UOP)
This is a 1 to 9-character word (usually the course, or the individual's sponsoring organization, at UOP). Here, spaces, punctuation and capitalization are all insignificant. Some valid user groups include:

Med School
UoP
Hospital
A101

Access-code: (optional, but always assigned at UOP)
This is a 1 to 8-character "word" which protects a signon against unauthorized use. Any combination of letters, digits and other ASCII characters, including spaces and punctuation, may be used. A user may change his or her password at any signon time. It is initially assigned by the individual who creates the sign-on record, usually the CAI Coordinator.

User-type: (optional, but always assigned at UOP)
This is a word describing the privileges accorded to this individual on the computer. At UoP, user-types in current use are:

Coordinator, Student, Advisor, Author, Administrator, Programmer and Visitor.

Because user-type (UT) is used primarily by CCDL courseware, new categories may be created at any time for any particular installation. The next paragraph describes some of the CDS system uses of user-type.

One of CDS's uses of UT is in the recordins of
averase student courseware-comPletion times --
only UT="Student" causes recordins.

For Author, Prosrammer, Advisor or Coordinator, an
@ issued in resPonse to any PromPt can also cause
transfer of control to a new frame.

Another use of UT is that only an "author"
or "Prosrammer" may create a new courseware file.

And finally, only the "coordinator" may create
sisn-on records files. (Thoush, in the case of
courseware and sisn-on files, ANYONE who knows
the chanse-code may make chanses once the file
has been created.)

<u>Router:</u> [oPtional]
This is a lesson name, and consequently contains a maximum of
nine sisnificant characters. The user is sent to
this lesson uPon comPletion of sisnon, uPon comPletion of
any other lesson the router may invoke, or uPon ESC.
The effect of sPecifysins a router is to keeP the user
totally away from the WHAT LESSON DO YOU WANT level.
(Of course, the router itself may offer the student a
choice of lessons to be run, and may JUMP to those the
user selects.)

2.1.1 Connecting to the computer

The following section illustrates the process of connecting to the PDP-11/34 computer at University of the Pacific's San Francisco campus, the location where CDS was developed.

If you're the really independent type (or if you already have some experience with computers),

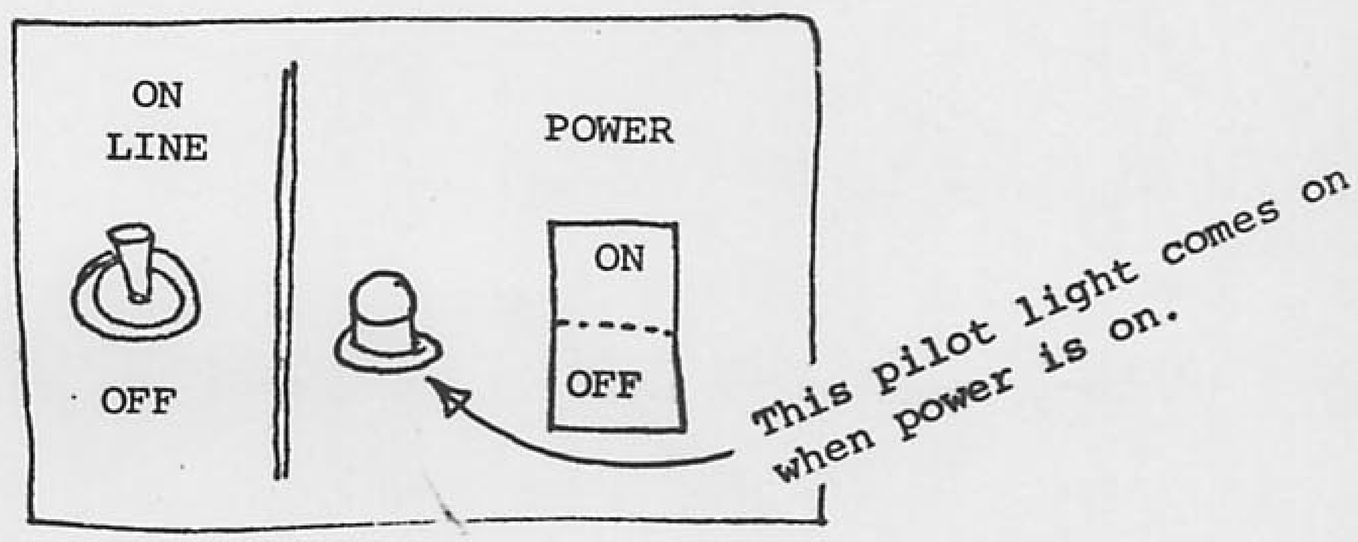
Go down the LEFT HAND COLUMN of "actions". If you have trouble, or are curious, look at the explanations in the right hand column.

If you're new to computers (or if you like be sure you fully understand things before doing them),

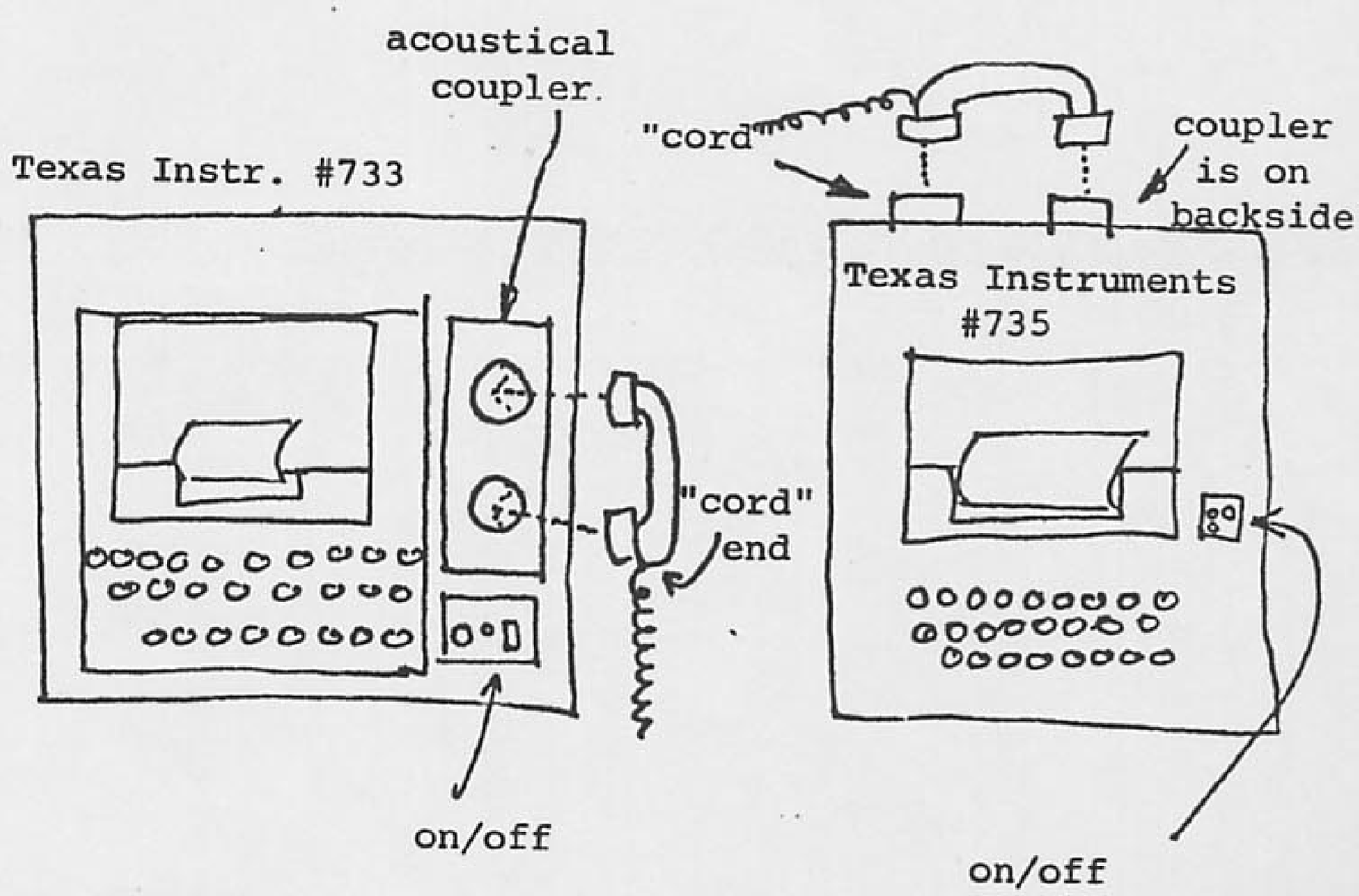
Go down the RIGHT HAND COLUMN of "explanations" first. Be sure you understand them fully, then go back to the beginning of the instructions and perform the Actions.

CONNECTING TO THE COMPUTER

Actions	Explanations & trouble-shooting
1. Turn the computer terminal ON	The diagram shows one common arrangement for on/off switches (on a Texas Instruments terminal). Both the main power and the acoustical coupler's power must be ON! The coupler switch is sometimes just a key on the keyboard, and it is often marked ONLINE, LINE, or just plain ON.



2. Locate the acoustical coupler	Be sure you determine which end of the acoustical coupler is the CORD end. This is the end which MUST receive the telephone's transmitter or "cord" end. Some couplers have the word "CORD" printed here -- others don't. The diagrams below will help you with Texas Instruments 733 or 735's:



Action	Explanations & trouble-shooting
3. Pick up the phone and call the computer*	Calling the computer is easy:
THEN IMMEDIATELY	IF IT ANSWERS, you'll hear a high-pitched tone. Go immediately to the next step. IF IT IS BUSY, hang up and try again in a few minutes. IF IT DOESN'T ANSWER, something is wrong at the computer end. Hang up and try again in a few minutes. If it doesn't answer (after a few heart-felt attempts by you), you may wish to call the Division of Respiratory Therapy for possible help or information on the computer's "health".
4. Immediately, place the telephone handset into the acoustical coupler	See the diagrams on the previous page. They indicate how the handset fits into the coupler.
	YOU HAVE ABOUT 15 SECONDS to do this before the computer hangs up on you. If it does hang up, go back to step 3 to call it again.
	IF YOU'RE SUCCESSFUL, the "online" light on the terminal or the acoustical coupler should come on, indicating that you're connected properly to the computer.

* (See your local coordinator for phone number.)

2.1.2 Sign-on

This section covers procedures once your terminal is connected to the computer via phone. The prospective CDS user MUST have arranged a sign-on name, user-group and access-code with a supervisory staff member before attempting sign-on.

SIGNING ONTO THE COMPUTER

Actions	Explanations & trouble-shooting
A. Type "hello", then press the CR or RETURN key	Once you're connected to the computer, you must signal CDS that you're ready to proceed. You do this by typing "hello" and pressing the CR or RETURN key. The computer will answer: HELLO--WHAT IS YOUR NAME? : If it doesn't respond, first be sure the acoustical coupler is in the ONLINE or LINE position. If it still doesn't respond, hang up and try again later. If , after several attempts, you just can't get it to work, call the Division of Respiratory Therapy.
B. Type your sign-on NAME	ALL INDIVIDUALS using CDS will be assigned sign-on names, user-groups and access-codes by the CAI Coordinator. Your sign-on name must be typed EXACTLY as specified. BE SURE YOU PRESS THE "CR" or "RETURN" key at the end of your name -- this signals CDS that you've finished! You must always press RETURN at the end of each and every line you type from now on. CDS will accept your sign-on name and ask: WHAT IS YOUR USER-GROUP? : *

---

* Note: User-groups are assigned to all users at the University of the Pacific, but other CDS installations may not do this. In such cases, CDS only asks for a user-group when the name given is not found in the main records file.

Actions	Explanations & trouble-shooting
C. Type your user-group name	Type the name of the group to which you were assigned by the CAI coordinator. As with your "name", spelling must be exactly right. (DID YOU PRESS THE "RETURN" key at its end?) CDS now checks your sign-on name against the valid names in the user-group you requested. IF CDS SAYS: STUDENT RECORDS ARE MISSING OR LOCKED! then the user-group you named does not exist (OR you mis-typed it). The computer will hang up the phone, and you'll have to call it again. IF CDS SAYS: YOUR NAME IS NOT REGISTERED! then your user-group exists but the sign-on name you typed was not found in it. Usually this is because you mis-typed your sign-on name, but it may be because the CAI coordinator has not yet entered your name. If you're getting an error message even though you're SURE you typed the right things, call the Division of Respiratory Therapy.
D. Type your access-code	CDS , once it's verified the existence of your name within the user-group, will request your access-code (if one was specified for your record): ACCESS-CODE: @@@@@@@@@@ CDS "blacks out" the area where you're going to type, so that nobody will see the code while you type. On certain kinds of terminals, it will also echo "x's" when you type. And, once you finish typing the access-code, it will go back and cover the line with "x's" to be sure.

Actions	Explanations & trouble-shooting
	IF CDS SAYS: WRONG! you've typed something other than your correct access-code. You'll have to call the computer again. If you get the access-code right, CDS will print a short bulletin, the time and day, and will search for any messages left on the computer for you. Those users for whom a router lesson has been specified will be sent directly to the router, skipping step "E" below.
E. Type the name of the "lesson" you want	CDS finishes showing the bulletin and asks: WHAT LESSON DO YOU WANT? : Instructions for beginners, and hints on selecting a lesson appear in the next section.

2.1.3 Selecting a lesson

Actions	Explanations
Experienced users: type a lesson name	Any CDS courseware module, whether it is really a "lesson" or not, will be called a "lesson". You may select ANY courseware module, or lesson, at this point in the interaction.
To "explore": type "index"	INDEX contains abstracts of Respiratory Therapy instructional CAI modules available on the UOP computer. The CAI COMPUTER WORKBOOK issued by UOP includes all of the courseware abstracts from INDEX.
To END A LESSON: press the ESC key	After you've selected a lesson and it's running, you can always terminate it -- simply press the ESC (escape) or ALT key any time CDS is waiting for a response to a question. You will return to: WHAT LESSON DO YOU WANT? : (Users for whom a router lesson was specified will be sent to the router instead.)
To END THE SESSION: hold down the CRTL key and press "Z"	To terminate your session with CDS altogether, hold down the CTRL (control) key and depress the "Z" key. This is called control-z. It terminates your interaction readying CDS for the next user. The computer will hang up the phone line -- be sure you hang up your end!

IMPORTANT! Be sure to use control-z to terminate your session. If you don't (that is, if you just hang up the phone), the next person who calls the computer on that phone line will already be signed on, using YOUR NAME, and can see YOUR RECORDS, or may send messages to other computer users using YOUR NAME. It could be embarrassing, or expensive (if you pay for your computer time).

2.1.4 Correcting typing mistakes

If you mis-type a single character, most PDP-11 operating systems will allow you to depress the DEL or RUBOUT key once to have it ignored. You then type the CORRECT character and continue. To have any particular number of characters ig-nored, one depresses the DEL or RUBOUT key the corresponding number of times, then re-types.

In the example below, a user who wishes to type "California" has made a mistake and corrected it. Note the presence of the "\" in the line when the user presses the DEL key, and again when correct typing resumes:

```
        Calf\f\ifornia
```

This results in a corrected input of "California".
This error correction is controlled by the RSX11M operating system.
On some telephone lines connected to the computer, the DEL or RUBOUT key causes actual backspacing. The mistake is then cor-rected by retyping (overtyping) with the correct characters:

```
        Calfforn ia
```

To cancel the whole line and begin again, one holds down the CTRL (control) key and presses "U". CDS or the operating system will cancel everything previously typed, will move to the next line on the paper, and will wait for the answer again.

As an example:

```
        Califreo^U                  (User pressed CTRL-U, computer sends a
        California                   carriage-return and waits again.)
```

Note to system implementers: On RSX11M, "TT" ports have DEL and ^U controlled by the operating system. On RSX11M "XL" ports, CDS controls DEL and ^U.

2.2 Sending comments to the author of a courseware module

Anybody with a sign-on name may send feedback or comments to the author of a courseware module, while they are running that module.

When using the lesson, any time CDS is waiting for input, one may answer with an "@" sign, followed immediately by RETURN. Control is transferred to a message-sending program, which records comments for the author. When the comment is finished, control returns to the lesson.

Note to system implementers: the message-sending functions of CDS are carried out by CCDL programs, NOT by CDS itself. Thus, once the "@" has initiated transfer to the message-switching module, there is no additional CDS overhead imposed.

EXAMPLE:
Below you see a student interacting with a courseware module covering personnel problems. At the arrow, where she would normally answer the question, she decides to send a message to the author of the module.

Lesson

```
WHICH OF THESE COVERS ILLNESS OR INJURY CONNECTED WITH WORK?
[ANSWER WCI, SDI OR SUI]
>@
```

sending the comment

```
^^^
```

```
HELLO
HELLO--WHAT IS YOUR NAME?
>SKY
WHAT IS YOUR USER-GROUP? UoP
ACCESS-CODE:
■■■■■■■■■■■■■■■■
GOOD EVENING! 8:49 P.M.
TUESDAY, 18 OCTOBER, 1977.

NO NEW MAIL.
YOU HAVE 1 MESSAGES TO REVIEW.
WHAT LESSON DO YOU WANT?  : MAIL
*************************
MAIL
*************************
812 MESSAGES SENT THRU THE COMPUTER TO DATE.
        TYPE SEE TO SEE NEW MESSAGES,
             SEND TO SEND A MESSAGE.
             REVIEW TO REVIEW SAVED MESSAGES.
             CLEANUP TO RETURN UNDELIVERED MAIL TO SENDER,
             HELP FOR SHORT EXPLANATIONS AT ANY POINT.
>SEND
WHO IS YOUR MESSAGE DIRECTED TO:GABRIEL
WHAT COURSE OR USER-GROUP IS GABRIEL IN?UoP

>DO YOU WANT TO HAVE LUNCH ON THURSDAY? 12 NOON ?
>_ cr
DO YOU WISH TO SEND A COPY TO SOMEONE ELSE? (YES/NO) NO
        TYPE SEE TO SEE NEW MESSAGES,
             SEND TO SEND A MESSAGE.
             REVIEW TO REVIEW SAVED MESSAGES.
             CLEANUP TO RETURN UNDELIVERED MAIL TO SENDER,
             HELP FOR SHORT EXPLANATIONS AT ANY POINT.
> esc
WHAT LESSON DO YOU WANT?  :
```

Signon

"Sky" of user-group "UoP" signs on and sends message to "Gabriel" of "UoP".

```
>HELLO
HELLO--WHAT IS YOUR NAME?
>GABRIEL
WHAT IS YOUR USER-GROUP? UOP
ACCESS-CODE:
■■■■■■■■■■■■■■■■
GOOD EVENING! 8:52 P.M.
TUESDAY, 18 OCTOBER, 1977.

UNIVERSITY OF THE PACIFIC (SAN FRANCISCO) &
INSTITUTES OF MEDICAL SCIENCES, PACIFIC MEDICAL CENTER
(SYSTEM OPERATES 24 HOURS A DAY EXCEPT MONDAY 9AM TO NOON)


THERE IS NEW MAIL FOR YOU.
WHAT LESSON DO YOU WANT?  : MAIL
************************
MAIL
************************
813 MESSAGES SENT THRU THE COMPUTER TO DATE.
        TYPE SEE TO SEE NEW MESSAGES,
             SEND TO SEND A MESSAGE.
             REVIEW TO REVIEW SAVED MESSAGES.
             HELP FOR SHORT EXPLANATIONS AT ANY POINT.
>SEE

FROM:SKY          OF USER-GROUP:UOP
SENT ON 10/18/77
AT 20:50
DO YOU WANT TO HAVE LUNCH ON THURSDAY? 12 NOON ?
"DELETE" OR "SAVE" THIS MESSAGE?DEL
DO YOU WISH TO REPLY NOW?YES
TYPE YOUR REPLY:

>LUNCH SOUNDS FINE. I'LL BE AT YOUR OFFICE AT NOON.
>
        TYPE SEE TO SEE NEW MESSAGES,
             SEND TO SEND A MESSAGE.
             REVIEW TO REVIEW SAVED MESSAGES.
             HELP FOR SHORT EXPLANATIONS AT ANY POINT.
>^Z

GOODBYE
```

Signon

"Gabriel" of user-group "UoP" signs on and gets the message...

Reply

2.4 Student performance records

The prototype interaction frames in use at University of the Pacific provide for creation of and maintenance of records on student performance within instructional CAI modules. Student-users of the computer may access a lesson called "REPORT" to examine records of their own performance on certain courseware. These records are also available to all Advisor users, but to nobody else.

Note to system implementers: REPORT and all programs which record data for its use, are CCDL programs which impose no direct overhead on the CDS program.

EXAMPLES:

Below you will see illustrated two of the functions of the REPORT lesson. First, an advisor signs on and asks for a complete list of students working on modules. Then, on the next page, the same advisor asks for a report on a single student's progress in a particular module ("Atelectasis" in this case).

```
WHAT LESSON DO YOU WANT?  : REPORT
**********************
REPORT STUDENT/MODULE DATA
**********************
AS AN ADVISOR YOU MAY:
        LIST ALL STUDENTS & THEIR ACTIVE MODULES
        UPDATE A STUDENT'S RECORD
        REPORT A STUDENT'S STATUS IN ANY MODULE
        DELETE A STUDENT OR A MODULE
(TYPE THE KEYWORD FOR YOUR CHOICE)
>LIST
CARLA    4 MODULE(S).
         1...MED-GAS
         2...AEROSOL
         3...ASTHMA
         4...AIRWAY
SCHMO    1 MODULE(S).
         1...VENTFAIL
KATHY    2 MODULE(S).
         1...ASTHMA
         2...ATELECTA
SUE    1 MODULE(S).
         1...ASTHMA
```

```
WHAT LESSON DO YOU WANT?  : REPORT
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
REPORT STUDENT/MODULE DATA
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
AS AN ADVISOR YOU MAY:
        LIST ALL STUDENTS & THEIR ACTIVE MODULES
        UPDATE A STUDENT'S RECORD
        REPORT A STUDENT'S STATUS IN ANY MODULE
        DELETE A STUDENT OR A MODULE
(TYPE THE KEYWORD FOR YOUR CHOICE)
>REPORT
WHAT STUDENT? [CR TO ABORT REPORT]
>KATHY



KATHY:      2 ACTIVE MODULE(S).
        1...ASTHMA
        2...ATELECTA
WHAT MODULE ARE YOU INQUIRING ABOUT?
>2
-------------------------------------------------------------------
        BACHELOR OF SCIENCE AND AFFILIATED PROGRAMS
                IN RESPIRATORY THERAPY
              UNIVERSITY OF THE PACIFIC
-------------------------------------------------------------------
                                                STUDENT:KATHY
                                                MODULE: ATELECTA
                                                18-OC-77

[REPORT CALLED UP BY:SKY]

        TODAY IS        :18-OC-77
        MODULE BEGUN    :19-AU-77

         (TEST)  :(DATE) (SCORE) (PROCTOR)
OBJECTIVE 1 ------------------------------- 1
OBJECTIVE 2 ------------------------------- 2
        ◆◆◆SELF :19-AU-77 91%
```

2.5 Computerized testing

Some of the criterion-tests for Division of Respiratory Therapy instructional modules are available on the computer. Once authorized by an administrator, the test is requested by a student's entering lesson "TEST".

Results for pre-module tests and post-module tests are stored so they are accessible to the REPORT program.

Note to system implementers: TEST and all programs connected with it (including REPORT), are CCDL programs which impose no direct overhead on the CDS program.

EXAMPLES:

This example shows some of the options available to an advisor. First, a listing of one of the items from the item bank for this module (module 400 is "Aerosol & Humidity Therapy").

Then, the advisor requests a listing of item 77, which is an essay question. When the test is being administered, the student would be shown the TEXT only, and would be asked to write an answer on paper. An advisor would score the item by looking at the guidelines given in the "answer" section (which he would have available separately). The computer can also print a test on the terminal, or on a high-speed printer, along with an answer key to be used for grading.

Next, the advisor requests a listing of which questions in the item bank pertain to which objectives from the instructional module. Students have the ability to choose the objectives they wish to be tested on in any session.

Finally, a sample of the test is shown. This sample was printed out for the advisor, but is exactly what the student would see once the test had been authorized for him by an advisor.

```
WHAT LESSON DO YOU WANT?  : TEST
************************
TESTING SYSTEM
************************
WHAT IS THE MODULE NUMBER?
>400
MODULE: AEROSOL


TYPE:
        "ACTIVATE" TO SET UP TESTING FOR A STUDENT,
        "UPDATE" TO UPDATE OR LIST THE ITEM BANK.
        "TRY" TO TAKE THE TEST AS IF YOU WERE A STUDENT,
        "PRINT" TO PRINT A TEST AND ANSWER KEY.
>UPDATE


UPDATING:    (97 ITEMS)
        "LIST" TO LIST OBJECTIVES AND/OR ITEMS,
        "OBJECTIVE" TO CHANGE SPECIFICATIONS FOR A SINGLE OBJECTIVE,
        "ITEM" (OR THE ITEM NUMBER) TO CHANGE A SINGLE TEST ITEM.
        "NAME" TO RESET THE MODULE NAME.
>ITEM 1
WHAT IS THE ITEM NUMBER? ("NEXT" TO ADD ITEM 98):
>25


ITEM: 25
        (DAY/MONTH/YEAR): 20/12/76
        0 CORRECT ANSWERS IN 0 TRIES
DURING HYPERVENTILATION AND/OR FEVER, THE INSATIABLE WATER LOSS FROM
THE PULMONARY TREE IS GOING TO BE APPROXIMATELY:
        A-- 250 ML
        B-- MORE THAN 250 ML,
        C-- LESS THAN 250 ML,
        D-- NONE OF THESE.
TYPE THE LETTER FOR THE BEST ANSWER:
        -----CORRECT ANSWER:B -----


(-CR- TO LEAVE ITEM) OR AN ITEM NUMBER
LIST TO SEE ENTIRE ITEM,
TEXT TO REPLACE ITEM TEXT,
INSERT TO ADD TEXT,
ANSWER TO SPECIFY CORRECT ANSWER.
NEXT TO MOVE TO THE NEXT ITEM
>_CR
```

25

ITEM: 77
(DAY/MONTH/YEAR): 21/12/76
0 CORRECT ANSWERS IN 1 TRIES
HOW IS A WATER VAPOR PRESSURE GRADIENT ESTABLISHED ACROSS THE RESPIRATORY MUCOUS MEMBRANE?
###
ANSWER: GASES INSPIRED THAT DON'T CARRY 43.8 MG OF WATER VAPOR PER LITER WILL ESTABLISH A GRADIENT BECAUSE THEY DO NOT EXERT A PARTIAL PRESSURE OF 47 MM HG ON THE RESPIRATORY MUCOSAL MEMBRANES. MANUAL OF RESPIRATORY THERAPY, CHAPTER 7.
-----CORRECT ANSWER:ESSAY -----

<-CR- TO LEAVE ITEM> OR AN ITEM NUMBER
LIST TO SEE ENTIRE ITEM,
TEXT TO REPLACE ITEM TEXT,
INSERT TO ADD TEXT,
ANSWER TO SPECIFY CORRECT ANSWER.
NEXT TO MOVE TO THE NEXT ITEM
>_

UPDATING: (97 ITEMS)
"LIST" TO LIST OBJECTIVES AND/OR ITEMS,
"OBJECTIVE" TO CHANGE SPECIFICATIONS FOR A SINGLE OBJECTIVE,
"ITEM" <OR THE ITEM NUMBER> TO CHANGE A SINGLE TEST ITEM.
"NAME" TO RESET THE MODULE NAME.
>LIST

MODULE: AEROSOL
OBJECTIVE 1: 80% IS THE MASTERY LEVEL. 11 ITEMS ASKED
THERE ARE 11 ITEMS IN THE BANK.
1,2,3,4,5,6,7,8,9,10
11,0,0,0,0,0,0,0,0,0
MODULE: AEROSOL
OBJECTIVE 2: 80% IS THE MASTERY LEVEL. 23 ITEMS ASKED
THERE ARE 23 ITEMS IN THE BANK.
12,13,14,15,16,17,18,19,20,21
22,23,24,25,26,27,70,71,72,73
74,75,76,0,0,0,0,0,0,0
MODULE: AEROSOL
OBJECTIVE 3: 80% IS THE MASTERY LEVEL. 8 ITEMS ASKED
THERE ARE 8 ITEMS IN THE BANK.
28,29,30,31,32,77,78,79,0,0
MODULE: AEROSOL
OBJECTIVE 4: 80% IS THE
THERE

```
TYPE:
        "ACTIVATE" TO SET UP TESTING FOR A STUDENT,
        "UPDATE" TO UPDATE OR LIST THE ITEM BANK.
        "TRY" TO TAKE THE TEST AS IF YOU WERE A STUDENT,
        "PRINT" TO PRINT A TEST AND ANSWER KEY.
>TRY
TEST ITEMS FOR THE FOLLOWING OBJECTIVES ARE AVAILABLE:
1
2
3
4
6
7
8
9
10
12
14
SELECT THE OBJECTIVES ON WHICH YOU WISH TO BE TESTED.
TYPE THE WORD "ALL" TO TRY THEM ALL, OR
TYPE ONE NUMBER EACH TIME THE ARROW APPEARS BELOW, AND
TYPE "0" (ZERO) WHEN DONE SELECTING:
>3
>_ cr
```

```
QUESTION:1
TRUE OR FALSE: GASES THAT ARE INSPIRED, THAT CARRY A VAPOR CONTENT
OF 43.8 MG PER LITER, EXERT A PARTIAL PRESSURE OF 47 MM HG.
[ANSWER T OR F]

>T
```

```
QUESTION:2
TRUE OR FALSE: A GAS CARRYING A VAPOR CONTENT OF 43.8 MG PER LITER
EXERTS A PARTIAL PRESSURE OF 100 MM HG.
[ANSWER T OR F]

>F
```

```
QUESTION:3
GIVEN A SAMPLE OF AIR AT 75 DEGREES FARENHEIT WITH A RELATIVE HUMIDITY
OF 60%, WHAT WILL HAPPEN TO THE RELATIVE HUMIDITY OF THAT SAMPLE IF
ITS TEMPERATURE IS INCREASED TO 98.6 DEGREES?
        A-- IT WILL REMAIN THE SAME,
        B-- IT WILL INCREASE,
        C-- IT WILL DECREASE,
        D-- NONE OF THESE.
TYPE THE LETTER OF THE BEST ANSWER:

>C
```

2.6 Choosing an access-code

When an individual is initially resistered with CDS, the CAI Coordinator consults him or her about a sisn-on name, and initial access-code. The access-code, at that point, is known only to the individual involved, and to the CAI Coordinator. It safesuards access to the individual's record because CDS requires that it be sivenn correctly before sisn-on is completed. When the individual FIRST sisns onto the computer it is recommended that s/he chanse the access-code so that only s/he and the computer know what it is. Thousn the new access-code is recorded on the computer, NOBODY has access to it, includins programmers, advisors and the CAI Coordinator. If the individual forsets it, it is possible for the CAI Coordinator to insert a new access-code.

Here are some hints on selectins an access-code:

a) Use a word which nobody can suess. DON'T use your first or middle name, your spouse's name, names of your kids, or your car license number.

b) Chanse your access-code from time to time -- say once a month.

c) You should use a word which is easy for you to remember, BUT from the standpoint of security it's better to use a set of "random" letters and numbers which would be impossible for anyone to suess.

d) When you're sisnins-on, beware the individual who may be watchins over your shoulder to "spy" your access-code. It's perfectly OK to ask them to look away for a moment.

It may seem silly to place so much emphasis on your access-code, but it DOES allow access to the computer, to your own student records and to MAIL (in which somebody could send any kind of messases under your name). On some systems, charses will be assessed to you for all time lossed assinst your sisn-on. Thus, mis-use could be expensive and/or embarrassins to you and to others.

2.6.1 Changing an access-code

Though an access-code may be changed by the CAI Coordinator, or by a user-group director, most changes are initiated by the "owner" of a sign-on. An access-code may be changed at any sign-on. Look at the example:

```
HELLO--WHAT IS YOUR NAME?  g_smith
WHAT IS YOUR COURSE OR USER-GROUP?  uop
ACCESS-CODE:
@@@@@@@@@@@ cr
```

During a NORMAL sign-on, the user types the access-code in the "blacked out" spaces, then presses the CR or RETURN key at its end.

To CHANGE ACCESS-CODE, however, the user types the "old" code in the "blacked out" spaces, then presses the ESC (escape) key at its end, rather than CR or RETURN. CDS says:

```
NEW CODE:
@@@@@@@@@@@
```

and waits for the "new" access-code. It must be typed CAREFULLY, SLOWLY, and CR or RETURN is used at its end. If any mistake is made during its typing, one may press ESC to cancel and type the new access-code once more -- however, once CR or RETURN is used, the code is permanently changed in the sign-on records.

Note to system implementers: The verification of and changing of access-codes is handled entirely by code in CDS. Access-codes, and courseware change- or inspect- codes are NOT available to CCDL programs or programmers.

3.0 Creating CDS courseware

The second major mode of CDS operation is the EDITING mode. To edit, one signs onto CDS (section 2.1) and when asked:

WHAT LESSON DO YOU WANT?

types the name of the lesson to be EDITED, then presses the ESC key rather than RETURN.

3.1 New lesson files

If no lesson of the requested name has been previously created, and if this user is a programmer or author, CDS will say:

THERE IS NO LESSON BY THAT NAME
DO YOU WISH TO CREATE ONE?

One answers YES or NO. (Or "Y" or "N")

For a YES answer, CDS requires more information:

TITLE: up to 77 characters -- short descriptive title for this lesson.

AUTHOR OR SOURCE: up to 47 characters -- use complete name rather than sign-on.

LESSON-ACCESS-CODES:

CHANGE-CODE: A password (like an access-code) which must be given by an individual who wishes to change the courseware file.

INSPECT-CODE: A password which, when correctly given by an individual, allows him or her to view the CCDL statements in a lesson but not to make changes.

TYPE 'C' FOR COURSEWARE FILE, 'S' FOR SIGNON RECORDS:

This option is offered only to users whose user-type (UT) is "Coordinator". Sign-on records files have a special attribute set, so they can be used by users signing onto the computer, but otherwise are like courseware files in terms of structure.
The creation of sign-on records is dealt with in section 3.4

DO YOU WISH TO ZERO STUDENT TIMING DATA? : YES or NO

Yes=zeros average completion times, session count, average time to ESC.

WHAT SIGNON IS TO RECEIVE FEEDBACK? : The sign-on name of the individual who is to receive the comments on this lesson made using "@".

WHAT USER-GROUP IS S/HE IN? : The user-group of this person.

Automatic translation:

If a file-extension is specified by the user, CDS asks: DO YOU WISH AUTOMATIC TRANSLATION?:
This requires a YES or NO answer. YES will cause CDS to activate the translator (CTT) whenever any frame is changed, to perform single-frame translation. A NO answer inhibits this translation. (The user may still request manual translation.) See section 6.1.3 for more on this subject.

```
WHAT LESSON DO YOU WANT?  : EXAMPLE 2 esc
THERE IS NO LESSON BY THAT NAME,
DO YOU WISH TO CREATE ONE? YES
LESSON TITLE:AN EXAMPLE LESSON SHOWING CREATION OF NEW FILE.
AUTHOR OR SOURCE:JIM SCHUYLER
CHANGE-CODE:
■■■■■■■■■■■■■■■■
INSPECT-CODE:
■■■■■■■■■■■■■■■■
TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
DO YOU WANT TO ZERO THE STUDENT TIMING DATA?YES
WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
WHAT COURSE OR USER-GROUP IS S/HE IN?UOP
DIRECTORY LEVEL-- WHAT NOW?
```

"A totally new lesson"

```
WHAT LESSON DO YOU WANT?  : EXAMPLE3.TU4
THERE IS NO LESSON BY THAT NAME,
DO YOU WISH TO CREATE ONE? Y
LESSON TITLE:TUTOR-IV EXAMPLE
AUTHOR OR SOURCE:JIM SCHUYLER
CHANGE-CODE:
■■■■■■■■■■■■■■■■
INSPECT-CODE:
■■■■■■■■■■■■■■■■
TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
DO YOU WANT TO ZERO THE STUDENT TIMING DATA?Y
WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
WHAT COURSE OR USER-GROUP IS S/HE IN?UOP
DO YOU WISH AUTOMATIC TRANSLATION?: Y
DIRECTORY LEVEL-- WHAT NOW?
```

"A new lesson, with file extension '.tu4' specified."

3.2 Old lesson files

If the requested file or lesson already exists, CDS will ask for its CHANGE-CODE. If the user gives the correct code, lesson information is printed:

```
WHAT LESSON DO YOU WANT? : AEROSOL
CHANGE-CODE?:
■■■■■■■■■■■■■■■■
************************
A GENERAL REVIEW OF AEROSOL AND HUMIDITY THERAPY
************************
AV. COMPLETION   15  MINUTES (OVER   36 RUNS. STUDENTS ONLY)
AV TIME TO ESC   11  MINUTES (OVER    2 RUNS. STUDENTS ONLY)
EDITING      =  7   HR 23   MIN
21:46 18-OCT-77
AUTHOR:SKY-
CREATED:
LAST EDITED:21-SEP-77
LAST USED:21-SEP-77
TOTAL USER TIME=  0   HR 27   MIN (ALL USER TYPES)
DIRECTORY LEVEL-- WHAT NOW?
```

3.3 Using prototypes

It is the prototype frames which provide CDS's flexibility as an authoring system. This section discusses the utilization of a prototype from the CDS library.

The CDS prototype library is named PROTO, and may be edited as a standard CDS file by those possessing the change-code. When a user requests a prototype frame which is not in his own courseware file, CDS searches PROTO to locate it as well.

Upon unitiation of an editing session, the user is taken to DIRECTORY LEVEL. Standard line-by-line editing of these files is described in chapter 5. To utilize a prototype to produce or modify courseware, one simply types the name of the prototype frame at DIRECTORY LEVEL. The name of a prototype always begins with an asterisk, and the use of the powerful set of prototypes developed at the Division of Respiratory Therapy (and stored in PROTO) is begun by typing "*DRILL".

The *DRILL prototype actually makes use of about a dozen other prototype frames of various names. Branching from one prototype to another is controlled by values stored in variables, and by the user's answers to questions asked by the prototypes.

Incidentally, any time the user wishes to escape from an interaction with a prototype, s/he may press the ESC key.

The EXAMPLE on the following pages illustrates the use of *DRILL. Note that the author begins work on a new lesson, "example 4", and at DIRECTORY LEVEL types "*DRILL". From then on, the rest of the first page is controlled by prototype frames.

On the second page, the author directly examines one of the frames produced by the prototype. The reader will notice that it contains a number of CALs, setting counters which control review, and IF/JUMP combinations providing branching. The "RV" frames control scoring and the review process itself.

```
WHAT LESSON DO YOU WANT?  : EXAMPLE 4
THERE IS NO LESSON BY THAT NAME,
DO YOU WISH TO CREATE ONE? Y
LESSON TITLE:EXAMPLE SHOWING USE OF *DRILL PROTOTYPE
AUTHOR OR SOURCE:JIM SCHUYLER
CHANGE-CODE:
■■■■■■■■■■■■■■■■
INSPECT-CODE:
■■■■■■■■■■■■■■■■
TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
DO YOU WANT TO ZERO THE STUDENT TIMING DATA?Y
WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
WHAT COURSE OR USER-GROUP IS S/HE IN?UOP
DIRECTORY LEVEL-- WHAT NOW? *DRILL
DRILL --  THIS PROTOTYPE CONTROLS CREATION OF DRILLS CONTAINING
        UP TO 25 ITEMS, INCLUDING REVIEW.
HAVE YOU ALREADY ENTERED SOME ITEMS FOR THIS DRILL?NO
TYPE YOUR INTRODUCTORY MESSAGE TO THE STUDENT:
SHOW  #THIS IS AN INTRODUCTION TO ELECTRICITY AND ELECTRONICS FOR
SHOW  #RESPIRATORY THERAPISTS.
SHOW  #_cr//
ITEM 1: WHAT TYPE OF ITEM WILL THIS BE?
        1-- STANDARD DRILL ITEM,
        2-- SERIES OF QUESTIONS USING A SINGLE COUNTER,
        3-- DISPLAY ONLY,
        4-- TUTORIAL WITH BRANCHING.
:1
ITEM 1:  ONE FORM OR TWO FORMS?  1
TYPE THE TEXT FOR THIS ITEM:
SHOW #THE STANDARD FREQUENCY FOR AC POWER IN THE UNITED STATES IS:
SHOW #  A-- 50 Hz
SHOW #  B-- 60 Hz
SHOW #  C-- 115 VOLTS
SHOW #  D-- 110 VOLTS
SHOW #_cr//
CORRECT ANSWER: B
FEEDBACK ("NONE" TO IGNORE): NONE
ARE THERE MORE CORRECT ANSWERS? : NO
ARE YOU GOING TO SPECIFY ANY ANTICIPATED INCORRECT ANSWERS? YES
INCORRECT ANSWER: C<OR>D
FEEDBACK FOR THIS ANSWER ("NONE" TO IGNORE):_cr//
SHOW #DON'T CONFUSE AMPLITUDE WITH FREQUENCY.  VOLTS MEASURE ELECTRICAL
SHOW #POTENTIAL OR AMPLITUDE FOR AC.  Hz (HERTZ) IS THE MEASURE OF
SHOW #FREQUENCY (CYCLES PER SECOND).
SHOW #_cr//
DO YOU WISH TO REPEAT THE FRAME FOR THIS ANSWER OR TO GO ON? YES, REPEAT
ARE THERE MORE ANTICIPATED INCORRECT ANSWERS? NO
DO YOU WANT FEEDBACK FOR UNANTICIPATED ANSWERS? YES
SHOW #PLEASE ANSWER A, B, C OR D.
SHOW #_cr//
DO YOU WISH TO REPEAT THE FRAME OR GO ON AFTERWORDS? REPEAT
WHAT FRAME DO YOU WANT TO WORK ON NEXT?
        "NEXT" FOR FRAME 2,
        "NONE" TO QUIT USING PROTOTYPES,
        "DONE" IF YOU'RE DONE WRITING THE DRILL  : DONE
DO YOU WANT THE STANDARD REVIEW INSERTED IN THIS LESSON? YES
HOW MANY ITEMS WERE THERE IN THIS DRILL? 10
WHAT PERCENT MASTERY IS REQUIRED ?  90
DO YOU WISH TO ADD A PARTING MESSAGE? >NO
```

```
DIRECTORY LEVEL-- WHAT NOW? F
************************
FRAMES ---
INTRO
1
RV
RV0
RV1
RV2
DATA
************************
DIRECTORY LEVEL-- WHAT NOW? @1
FRAME LEVEL-- [1] WHAT NOW ?LIST
************************
[1]  588  CHARACTERS
     1:QUESTION
     2:SHOW <ERASE>
     3:SHOW THE STANDARD FREQUENCY FOR AC POWER IN THE UNITED STATES IS:
     4:SHOW       A-- 50 Hz
     5:SHOW       B-- 60 Hz
     6:SHOW       C-- 115 VOLTS
     7:SHOW       D-- 110 VOLTS
     8:WAIT
     9:CAL V27=V27+1
    10:ANSWER <OK><SPELL>B
    11:CAL V1=V1-1
    12:CAL V28=V28+1
    13:IF [V26 IS TRUE]JUMP RV/1
    14:ANSWER <NO><SPELL>C<OR>D
    15:CAL V1=V1+1
    16:SHOW
    17:SHOW DON'T CONFUSE AMPLITUDE WITH FREQUENCY.  VOLTS MEASURE ELECTR
    18:SHOW POTENTIAL OR AMPLITUDE FOR AC.  Hz (HERTZ) IS THE MEASURE OF
    19:SHOW FREQUENCY (CYCLES PER SECOND).
    20:SHOW <END DISPLAY>
    21:JUMP /WAIT
    22:ANSWER <NO><UN>
    23:CAL V1=V1+1
    24:SHOW PLEASE ANSWER A, B, C OR D.
    25:SHOW <END DISPLAY>
    26:JUMP /WAIT
************************
FRAME LEVEL-- [1] WHAT NOW ?
```

3.4 Creating sign-on records

The *SIGNON prototype provides guidance needed to create new sign-on records.

A sign-on record file is initially created by the system Coordinator. This is done the same way any courseware file would be created (see section 3.1) except that when asked TYPE 'C' FOR COURSEWARE FILE, 'S' FOR SIGNON RECORDS:" the coordinator responds with "S".

Then at DIRECTORY-LEVEL, s/he types:

```
        *SIGNON

  WHAT LESSON DO YOU WANT?  : NLM
  THERE IS NO LESSON BY THAT NAME,
  DO YOU WISH TO CREATE ONE? Y
  LESSON TITLE:NLM SIGNON RECORDS
  AUTHOR OR SOURCE: CSG
  CHANGE-CODE:
  ■■■■■■■■■■■■■■■■
  INSPECT-CODE:
  ■■■■■■■■■■■■■■■■
  TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:S
  DIRECTORY LEVEL-- WHAT NOW? *SIGNON
  *SIGNON -- SIGNON RECORD(S) CREATION:
  WHAT IS THE SIGN-ON NAME YOU WISH TO CREATE? CHUCK
                    PERSON'S FULL NAME: CHARLES M. GOLDSTEIN
                    AFFILIATION ("NONE" IF NONE): NLM
                    USER-TYPE ("NONE" IF NONE): COORDINATOR
                    ACCESS-CODE ("NONE" IF NONE): XXX
                    ROUTER ("NONE" IF NONE): NONE
                    ANY MORE SIGNONS TO CREATE RIGHT NOW? YES
  WHAT IS THE SIGN-ON NAME YOU WISH TO CREATE? RICK
                    PERSON'S FULL NAME: RICK HOLT
                    AFFILIATION ("NONE" IF NONE): NLM
                    USER-TYPE ("NONE" IF NONE): COORDINATOR
                    ACCESS-CODE ("NONE" IF NONE): XXX
                    ROUTER ("NONE" IF NONE): NONE
                    ANY MORE SIGNONS TO CREATE RIGHT NOW? NO
  FRAME LEVEL-- [RICK] WHAT NOW ?LIST
  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
  [RICK]    51  CHARACTERS
      1:RICK HOLT
      2:NLM
      3:UT=COORDINATOR
      4:AC=
      5:TIME=      0
  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
  FRAME LEVEL-- [RICK] WHAT NOW ?
```

A sign-on record may be examined by individuals who know the change-code or inspect-code of the user-group. A typical record is shown below. This record was created by the *SIGNON prototype. It contains the user's name, access-code (AC=), user-type (UT=), and user time (TIME=) in minutes. Following that information, uses of individual lessons are recorded. Each contains the date, time (24-hour clock), number of minutes spent in the lesson. A special symbol appended to the lesson name indicates:

@ : that the lesson was edited by this individual
^ : that the user used ESC to terminate the lesson

```
WHAT LESSON DO YOU WANT?  : UOP
CHANGE-CODE?:
■■■■■■■■■■■■■■■■■■
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
UOP SIGNON-RECORDS
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
^^^^^^SIGN-ON RECORDS FILE^^^^^^
EDITING       =  0   HR 58   MIN
22:35 18-OCT-77
AUTHOR:SKY-UOP
CREATED:14-APR-77
LAST EDITED:18-OCT-77
LAST USED:18-OCT-77
TOTAL USER TIME=228   HR 47   MIN (ALL USER TYPES)
DIRECTORY LEVEL-- WHAT NOW? @SKY
FRAME LEVEL-- [SKY] WHAT NOW ?LIST
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
[SKY] 1012  CHARACTERS
    1:JAMES A. SCHUYLER
    2:AC=
    3:UT=COORDINATOR
    4:TIME= 6498
    5:$18-OCT-77 20:45   0 FIREUP
    6:$18-OCT-77 20:45   0@RECORDS
    7:$18-OCT-77 20:48   0 FIREUP
    8:$18-OCT-77 20:48   0 FIREUP
    9:$18-OCT-77 20:48   0 MAIL
   10:$18-OCT-77 20:49   0 FIREUP
   11:$18-OCT-77 20:50   1^MAIL
   12:$18-OCT-77 21:06   0 FIREUP
   13:$18-OCT-77 21:08   ^
   14:$18-OCT-77
   15:$18-
```

3.5 Deleting frames, sign-ons, files

At DIRECTORY-LEVEL, the user may delete individual frames by typing "D". CDS asks WHAT FRAME?, and the user responds with the exact name of the frame to be deleted.

A sign-on record file is composed of frames which have the names of the individuals who will be using the user-group. Thus, typing "D", and responding with the sign-on name to be deleted, performs the deletion function.

An entire courseware or sign-on file may be deleted at DIRECTORY-LEVEL by typing "ZAP". CDS asks whether you wish to permanently delete the file, and following a YES response, it will purge the file from the system directory.

4.0 Core C.A.I. Design Language (CCDL)

4.1 Introduction

All CDS courseware is eventually represented in terms of functions or commands in the Core C.A.I. Design Language. CCDL comprises a small set of commands, with a small but rich set of syntactic options serving to embellish these functions. The description of the CCDL given here will include some un-implemented functions, and is intended to provide full equivalence with TUTOR-IV graphics.

Each function is described in a standard format (which matches the format used by BCS in PILOT documentation developed for the National Library of Medicine). Each function's description includes:

SYNTAX
PROSE DESCRIPTION
LIMITATIONS
SYSTEM VARIABLE EFFECT
EXAMPLES
ERRORS and ERROR MESSAGES with CORRECTIVE MEASURES

4.2 "Program" structure

Each CDS courseware module, be it a lesson, program or sign-on records, is composed of a number of "frames". (This term "frame" is also equivalent to TUTOR-IV's "unit".) Each frame may be separately created, edited or deleted.

A frame is composed of CCDL "statements". Each statement is a single line of text. It is composed of a "function" and a "tag". The "function" directs CDS to a particular process, and the "tag" provides additional syntax-dependent information to be used in carrying out the function.

[Note: TUTOR-IV handles a maximum of 395 units per lesson!]

Within a frame, any statement may be labelled, and referenced from any other piece of courseware. A label actually consists of three parts:

lesson name
frame name
local label (a number)

The syntax of the JUMP function controls all branching.

4.3 Arithmetic expressions

CDS accepts the use of arithmetic expressions everywhere numbers may be used. Evaluation of an expression may include assisnment of the resultins value to storase.

An expression consists of any of the followins:

A user-variable ["V1" thru "V100"]
Strinss stored in a user-variable ["A1" thru "A100"]
A subscripted user-variable [V(expr)]
"expr" may be any expression which has a value between 1 and 100.
A number between 32,767 and -32,768.
A dataset variable [dataset name(expr)]
Binary operators combinins other valid expressions :
expr*expr
expr/expr
expr+expr
expr-expr
Any numeric system-variable or constant:
DAY: day of the month
MONTH: month of the year.
YEAR: year
TRUE:
FALSE:
VALUE: Equal to the numeric value of the student's last answer, if it was numeric.
LONG: The number of characters to be accepted by the next WAIT function. This variable may be set by the CCDL prosram.
TIMEOUT: Number of minutes CDS is to wait for the student to answer at the next WAIT function.
LINELIMIT: The number of lines to be allowed by CDS between WAIT functions.
JUMP: The number of JUMP functions executed by CDS since startup.
QJUMP: The number of "quick jumps" executed since CDS was started.
CAL: The number of CALS executed.
QCAL: The number of "quick CALS".
ALIVE: Minutes since CDS started.
ACTIVE: Number of current CDS users.
FFD: Number of frames fetched from disk since CDS was started.
DPR: Number of dataset pases read from disk since CDS startup.
DPW: Number of dataset pases written.

HOUR: hour of the day
MINUTE:minutes past the hour
SECOND:seconds into the minute
PORTS: Total number of user ports defined in this CDS.
STATION: Port number for the current user.

String variables and constants:

NAME: Student's name.
RESPONSE: Student's response to the last WAIT.
UT: User-type.
UG: User-group.
FINISH: Name of "finish" frame.
ROUTER: Name of this user's router lesson.
FRAME: name of the current frame.
OLESS: If this lesson was entered via a JUMP with return specified, OLESS has the name of the previous lesson.
OAUTHOR: As for OLESS, but contains name of the author of the previous lesson.
OFRAME: As for OLESS, but contains the name of the frame in the previous lesson.
OCOURSE: As for OLESS, but contains user-group for the author of the previous lesson.

The following items are presented only to illustrate work which would be performed to bring CDS up to TUTOR-equivalence. None are currently implemented.

Functions available in expressions:

TUTOR --- $union$, $diff$, $mask$, ars, cls, abs, sin, cos, log, ... arctan, exp, frac, int, ln, round, sqrt, not, (constants: pi)

TUTOR system variables to be considered:

backout, baseu, clock, lcommon, mainu, nhelpop, proctim, station (port), tactive(active), user(ut), usersin, error, mode, size, where, wherex, wherey, zreturn, ldone, lscore, rstartl, rstartu

TUTOR markups and variables:

spell, entire, phrase, capital, extra, order, vocab, anscnt, ansok, jcount, wcount, judsed, key, ntries

Setting CDS system variables:

Any of the following variables may be set by a CCDL program.

LONG: The value to which this is set determines how many characters CDS will accept at the next WAIT function before "forcing" a RETURN (for LONG greater than 1). If LONG=1, only one character may be typed by the student, but no RETURN is forced.

TIMEOUT: The value to which this variable is set determines how long (in minutes) a terminal may be inactive before CDS terminates. (Effectively a CTRL-Z) Default value is 10 minutes.

LINELIMIT: The value to which this is set determines how many lines of output may be printed on the terminal before CDS pauses and asks the user whether s/he wishes to continue. Default is 100 lines.

FINISH: (String) If set, this string variable gives the name of a frame to be executed when the student terminates the lesson with the ESC or CTRL-Z keys. The frame is executed before termination takes place.

BASE: (String) This string variable is normally set by KEYCODE interrupts (see the CALCULATE/KEYCODE description in section 4.4) and contains the name of the frame in which a WAIT function was being executed when the interrupt took place. Its value may be changed by the CCDL program.

4.4 Function descriptions

In the descriptions which follow, anything CAPITALIZED must appear exactly as-is in the actual statement. Items enclosed in brackets [] are optional.

4.4.1 Overall syntax

A CCDL line or statement begins with a function, and may contain a subsequent "tag" giving additional information. The TAG may also contain "embedded" information enclosed within markers ("<" and ">" are the markers). Once a CCDL function has been recognized on a line, the next step is to reduce embedded expressions to their values and substitute ASCII strings.

EXAMPLE:

```
        CAL v1=77
        CAL a10="AA"
        CAL A20="yes"
        SHOW  <v1> is the value.
produces the function line:
        SHOW  77 is the value.
```

And:

```
        JUMP  <a10>
produces the function line:
        JUMP  AA
```

And:

```
        ANSWER <SPELL><a20>
produces the function line:
        ANSWER <SPELL>yes
```

```
ANSWER [<feedback>] string-specifier
ANSWER [<feedback>]<UN>

string-specifier =  <EXACT> string    |
                    <SPELL> string    |
                    <KEYWORD> string  |
                    string-specifier <OR> string-specifier
```

ANSWER specifies answer-checking procedures. When a ANSWER function is encountered, CDS checks the string given in its tag against the last-given input from the terminal, using the qualifiers given (such as "exact", "spell" or "keyword"). If a match is obtained, CDS executes the next function from the current frame. If no match is obtained, CDS goes to the next ANSWER (or the next QUESTION) function in the frame. (At end of frame, control goes to the next frame in the courseware file.)

All strings to be checked must fit on a single input line. The input data will be 72 characters or less (not counting blanks), but CDS additionally limits input to 80 chars.

SIDE EFFECTS: None.

EXAMPLES:

ANSWER yes

CDS checks for a "yes" answer. KEYWORD and SPELL are set by default.

ANSWER <no> michigan

Again, SPELL and KEYWORD set by default. If the answer "michigan" is matched, CDS will print the feedback word "no" after the input line.

ANSWER <no><spell> california <OR> oregon <OR> washington

Any of the three answers will be accepted, or their misspellings. Feedback will be "no".

ANSWER <ok><exact> H2CO3 <or><spell> carbonic acid

The EXACT answer "H2CO3" will be accepted, or "carbonic acid" and its misspellings. The feedback printed will be "ok".

ANSWER <UN>

Will match only if no other previous ANSWER was matched. CDS executes the following function lines.

ANSWER (continued)

SYNTAX: A[NSWER] expression
A[NSWER] expression <+-expression[%]>

DESCRIPTION: Numeric answers are accepted from the student or user at all WAITs -- ANSWER is used to check them. If the user's answer is equal to the expression given (or within the specified range), the ANSWER-function is satisfied.

[THIS FUNCTION IS PLANNED BUT NOT CURRENTLY IMPLEMENTED.]

CALCULATE

SYNTAX: [CALCULATE] variable=expression

Rules for construction of expressions appear in 4.3

DESCRIPTION: CALCULATE uses the CDS expression processor to evaluate an expression and store its result(s). Storage may be in user-variables V1 thru V100, or in the dataset in use by the courseware module. Rules for the construction of expressions appear in section 5.3 .

LIMITATIONS: CAL is limited to one line in length. All results will be single precision integers (15 bits plus sign-bit).

SYSTEM VARIABLE EFFECTS: CAL may be used to set "long", "timeout", "linelimit" or "finish". Otherwise it has no effect on system variables.

EXAMPLES:

```
CAL v1=77
CAL v1=v55
CAL v(v1)=v(v1+1)
CAL data(v1*256)=0
CAL v1="Hello"
```

ERRORS: Attempts to store beyond V100 or below V1 are IGNORED by CDS. Syntax errors in expressions are IGNORED. The use of a dataset-name which is not referenced first in a DATASET command generates the message:

(CDS)--dataset name mismatch

CALCULATE (keycodes)

SYNTAX: C[ALCULATE] KEYCODE(expression)="framename"
C[ALCULATE] KEYCODE(Ø)=Ø

DESCRIPTION:
Each user of CDS has a corresponding KEYCODE table. Entries in this table consist of an ASCII keycode plus the name of a frame to be executed if that key is pressed.

At the beginning of each lesson, the KEYCODE table is cleared. Answers to all questions (WAIT functions) are treated normally. When a CALCULATE function is used to put an entry into the KEYCODE table, however, CDS begins processing answers on a character-by-character basis; if a key is pressed which coresponds to an ASCII value in the KEYCODE table, a JUMP is immediately made to the frame whose name is found in the table for that ASCII value.

CDS places the name of the frame in which the WAIT function appeared into the system variable BASE. It is the programmer's responsibility to issue a JUMP command to return to that frame when processing is completed for the KEYCODE which caused the "interrupt" in normal program flow.

The form of the keycode CALCULATE function is as follows:

"expression" -- must evaluate to an 8-bit ASCII keycode. The expression may be a quoted character (like "a", "b") or a numeric expression (like 97 for "a", 98 for "b"). Control characters are valid. Keys CTRL-U, CTRL-Z, ESC and CTRL-H (backspace) which are treated specially by CDS will not be processed in KEYCODE tables.

"framename" -- must evaluate to a frame name in the current lesson. This will almost always be a quoted string.

If "expression" is zero, the KEYCODE table is cleared and normal processing of subsequent WAIT functions resumes.

LIMITATIONS:
At most ten keycodes may be entered in the table.

SYSTEM VARIABLE EFFECTS:
The system variable BASE (a string variable) is set to the name of the frame in which the KEYCODE interrupt takes place. The BASE variable may be interrogated or set by IF and CALCULATE functions.

EXAMPLES:

```
frame 1--
  CAL KEYCODE("x")= "2"
  SHOW Try pressing the "x" key, or any other key.
  WAIT
  ANSWER <UN>
  JUMP 1
frame 2--
  SHOW <ERASE>
  SHOW You pressed "x"!
  JUMP <BASE>
```

The effect of this example program is as follows. When the student sees the message "Try pressing the 'x' key, or any other key." s/he may begin typing. As long as s/he doesn't press the "x" key, typing appears

as usual on the terminal. The student could simply make a full answer and press the RETURN key (as long as it didn't contain an "x").

If s/he presses the "x" key, BASE is set to the value "1" and frame 2 is executed. It clears the screen, types the message "You pressed 'x'!", and then re-executes frame 1. (Note that BASE is not cleared; it could be, but it doesn't make any difference in this example.)

In general, KEYCODE table entries correspond to special function keys a student may have on his or her terminal. These function keys are wired up so that they generate ASCII control characters when pressed, and CDS can therefore respond immediately to them by branching to an appropriate frame.

ERRORS:

Three error messages may be generated:

SYNTAX ERROR IN EXPRESSION -- if "expression" or "framename" is not a valid CDS expression (see section 4.3)

NO SUCH FRAME -- if "framename" is not the name of a frame in this lesson.

KEYCODE TABLE OVERFLOWED -- if more than ten entries are attempted.

Each of these messages causes termination of the lesson.

DATASET

SYNTAX: DA[TASET] name
DA[TASET] name=expr
DA[TASET]

DESCRIPTION: DATASET names an array of virtual storage to be made available to the program. Each array is permanent (surviving independent of program or courseware), and may be accessed by any number of courseware modules independently or simultaneously.

The "name" used in the DATASET function may subsequently appear in CAL functions, with subscript. If the "name=expr" form of DATASET is used, "expr" must contain a valid string giving the true name of the dataset -- "name", however must be used locally within CAL commands which follow.

DATASET with no "name" given, releases the dataset from use by the courseware module. Because DATASET is executed, it may be used to dynamically change datasets during execution of a courseware module.

LIMITATIONS: DATASETs are limited to 32,767 words of virtual storage (15 significant bits in the expression used as subscript).

SYSTEM VARIABLE EFFECTS: None.

EXAMPLES:

```
DATASET XYZ
DATASET N="pdq"    (CAL would use N(expr) in this case,
                     but the actual dataset used would be PDQ)
DATASET n=response
DATASET n=a(v44)
```

ERRORS: If a dataset is named which does not yet exist, it is immediately created -- no error message is printed. If the named dataset exists but CDS cannot attach it, the message "(CDS)--Unable to access dataset!" will be issued. The courseware module will be terminated.

DO

SYNTAX: DO framename

DESCRIPTION:

CDS executes the frame whose name is given in the tag. Upon completion of that frame, control returns to the function following the DO function in the current frame.

LIMITATIONS:

A frame which has been invoked by a DO function may itself contain a DO, until a limit of five outstanding DOs has been reached.

EXAMPLE:

```
DO  ABC
IF [v1 GT 77]DO XYZ
```

ERRORS:

If the limit of five outstanding DO functions is exceeded, the message

(CDS)-- DO-STACK OVERFLOWED

will be issued and the lesson will be terminated.

IF

SYNTAX: IF [expr] function

DESCRIPTION: An "if" clause, prefixed to any legal CCDL function, is evaluated, and IF THE VALUE OF THE EXPRESSION IS "true", the function is processed. If "false", the function is ignored.

LIMITATIONS:

SYSTEM VARIABLE EFFECTS: Evaluation of the expression MAY involve the setting of certain system variables. See section 4.3 on evaluation of arithmetic expressions for particulars.

EXAMPLES:

```
IF [v1 EQ 77]JUMP 5
IF [v1 IS true]SHOW Hello there.
IF [(v5 IS 55)AND(v6 IS 55)]CAL v7=v5+v6+v8
```

ERRORS: No error conditions are anticipated.

JUMP

SYNTAX: JUMP] (lesson)frame/label
JUMP]

DESCRIPTION: The JUMP function provides branching capabilities. JUMP with no tag terminates execution of the courseware module. The tag "/wait" causes a JUMP to the last WAIT function executed in the current frame.

If the tag of the JUMP is of the form:
JUMP <expr>
the expression is evaluated, and the number so created is used as the name of the target frame.

If the tag of the JUMP is of the form:
JUMP <string-expression>
the string is fetched and used as the target name. It may contain the full range acceptable for a label (lesson, frame, local label).

LIMITATIONS: The following forms are accepted:

```
(lesson)
(lesson)frame
(lesson)frame/label
frame
frame/label
/label
```

SYSTEM VARIABLE EFFECTS: None.

EXAMPLES:

```
JUMP pdq
JUMP (abc)1
JUMP /wait
```

ERRORS: If the label does not exist, one of the following messages may be printed. The module is aborted.

```
(CDS)--NO SUCH FRAME!
(CDS)--NO SUCH LOCAL LABEL!
(CDS)--NO PREVIOUS WAIT!
NO SUCH LESSON
```

SHOW

SYNTAX: [SHOW] string

DESCRIPTION: The "string" which appears is displayed on the user's computer terminal.

The string may contain embedded arithmetic or string expressions or special markers, as follows:

<expr> Any embedded expression's value is calculated and inserted in the string to be shown.

<NOCR> No carriage-return or line-feed is performed at the end of the string.

<ERASE> The screen is erased when the embedded marker is reached.

<END DISPLAY> On a CRT, CDS pauses, displaying "press RETURN to continue" at the bottom of the screen. When the user does so, CDS continues with the showing of the string. On printing terminals, there is no effect.

<LP> Subsequent printing in this and other SHOW functions is directed to the system printer.

<TI> Printing is redirected to the terminal.

^X CTRL-x's may be used to bracked sections of a string which are not to be shown.

LIMITATIONS: Each string is limited to 80 characters. The total length of the generated string (including embedded numbers, graphics, etc.) must not exceed 100 ASCII characters.

SYSTEM VARIABLE EFFECTS: None.

EXAMPLES:

```
SHOW Hello there.
SHOW <v1> is the value of v1.
SHOW <end display>
SHOW <erase>
SHOW <end display><erase>
```

ERRORS:

Printing which is directed to the system line printer is actually written to a disk unit, then "spooled" with other output from CPP or other software on the computer system. Because only one user may be creating such output at a time, CDS may issue the message:

(CDS)-- PRINTER IS BUSY

if a second user attempts to use the <LP> option.

WAIT

SYNTAX: W[AIT]
W[AIT] <NOPROMPT>

DESCRIPTION: Causes CDS to pause until the user types a "response" and presses the RETURN key.

If TIMEOUT is set, CDS will abort the WAIT after "timeout" minutes.

If LONG is set, CDS will accept that number of characters, and will then force a RETURN. If RETURN is pressed before the requisite number of characters have been typed, CDS will continue as it would for any answer.

LIMITATIONS:

SYSTEM VARIABLE EFFECTS:

EXAMPLES:

```
WAIT
WAIT <NOPROMPT>
```

ERRORS:

(If the system variable FINISH is set to the name of a frame, and the student pressed CTRL-Z or the ESC key, CDS will pass control to the finish frame before terminating the lesson.)

4.5 Prototype Level Functions

Section 5.2 of this document outlines those CCDL functions which are available on the prototype level (author interaction) of CDS. They are briefly described here as well.

CCDL functions to be executed during author interaction are prefaced by a dot "." in the prototype frames, and include:

```
.ANSWER
.CALCULATE (including KEYCODE processing)
.DATASET
.IF
.JUMP      (treated specially, see note in section 5.2)
.SHOW
.WAIT
```

5.0 Programmer use of CDS

5.1 Editing

The user wishing to edit a courseware or sign-on file requests access by typing the filename followed by ESC rather than CR, as described in section 3.1 or 3.2 .

Once access to a courseware or sign-on file is granted by CDS, the user is at DIRECTORY LEVEL. The options available at this level are listed and described below.

DIRECTORY LEVEL --

@xxxxxxxx : (also available at frame level) The user may type the "@" followed by a frame name. If the frame does not yet exist, CDS creates it. If it exists, it is fetched from disk and made available for editing. The user is taken to FRAME LEVEL.

@*yyyyyyy : (also available at frame level) The user is taken to FRAME LEVEL to edit the prototype whose name was typed (prototypes begin with an asterisk). The creation of prototypes is described fully in 5.2, below.

*zzzzzzzz : (also available at frame level) A prototype name without preceding "@" begins EXECUTION of the prototype. The CCDL code thus produced goes either into the current frame (at FRAME LEVEL), or into a frame the prototype will specify.

F : This causes CDS to print the names of all frames in the courseware file, excluding prototypes.

P : This causes printing of prototype names.

D : CDS asks WHAT FRAME?, and then deletes that frame from the courseware or sign-on file.

PR : CDS calls CPP to make a full lesson printout on the system printer. Printouts are spooled and initiated whenever the printer is next available.

ZAP : CDS asks whether you really want to permanently delete the courseware or sign-on file, then if your answer is "Y", deletes it.

TR : CDS calls CTT to translate the lesson, after asking what ruleset is to be used.

INFO : Information such as title, author, change-code and inspect-code may be specified again using the INFO command.

RUN : CDS ends the editing session and RUNs the lesson.

When the user moves to FRAME LEVEL (using the "@"), a second set of options appears:

FRAME LEVEL --

I n	: INSERT after line "n". CDS prompts the user for lines of text to be placed after line "n" in the frame. The user ends INSERT mode by sending a null line (a carriage-return immediately after the prompt). INSERT may also be ABORTED by pressing the ESC key at any time -- all lines typed so far are discarded and the old frame is restored from disk.
D a,b	: DELETE lines "a" through "b".
D n	: Delete line "n".
R a,b	: REPLACE lines "a" through "b" with text to be typed. CDS prompts as it does for INSERT mode. Any number of lines may replace those deleted.
R n	: REPLACE a single line.
L	: LIST the entire frame.
L n	: List line "n".
L a,b	: LIST lines "a" through "b".
COPY	: CDS asks for a lesson name, frame name and the change-code or inspect-code on that lesson, then copies the requested frame into this one.
@xxxxxxxx, @*yyyyyyy, *zzzzzzz and RUN are also available at FRAME LEVEL and are described earlier in this section.	

EXAMPLE:

A user begins a new lesson:

```
WHAT LESSON DO YOU WANT?  : EXAMPLE 5
THERE IS NO LESSON BY THAT NAME,
DO YOU WISH TO CREATE ONE? YES
LESSON TITLE:FIFTH EXAMPLE -- SHOWING EDITING
AUTHOR OR SOURCE:JIM SCHUYLER
CHANGE-CODE:
■■■■■■■■■■■■■■■
INSPECT-CODE:
■■■■■■■■■■■■■■■
TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
DO YOU WANT TO ZERO THE STUDENT TIMING DATA?YES
WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
WHAT COURSE OR USER-GROUP IS S/HE IN?UDP
```

Then (next page) the user inserts frame "1". Note that the INSERT command is given with no line number specified, so CDS inserts after line Ø (or at the beginning of the frame). Following the command, CDS prompts for each line with three dots. The user signals the end of insertion by inserting a null line (pressing carriage-return immediately after the prompt).

```
DIRECTORY LEVEL-- WHAT NOW? @1
NEW FRAME
FRAME LEVEL-- [1] WHAT NOW ?INSERT
...SHOW THE DOCTOR HAS DECIDED THAT IPPB WITH BRONKOSOL (ISOETHARINE)
...SHOW AND NEOSYNEPHRINE IS INDICATED.  IN THIS PARTICULAR PATIENT
...SHOW WOULD WE EXPECT ANY UNTOWARD REACTION TO THERAPY?
...__cr
```

Now, a listing of the entire frame is to be made.

```
FRAME LEVEL-- [1] WHAT NOW ?LIST
************************
[1]  187  CHARACTERS
     1:SHOW THE DOCTOR HAS DECIDED THAT IPPB WITH BRONKOSOL (ISOETHARINE
     2:SHOW AND NEOSYNEPHRINE IS INDICATED.  IN THIS PARTICULAR PATIENT
     3:SHOW WOULD WE EXPECT ANY UNTOWARD REACTION TO THERAPY?
************************
```

The next part of this frame is now inserted. This time, the insertion is to follow line 3. (If the user had simply said INSERT, the inserted lines would go at the beginning of the frame, of course.)

```
FRAME LEVEL-- [1] WHAT NOW ?INSERT 3
...WAIT
...ANSWER <OK><SPELL>MAYBE<OR>POSSIBL<OR>PROBABL
...SHOW  THAT'S RIGHT.  WITH BOTH PARENTS HYPERTENSIVE, WE MIGHT
...SHOW  ANTICIPATE PROBLEMS.
...__cr
FRAME LEVEL-- [1] WHAT NOW ?LIST
************************
[1]  327  CHARACTERS
     1:SHOW THE DOCTOR HAS DECIDED THAT IPPB WITH BRONKOSOL (ISOETHARINE
     2:SHOW AND NEOSYNEPHRINE IS INDICATED.  IN THIS PARTICULAR PATIENT
     3:SHOW WOULD WE EXPECT ANY UNTOWARD REACTION TO THERAPY?
     4:WAIT
     5:ANSWER <OK><SPELL>MAYBE<OR>POSSIBL<OR>PROBABL
     6:SHOW  THAT'S RIGHT.  WITH BOTH PARENTS HYPERTENSIVE, WE MIGHT
     7:SHOW  ANTICIPATE PROBLEMS.
************************
```

And a minor change is made to line 5, using the REPLACE command.

```
FRAME LEVEL-- [1] WHAT NOW ?REPLACE 5
...ANSWER <OK><SPELL>YES<OR>MAYBE<OR>POSSIBL<OR>PROBABL
...__cr
FRAME LEVEL-- [1] WHAT NOW ?LIST
************************
[1]  334  CHARACTERS
     1:SHOW THE DOCTOR HAS DECIDED THAT IPPB WITH BRONKOSOL (ISOETHARIN
     2:SHOW AND NEOSYNEPHRINE IS INDICATED.  IN THIS PARTICULAR PATIENT
     3:SHOW WOULD WE EXPECT ANY UNTOWARD REACTION TO THERAPY?
     4:WAIT
     5:ANSWER <OK><SPELL>YES<OR>MAYBE<OR>POSSIBL<OR>PROBABL
     6:SHOW  THAT'S RIGHT.  WITH BOTH PARENTS HYPERTENSIVE, WE MIGHT
     7:SHOW  ANTICIPATE PROBLEMS.
************************
```

Returning to DIRECTORY LEVEL (CR does it), the user lists the frame directory, which only contains one frame at this point.

```
FRAME LEVEL-- [1] WHAT NOW ?↵
DIRECTORY LEVEL-- WHAT NOW? F
**********************
FRAMES ---
1
**********************
```

The user decides to copy a frame from a previous lesson which would be useful in the new lesson. After copying, he requests that the first 10 lines be listed to verify that it is the one he wanted.

```
DIRECTORY LEVEL-- WHAT NOW? 02
NEW FRAME
FRAME LEVEL-- [2] WHAT NOW ?COPY
FROM WHAT LESSON?  CHECK1
WHAT FRAME DO YOU WANT TO COPY?  18
INSPECT(OR CHANGE)-CODE
**************
FRAME LEVEL-- [2] WHAT NOW ?LIST 1,10
    1:QUESTION:
    2:SHOW <ERASE>
    3:SHOW WOULD YOU SUSPECT THAT THIS PATIENT IS HYPERINFLATED?
    4:WAIT
    5:CAL   V27=V27+1
    6:ANSWER <OK><SPELL>YES<OR>WOULD
    7:CAL   V18=V18-1
    8:CAL   V28=V28+1
    9:SHOW WHAT INFORMATION SUGGESTS THAT?
   10:SHOW <END DISPLAY>
FRAME LEVEL-- [2] WHAT NOW ?↵
```

As a final step, the user translates the lesson to TUTOR-IV. The file created by the translator will be called EXAMPLE5.TU4, and could be transferred to tape or edited directly (but not RUN) by CDS.

```
DIRECTORY LEVEL-- WHAT NOW? TR
RULESET? : CDS-TU4
WHAT LESSON DO YOU WANT?  :
```

5.2 Constructing Prototypes

Prototype frames may be constructed within any lesson. The name of a prototype must begin with an asterisk (*). To begin a new prototype named *XYZ, for example, one would type:

@*XYZ

at either DIRECTORY LEVEL or FRAME LEVEL.

I, D, R, L and COPY work as usual on prototypes.

A prototype has as its purpose the creation of student-level CCDL frames. Thus, prototypes contain instructions for interaction on two levels:

a) instructions on what's to be created in the final CCDL frame, and

b) instructions for carrying out the interaction with an author or content expert.

Any CCDL function which appears in a prototype will be carried through to the final frame. Substitution markers of various sorts make it possible to tailor these lines:

: regular substitution marker. CDS pauses and waits for the author to type text to replace the marker.

#...# : "indefinite" substitution. CDS pauses as for a regular marker. If text is typed, CDS makes the substitution, then repeats the line asking for another substitution. If the user's response is null (CR), however, CDS continues to the next line in the prototype.

#expr# : CDS substitutes the value of the expression, in ASCII string form, for the markers. There is no pause for input.

#frame+n# : CDS substitutes the value of the current frame name (binary) plus the value of "n".

EXAMPLES:

The first example shows use of the substitution marker:

creating the prototype

```
DIRECTORY LEVEL-- WHAT NOW? @*XYZ
NEW FRAME
FRAME LEVEL-- [*XYZ] WHAT NOW ?INSERT
...SHOW <ERASE>
...SHOW #
...↵
```

using the prototype

```
FRAME LEVEL-- [*XYZ] WHAT NOW ?@1
NEW FRAME
FRAME LEVEL-- [1] WHAT NOW ?*XYZ
SHOW #THIS IS A SAMPLE FRAME.
```

the result

```
FRAME LEVEL-- [1] WHAT NOW ?LIST
**********************
[1]    42  CHARACTERS
     1:SHOW <ERASE>
     2:SHOW THIS IS A SAMPLE FRAME.
**********************
FRAME LEVEL-- [1] WHAT NOW ?
```

In the second example, the only difference is that "indefinite" substitution has been used. Thus, the user may fill in as many lines as he likes for the text of the frame being created:

```
DIRECTORY LEVEL-- WHAT NOW? @*XYZ
NEW FRAME
FRAME LEVEL-- [*XYZ] WHAT NOW ?INSERT
```

the prototype

```
...SHOW <ERASE>
...SHOW #...#
...↵
```

```
FRAME LEVEL-- [*XYZ] WHAT NOW ?@1
NEW FRAME
```

the interaction

```
FRAME LEVEL-- [1] WHAT NOW ?*XYZ
SHOW #THIS IS A SAMPLE FRAME,
SHOW #USING "INDEFINITE" SUBSTITUTION.
SHOW #SUBSTITUTION CONTINUES UNTIL A NULL LINE IS TYPED.
SHOW #↵
FRAME LEVEL-- [1] WHAT NOW ?LIST
```

the result

```
**********************
[1]   136  CHARACTERS
     1:SHOW <ERASE>
     2:SHOW THIS IS A SAMPLE FRAME,
     3:SHOW USING "INDEFINITE" SUBSTITUTION.
     4:SHOW SUBSTITUTION CONTINUES UNTIL A NULL LINE IS TYPED.
**********************
FRAME LEVEL-- [1] WHAT NOW ?
```

In the final example, a more complex prototype is created and used to create two final CCDL frames. Because "frame" is a valid expression (an ASCII string), its value is used wherever #frame# appears. The same construction is also used in the CAL, to assign a value to V1: in frame 2, V2 is used.

```
                    DIRECTORY LEVEL-- WHAT NOW? @*XYZ
                    NEW FRAME
                    FRAME LEVEL-- [*XYZ] WHAT NOW ?INSERT
the                 ...SHOW <ERAS>
prototype           ...SHOW ITEM #FRAME#:
                    ...SHOW #...#
                    ...CAL  V#FRAME#=V#FRAME#+1
                    ...cr//
                    FRAME LEVEL-- [*XYZ] WHAT NOW ?@1
                    NEW FRAME
the                 FRAME LEVEL-- [1] WHAT NOW ?*XYZ
interaction         SHOW #THIS IS A SAMPLE ITEM.
                    SHOW # cr//
                    FRAME LEVEL-- [1] WHAT NOW ?LIST
                    ************************
the                 [1]    66  CHARACTERS
result                 1:SHOW <ERAS>
                       2:SHOW ITEM 1:
                       3:SHOW THIS IS A SAMPLE ITEM.
                       4:CAL  V1=V1+1
                    ************************
(go to frame 2)     FRAME LEVEL-- [1] WHAT NOW ?@2
                    NEW FRAME
another             FRAME LEVEL-- [2] WHAT NOW ?*XYZ
interaction         SHOW #THIS IS ANOTHER FRAME.
                    SHOW # cr//
                    FRAME LEVEL-- [2] WHAT NOW ?LIST
                    ************************
another             [2]    66  CHARACTERS
result                 1:SHOW <ERAS>
                       2:SHOW ITEM 2:
                       3:SHOW THIS IS ANOTHER FRAME.
                       4:CAL  V2=V2+1
                    ************************
                    FRAME LEVEL-- [2] WHAT NOW ?
```

To control the interaction with the AUTHOR, the prototype may contain CCDL functions prefaced by a dot (.) . These functions are processed as the prototype is being scanned, and are not passed through for inclusion in the final student-level frame. All of the standard CCDL functions of chapter 4 are available.

The .JUMP function, however, is interpreted slightly differently on the prototype level:

.JUMP xxxxxxxx : Proto-jumps with frame names as their targets cause the student-level CCDL frame now being produced to be completed and a new frame by the name given is begun.

EXAMPLE:
.JUMP pdq
Causes the current prototype to begin working on frame "PDQ".

.JUMP *yyyyyyy : Proto-jumps with prototypes as their targets cause the named prototype to be invoked. The CCDL frame being created is not affected.

EXAMPLE:
.JUMP *drill
Causes the *drill prototype to begin.

The combination of these two jumps at prototype level makes it possible to link large numbers of prototypes in a general way, to create complex strategies.

Here is a more complex example of a prototype, with the interaction it might create, and the CCDL result:

example:

```
DIRECTORY LEVEL-- WHAT NOW? @.XYZ
NEW FRAME
FRAME LEVEL-- [.XYZ] WHAT NOW ?INSERT
....SHOW THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
....SHOW TYPE THE TEXT FOR THE QUESTION:
...SHOW  #...#
...WAIT
....SHOW WHAT IS THE CORRECT ANSWER TO THE QUESTION? <NOCR>
....WAIT <NOPROMPT>
...ANSWER <SPELL> #RESPONSE#
...SHOW  CORRECT.
...tv
```

To illustrate the difference between CCDL and the prototype level, we have listed the *XYZ prototype below, outlining the portions which will be executed at the AUTHOR-level:

```
FRAME LEVEL-- [*XYZ] WHAT NOW ?LIST
**********************
[*XYZ]   223  CHARACTERS
     1:.SHOW THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
     2:.SHOW TYPE THE TEXT FOR THE QUESTION:
     3:SHOW  #...#
     4:WAIT
     5:.SHOW WHAT IS THE CORRECT ANSWER TO THE QUESTION? <NOCR>
     6:.WAIT <NOPROMPT>
     7:ANSWER <SPELL> #RESPONSE#
     8:SHOW  CORRECT.
**********************
```

Note that each function to be performed at AUTHOR level is prefaced by the dot (.) in the prototype. Those lines not outlined above will be passed through to the final CCDL frame. There are substitution markers in two of them, so CDS will, of course, require some material from the author or programmer at those points.

Below you see an interaction in which the author creates frame "1":

```
FRAME LEVEL-- [*XYZ] WHAT NOW ?@1
NEW FRAME
FRAME LEVEL-- [1] WHAT NOW ?*XYZ
THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
TYPE THE TEXT FOR THE QUESTION:
SHOW  #CESSATION OF BREATHING IS CALLED (---).
SHOW  #@//
WHAT IS THE CORRECT ANSWER TO THE QUESTION? APNEA
FRAME LEVEL-- [1] WHAT NOW ?LIST
**********************
[1]    87  CHARACTERS
     1:SHOW  CESSATION OF BREATHING IS CALLED (---).
     2:WAIT
     3:ANSWER <SPELL> APNEA
     4:SHOW  CORRECT.
**********************
FRAME LEVEL-- [1] WHAT NOW ?
```

It's only a minor stylistic point, but notice that the programmer who wrote this prototype used the NOCR and NOPROMPT options on the SHOW and WAIT functions to save a little space when asking "What is the correct answer...".

As a final touch, the programmer improves the *XYZ prototype so that it shows the item number, and so that it asks whether the author wants to do another frame. Again, the AUTHOR level portions have been outlined in the listing below for emphasis:

```
FRAME LEVEL-- [*XYZ] WHAT NOW ?LIST
**********************
[*XYZ]   389  CHARACTERS
     1:.SHOW ITEM #FRAME#:
     2:.SHOW THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
     3:.SHOW TYPE THE TEXT FOR THE QUESTION:
     4:SHOW  #...#
     5:WAIT
     6:.SHOW WHAT IS THE CORRECT ANSWER TO THE QUESTION? <NOCR>
     7:.WAIT <NOPROMPT>
     8:ANSWER <SPELL> #RESPONSE#
     9:SHOW  CORRECT.
    10:.SHOW DO YOU WANT TO DO ANOTHER ITEM? <NOCR>
    11:.WAIT <NOPROMPT>
    12:.ANSWER <SPELL>YES
    13:.JUMP #FRAME+1#
    14:.JUMP *XYZ
    15:.ANSWER <UN>
    16:.SHOW THAT'S ALL THEN...
**********************
```

And now the programmer uses the refurbished prototype to create two courseware frames.

```
DIRECTORY LEVEL-- WHAT NOW? @1
FRAME LEVEL-- [1] WHAT NOW ?*XYZ
ITEM 1:
THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
TYPE THE TEXT FOR THE QUESTION:
SHOW  #NORMAL BREATHING, OF WHICH WE ARE USUALLY NOT AWARE,
SHOW  #IS CALLED (---).
SHOW  #↵
WHAT IS THE CORRECT ANSWER TO THE QUESTION? EUPNEA
DO YOU WANT TO DO ANOTHER ITEM? YES
ITEM 2:
THIS PROTOTYPE HELPS YOU WRITE A SIMPLE FRAME:
TYPE THE TEXT FOR THE QUESTION:
SHOW  #CESSATION OF BREATHING IS CALLED (---).
SHOW  #↵
WHAT IS THE CORRECT ANSWER TO THE QUESTION? APNEA
DO YOU WANT TO DO ANOTHER ITEM? NO
THAT'S ALL THEN...
FRAME LEVEL-- [2] WHAT NOW ?
DIRECTORY LEVEL-- WHAT NOW? F
**********************
FRAMES ---
1
2
**********************
DIRECTORY LEVEL-- WHAT NOW?
```

This chapter has really contained only a technical description of the components of CDS prototypes. The complex interactions which may result from some of the more sophisticated prototypes still must be imagined by the reader. The interested programmer may wish to consult some of the prototype frames supplied with the CDS release tape. These appear in a "lesson" named PROTO. These prototype frames manipulate the user-variables V1 through V100, plus a number of datasets, during creation of drills and learner-control modules.

In addition to the *SIGNON and *DRILL prototypes, PROTO contains a *LDL prototype which is used to create learner-control modules of the sort found in the TICCIT system. *LDL has been used on an experimental basis to create two learner-control modules, and is still undergoing significant development -- however, feedback from potential authors who might use it would be valuable.

Much more experience with a range of courseware authors must be gained before a true evaluation of the power and flexibility of these prototypes can be made.

5.3 Internal variables for prototypes *DRILL and *LDL

The CAI Design System is released with several sets of "standard" prototypes for author-level interactions. These make extensive use of the user-variables v1 through v100, and employ datasets for more permanent data storage.

*DRILL and *LDL use the following variables at the prototype level:

vars.	prototypes using them	description
v1 to v25	*data	Used to hold objective numbers corresponding to items 1 through 25 in a drill.
v50	*drill	Number of items entered during this session. If zero, *drpick shows a "long" description of options.
v51	*drill	Number of times "REPEAT or JUMP" message has been shown by *drtut. If greater than 5, an abbreviated form of the message is shown.
v52	*drill & *ldl	Non-zero if prototypes assume only one form of each drill item. Reset from AUTHORS dataset.
v53	*dritems	Sub-item number.
v54	*dritems	Item number.
v55	*data	Non-zero is a JUMP to the next data frame is to be assembled. (It is zero initially.)
v56	*data	Number of items in the current objective. (*data loops through all objectives of an instructional module which are related to items in the CBE module.)
v57	*data	Current instructional objective number.
v58	*data	Maximum instructional objective number.
v59		Temporary storage (any prototype may use it).
a60 to 64	*drill & *ldl	Contains "<EXACT>" or "<SPELL>", depending upon author's choice. (Reset from AUTHORS dataset.)
v65	*dritems	Non-zero if screen is not to be erased between sub-items in a series.

vars.	prototypes using them	description
v86		Unassigned.
v87	*ldl	Number of PRACTICE items necessary for mastery of current objective. (Also used at student level.)
v88	*ldl	Number of items on current menu. (Also at student level.)
v89	*ldl	Current item number (on menu) or objective number. (Also at student level.)
v90		Unused.
v91	*ldl	Non-zero if RULE for this objective has been seen. Zero if not. (Student level only.)
v92	*ldl	Number of practice items for this objective. (Also at student level.)
v93	*ldl	Number of examples for the objective. (Also at student level.)
v94	*ldl	Current PRACTICE number. (Author level only.)
v95	*ldl	Current EXAMPLE number. (Author level only.)
v96		Unused.
v97	*ldl	Pointer to current rule in dataset (Author level only.)
v98	*ldl	Current RULE number (corresponds to objective number). (Also at student level.)
v99		Unused.
v100	*ldl	Non-zero in NARRATIVE or SCRIPT mode. Zero in LEARNER CONTROL mode. (Student level.)

DATASETS ---

"AUTHORS":

This dataset is accessed by both *drill and *ldl series prototypes. It contains 100-word records for each author on the CDS system who has used either of these prototype sets. Very few of the variables in these 100 have been allocated; however, they will be used to maintain information on each author's preferences so that questions need not be asked at the start of each and every interaction.

words 1 to 4	The User's siqnon NAME.
words 5 to 9	The user's User-Group (UG).
word 10	The number of *drill or *ldl sessions begun by this computer user.
word 52	Non-zero if this author prefers to write only one form of each item in a drill. (Corresponds to v52 in *ldl and *drill prototypes.)
words 60 to 64	Contains " EXACT " or " SPELL ". (Corresponds to a60 through 64 as used in *ldl and *drill.)

It is anticipated that data to be stored "permanently" in AUTHORS dataset will be stored in corresponding locations; i.e. v78 would go in word 78, v90 in word 90, etc.

"Xlesson":

As the *ldl (learner design language/TICCIT) prototype creates an instructional module, it creates a separate dataset in which to store structural information. The dataset name is "X" followed by the lesson name.

word 1	Number of rules in the module.
2	Number of rules completely specified (author level).
3	Non-zero if menu is completed.

The rest of the dataset is composed of 25-word records. The word numbers specified below are relative to the beginning of each record. There is one 25-word record for each RULE (objective) in the module.

word 1	Unused
2	Unused
3	Total EXAMPLE/EXPLANATIONs.
4	Completed EXAMPLE/EXPLANATIONs.
5	Total PRACTICE items.
6	Completed PRACTICE items.
7	
8	
9	Number of PRACTICE items to achieve mastery.
10-25	Unused.

6.0 Using the translator

The CDS system contains a simple, rule-driven translator, CTT. This program makes possible the translation of courseware from one CAI language to another. Because of the differences among the most popular CAI languages, it is not possible to ensure complete translation in a "transparent" way among all languages. However, if a courseware author uses some restraint in his or her programming style, OR uses CDS's library of prototypes to create all courseware, translatability CAN be ensured. Because CDS is primarily an authoring system based on these prototype interactions, it is to be expected that the bulk of courseware on CDS will be fully translatable, at least to TUTOR, Coursewriter-III, PILOT and probably BASIC and MUMPS as well.

"Imported" courseware from many systems may similarly be translated, using appropriate sets of translation rules, for use on CDS. PILOT, Coursewriter-III and MULTITUTOR courseware (limited to non-graphics) should be ideal. As graphics are implemented in CCDL, TUTOR courseware which utilizes PLATO's more advanced features should also become translatable.

Finally, it is possible for courseware authors to write courseware in other languages on the CDS system, and have it "automatically" translated for immediate testing.

This section of the CDS documentation covers the use of, and the creation of rulesets for CTT, the translator.

6.1 Translation using existing rulesets

6.1.1 Whole-lesson translation

Translation, like all other manipulations of the courseware file, requires that either the change-code or inspect-code be given correctly by the potential user. One accesses the courseware file as if beginning a regular editing session (lesson-name terminated by ESC key). At DIRECTORY LEVEL, one types "TR" to select translation. CDS then requests the name of the ruleset to be used by the translator.

Ruleset names: The name has two parts, separated by a dash. The first part corresponds to the "source" (original) language or system, and will be "CDS" in most cases. The second part corresponds to the "target" language or system. These correspond to the file "extensions" which are appended to filenames by CDS, and currently we use the following:

"CDS" for CDS or CCDL files,
"TU4" for TUTOR-IV , PLATO-IV's language,
"CW3" for Coursewriter-III, IBM's language,
"HPB" for Hewlett-Packard 2000/access BASIC,
"SMP" for "simplified" format printouts.

As an example, to translate from CCDL to Coursewriter-III, the ruleset "CDS-CW3" would be used. To go from CDS to TUTOR, "CDS-TU4" is used.

Translations requested by the "TR" option at DIRECTORY LEVEL are carried out as soon as CTT can be activated. In an idle system (with only CDS running), this generally requires 5 to 30 seconds depending upon lesson length. The result of translation may be immediately examined, printed, or even translated again using another ruleset.

6.1.2 Examining translated courseware

Each courseware file created by the CDS system, or by the translator, CTT, has a three-character file type or "extension" at the end of its name. All CCDL courseware has the default extension "CDS" if none is explicitly given by the author. For instance, lessons GAS and INDEX are stored in files GAS.CDS and INDEX.CDS on device cw0: .

At the WHAT LESSON DO YOU WANT message, the extension "CDS" is added automatically, unless the author specifies another extension. Any extension of three or fewer characters is valid, but only the extensions CDS, TU4, CW3, SMP and HPB are in frequent use at UOP.

The ruleset used for full lesson translation determines the file extensions used by the translator. Translation of the lesson GAS, for instance using ruleset CDS-TU4, would take input from the file GAS.CDS, and produce translated output on the file GAS.TU4 .

One examines the translated output by specifying full filename, including extension:

```
        WHAT LESSON DO YOU WANT? : gas.tu4
```

(terminated by ESC, of course, to enter editing mode).

A full example appears on the next page.

EXAMPLE: (next page)

1) In the example, at this point, the user has given the name of a lesson to be created (terminated by ESC). CDS requests lesson information and ends up at DIRECTORY LEVEL.

2) At FRAME LEVEL, in frame "1", the user creates a 6-line frame.

3) Returning to DIRECTORY LEVEL, he asks for full lesson translation, and specifies the standard ruleset for CDS to TUTOR-IV translation.

4) On the next page, the user has accessed "example.tu4", the tutor-IV lesson created by the translator. Note that the file extension ".tu4" was determined by the ruleset name. Listing the contents of frame "1", he sees that it has been correctly translated to TUTOR. Of course, TUTOR cannot be used on the CDS system, but this lesson may be written to 9-track magnetic tape for delivery to a PLATO or MULTITUTOR system.

5) Just to illustrate the fact that "example" and "example.tu4" are two separate files, the user is here using "ZAP" to get rid of the TUTOR version, then must go into "example" to get rid of the CDS version.

EXAMPLE of lesson translation from CDS to TUTOR-IV:

```
        HELLO
        HELLO--WHAT IS YOUR NAME?
        >SKY
        WHAT IS YOUR USER-GROUP? UoP
        ACCESS-CODE:
        ■■■■■■■■■■■■■■■■
        GOOD AFTERNOON! 2:19 P.M.
        TUESDAY, 16 AUGUST, 1977.

        NO NEW MAIL.
        YOU HAVE 7 MESSAGES TO REVIEW.
1.  ->  WHAT LESSON DO YOU WANT?  : EXAMPLE
        THERE IS NO LESSON BY THAT NAME,
        DO YOU WISH TO CREATE ONE? YES
        LESSON TITLE:CDS TO TUTOR-IV TRANSLATION EXAMPLE
        AUTHOR OR SOURCE:JIM SCHUYLER
        CHANGE-CODE:
        ■■■■■■■■■■■■■■■■
        INSPECT-CODE:
        ■■■■■■■■■■■■■■■■
        TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
        DO YOU WANT TO ZERO THE STUDENT TIMING DATA?YES
        WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
        WHAT COURSE OR USER-GROUP IS S/HE IN?UoP
        DIRECTORY LEVEL-- WHAT NOW? @1
        FRAME LEVEL-- [1] WHAT NOW ?INSERT
        ...SHOW WHAT TYPE OF COMPUTER IS THIS RUNNING ON?
2.      ...WAIT
        ...ANSWER <OK>PDP-11
        ...SHOW CORRECT.
        ...ANSWER <NO>IBM
        ...SHOW MOST DEFINITELY NOT.
        ...
        FRAME LEVEL-- [1] WHAT NOW ?
3.  ->  DIRECTORY LEVEL-- WHAT NOW? TR
        RULESET? : CDS-TU4
        WHAT LESSON DO YOU WANT?  : EXAMPLE.TU4
```

```
WHAT LESSON DO YOU WANT?   : EXAMPLE.TU4
CHANGE-CODE?:
■■■■■■■■■■■■■■■■
OK
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
CDS TO TUTOR-IV TRANSLATION EXAMPLE
JIM SCHUYLER
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
AV. COMPLETION     0  MINUTES (OVER     0 RUNS. STUDENTS ONLY)
EDITING       =  0   HR  0   MIN
14:24 16-AUG-77
AUTHOR:SKY-UOP
CREATED:16-AUG-77
LAST EDITED:
LAST USED:
AV TIME TO ESC     0  MINUTES (OVER     0 RUNS. STUDENTS ONLY)
TOTAL USER TIME=   0   HR  0   MIN (ALL USER TYPES)
DIRECTORY LEVEL-- WHAT NOW? @1
FRAME LEVEL-- [1] WHAT NOW ?LIST
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
[1]  136  CHARACTERS
   1:WRITE    WHAT TYPE OF COMPUTER IS THIS RUNNING ON?
   2:ARROW    2105
   3:ANSWER   PDP-11
   4:WRITE    CORRECT.
   5:WRONG    IBM
   6:WRITE    MOST DEFINITELY NOT.
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
FRAME LEVEL-- [1] WHAT NOW ?_
DIRECTORY LEVEL-- WHAT NOW? ZAP
DO YOU WISH TO PERMANENTLY DELETE THIS LESSON?YES
WHAT LESSON DO YOU WANT?   : EXAMPLE
CHANGE-CODE?:
■■■■■■■■■■■■■■■■
OK
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
CDS TO TUTOR-IV TRA
JIM SCHUYLER
◆◆◆◆◆◆◆◆
AV

DIRECTORY LEVEL-- WHAT NOW? ZAP
DO YOU WISH TO PERMANENTLY DELETE THIS LESSON?YES
WHAT LESSON DO YOU WANT?   : ^Z


GOODBYE
```

6.1.3 Auto-translation -- Authoring in other Languages

In sections 6.1.1 and 6.1.2, it was assumed that an author or programmer had courseware in CCDL to be translated to another language. It is also possible to create courseware on the Design System in other languages than CCDL.

One specifies such a courseware file at creation time by adding a file extension corresponding to the language to be used. The extension "TU4" is the only one supported on the current CDS release.

During the file creation process, CDS asks whether auto-translation is to be performed. If this is the case, CDS uses a standard ruleset from lesson RULES to translate the TU4 file into a CDS file for execution. The ruleset name used by CDS corresponds to the languages involved; it will be TU4-CDS for a TUTOR to CCDL translation.

When auto-translation has been specified for a courseware file, CTT is called on a frame-by-frame basis, whenever any change is made to a courseware file. The translation is generally instantaneous and is not noticed by the author.

```
WHAT LESSON DO YOU WANT?  : EXAMPLE 7.TU4 esc
THERE IS NO LESSON BY THAT NAME,
DO YOU WISH TO CREATE ONE? Y
LESSON TITLE:TUTOR-IV EXAMPLE
AUTHOR OR SOURCE:JIM SCHUYLER
CHANGE-CODE:
■■■■■■■■■■■■■■■■
INSPECT-CODE:
■■■■■■■■■■■■■■■■
TYPE 'C' TO CREATE COURSEWARE FILE, 'S' FOR SIGNON RECORDS:C
DO YOU WANT TO ZERO THE STUDENT TIMING DATA?Y
WHAT SIGNON NAME SHOULD RECEIVE FEEDBACK?SKY
WHAT COURSE OR USER-GROUP IS S/HE IN?UOP
DO YOU WISH AUTOMATIC TRANSLATION?: YES
DIRECTORY LEVEL-- WHAT NOW? @1
```

In the EXAMPLE, which begins above, note that the user has put an "extension" on the filename (.TU4 in this case) indicating the language to be used. Consequently, CDS asks DO YOU WISH AUTOMATIC TRANSLATION?, and he answers YES.

Because the prototype library contains only CCDL prototypes (though there is no theoretical restriction on its content), prototypes will not be used.

At the top of the next page, the user begins editing frame "1" (or unit 1, as we should say for TUTOR). The user need not enter UNIT commands, as the translator will supply them when translating to TUTOR-IV for transport. Tag position as well is inimportant, as the translator will move all tags to column 8 for transport.

```
DIRECTORY LEVEL-- WHAT NOW? @1
NEW FRAME
FRAME LEVEL-- [1] WHAT NOW ?INSERT
...WRITE   THIS LESSON IS WRITTEN IN THE (---) LANGUAGE.
...ARROW   2105
...ANSWER  TUTOR
...WRITE   CORRECT. YOU'RE SMART!
...WRONG   CDS
...WRITE   PLATO WOULDN'T APPROVE. IT'S IN TUTOR.
...NEXTNOW 2
...NO
...WRITE   IT'S TUTOR-IV.
...NEXTNOW 2
...cr
FRAME LEVEL-- [1] WHAT NOW ?@2
NEW FRAME
FRAME LEVEL-- [2] WHAT NOW ?INSERT
...WRITE  THANKS FOR VISITING. SEE YOU LATER.
...cr
FRAME LEVEL-- [2] WHAT NOW ?
DIRECTORY LEVEL-- WHAT NOW? RUN
************************
TUTOR-IV EXAMPLE
JIM SCHUYLER
************************
THIS LESSON IS WRITTEN IN THE (---) LANGUAGE.
>TUTOR ?        OK
CORRECT. YOU'RE SMART!
THANKS FOR VISITING. SEE YOU LATER.
WHAT LESSON DO YOU WANT?  :
```

Just to show that this translation works and that courseware may be immediately run upon completion, the author has requested a RUN after finishing work on units 1 and 2.

Auto-translation is done on a frame-by-frame basis, so whenever the author makes a change to a particular frame, CTT will be called to do a one-frame translation.

6.2 Creating and Testing Rulesets

As a first step toward a useful translator for CAI languages, a set of pattern-recognizing procedures has been incorporated into the Design System. The program, CTT, reads a single set of translation rules, formed according to the syntax described in this section, and translates single frames or whole lessons, producing a second courseware file.

Each rule in a ruleset has two halves, a "pattern" to be recognized, and a "production" specifying what is to be put into the newly-created frame. These are separated by an equal sign:

pattern = production

PATTERN RECOGNITION:

The left side of a rule may contain any of the following:

" " : Quoted strings. Such strings must be present in the line from the courseware file for the rule to match. Capitalization is unimportant.

" " at the beginning of a rule :: The quoted string must be present at the beginning of the input line for the rule to match.

" " followed by #A# :: (See next page for explanation of #A# string-save marker) If the line begins with the quoted string, string-saving begins at the next position from the input.

EXAMPLE: Rule >>"show" #A# ...
Input >>show Hi there.
RESULT>>#A# contains " Hi there."

" " after #A# :: (See next page for explanation of #A# string-save marker) The input line is saved in the string-save area UNTIL the quoted string is found in the input line. (Quoted string is NOT saved.)

EXAMPLE: Rule >> #A# "<ERASE>" ...
Input >> show <ERASE>
RESULT>> #A# contains "show "

```
-        : Dash.  Skips over blanks in the input line.  There
         must be at least one blank for the rule to match.

         - followed by #A# :: (See below for #A# string-save
             marker). String saving begins with the first non-
             blank character which follows.

#A#      : String-save markers #A# thru #H#.  The input line is
         stored, character by character, in the string-save area.
         Each marker #A# thru #H# is capable of storing 64
         characters.

         followed by " " :: The input line is saved UNTIL the
             quoted string is encountered [if it is not en-
             countered, the pattern fails].  The pattern con-
             tinues at the end of the quoted string.
                 EXAMPLE: Rule  >> #C# "<ERASE>" ...
                          Input >> show <ERASE>
                          RESULT>> #C# contains "show "

                 EXAMPLE: Rule  >> #E# "<EXACT>" #F#
                          Input >> answer <EXACT> Sacramento
                          RESULT>> #E# contains "answer "
                                   #F# contains " Sacramento"

         followed by = :: The rest of the input line is saved.
                 EXAMPLE: Rule  >> #D# = ...
                          Input >> show Hi there.
                          RESULT>> #D# contains "show Hi there."

         followed by - :: The input line is saved, up to the
             next blank.  Blanks are then skipped.
                 EXAMPLE: Rule  >> #A# - ...
                          Input >> show Hi there.
                          RESULT>> #A# contains "show"
```

As translation of each new frame in a courseware file is begun, CTT puts a special line in the input buffer, containing only the ASCII character "@". It then invokes the ruleset once. Thus, at beginning-of-frame, the ruleset has the opportunity to output any special programming-language statements required in the target language. (For example, TUTOR-IV requires a "unit" command at the beginning of each unit or frame.)

EXAMPLE:

```
"@" = "unit     "#frame#
```

[Note that the quoted string must be found for the rule to match, and it will only be found at the beginning of a frame, producing a "unit" command as output.]

PRODUCING AN OUTPUT LINE:

The right side of a rule controls output. The right side is only processed when the entire left side has matched. It may contain:

" " : Quoted strings. These are moved to the output line character by character.

EXAMPLE:

```
Rule   >> "show"-"<ERASE>" = "erase"
Input >> show    <ERASE>
RESULT>> output line contains "erase"
```

#A# : String markers. (#A# thru #H# are available). The contents of the indicated marker are moved into the output line. Those contents may have been stored by the current rule, or by some previous rule.

EXAMPLE:

```
Rule   >> "show"-#H# = "write    "#H#
Input >> show Hi there.
RESULT>> output line contains "write    Hi there."
         #H# contains "Hi there."
```

#frame# : The current frame name is moved to the output line.

EXAMPLE:

```
Rule   >> "@" = "unit     "#frame#
Input >> beginning of frame 17
RESULT>> Output line contains "unit     17"
```

/ : The current output line is ended and a new output line is begun. Using this construction, a rule may create more than one output line from a single input line.

EXAMPLE:

```
Rule  >> "answer"-#A#"<OR>"#B# =
                    "answer "#A#/"answer "#B#^
Input >> answer yes<OR>true
RESULT>> output first contains "answer yes",
         which is written to output, then
         "answer true". Ruleset repeats
         due to "^" (see below).
```

^ : The current output line is moved to the input buffer and the ruleset is restarted.

EXAMPLE:

```
Rule  >> #A# "<EXACT>" #B# = #A##B#^
Input >> answer <EXACT> yes
RESULT>> output contains "answer  yes", then
         it is moved to input and the ruleset
         is restarted.
```

Testing Rulesets:

The standard rulesets which are used by CDS are kept in a file which is constructed as a standard courseware file. (Its name is RULES.) To create a new ruleset, one edits RULES, creating a frame with the name of the ruleset desired (see sections 3.2 and 5.1).

If full-lesson translation will be requested by the user, a ruleset may have any name. (that is, subject to the 8-character restraint on all frame names.)

For rulesets to be used in auto-translation, there is a prescribed format to be followed in creating the name:

original file's extension,
followed by a dash,
followed by new (target) file's extension.

EXAMPLE: For auto-translation from CCDL to TUTOR-IV, the ruleset name would be CDS-TU4. For the reverse translation, TU4-CDS would be appropriate.

The next few pages contain rulesets currently in use at University of the Pacific.

1) TU4-CDS: Used to translate limited TUTOR-IV to CCDL. This ruleset is used for experimental authoring done IN TUTOR-IV on CDS.
2) CDS-SMP: Used to provide "simplified" printouts of regular CDS courseware for peer-reviewers.
3) CDS-TU4: Used to translate CCDL courseware into TUTOR-IV for use on MULTITUTOR systems.

The experience gained over the past two years at UOP has clearly indicated that redesign of the translator would be required for advanced authoring support, even though the translator is producing perfectly adequate TUTOR-IV courseware at the moment. Presumably, that redesign effort would involve the translator's reading descriptions of the languages to be translated, and developing its own procedures for translation.

```
CWO:
RULES.CDS
*****************
TRANSLATION RULES

*****************
AV. COMPLETION     0  MINUTES (OVER     0 RUNS. STUDENTS ONLY)
AV TIME TO ESC     0  MINUTES (OVER     0 RUNS. STUDENTS ONLY)
EDITING       =  2  HR  28  MIN
14:41 26-OCTOBER-77
AUTHOR:SKY-UOP
CREATED:
LAST-EDIT:20-OCTOBER-77
LAST-USED:20-OCTOBER-77
TOTAL USER TIME=  2   HR 15   MIN (ALL USER TYPES)
*****************
***************** [TU4-CDS]=  349 CHARACTERS
    1:"@"=""


    2:"UNIT"=""
    3:"WRITE"-#A#="SHOW "#A#
    4:"ARROW"="WAIT"
    5:"CALC"-#A#="CAL "#A#
    6:"ANSWER"#A#","#B#="ANSWER"#A#"(OR)"#B#^
    7:"WRONG"#A#","#B#="WRONG"#A#"(OR)"#B#^
    8:"ANSWER"-#A#="ANSWER (OK)(SPELL)"#A#
    9:"WRONG"-#A#="ANSWER (NO)(SPELL)"#A#
   10:"NO"="ANSWER (UN)(NO)"
   11:"OK"="ANSWER (OK)(UN)"
   12:"JUMP"-#A#="JUMP "#A#
   13:"NEXT"-#B#="SHOW (END DISPLAY)"/"JUMP "#B#
   14:#B#="*"#B#
***************

***************** [CDS-SMP]=  515 CHARACTERS
    1:"@"=""
    2:"Q"=/"QUESTION:"
    3:#A#"(ER"=""
    4:#A#"(EN"=""
    5:"IF"-"[(Y"#A#"/2"="FIRST FORM OF QUESTION:"
    6:"2"="ALTERNATE FORM OF QUESTION:"
    7:"SHOW"-#A#="    "#A#
    8:#A#"(EXACT)"#B#=#A#"  "#B#^
    9:#A#"(SPELL)"#B#=#A#"  "#B#^
   10:"ANSWER"#A#"(UN)"="FEEDBACK FOR UNANTICIPATED ANSWER:"
   11:"ANSWER"-"(OK)"#B#="CORRECT ANSWER:"#B#
   12:"ANSWER"-"(NO)"#B#="WRONG ANSWER:"#B#
   13:"ANSWER"-#A#="POSSIBLE ANSWER:"#A#
   14:"WAIT"=""
   15:"CAL"=""
   16:"IF [Y26"=""
   17:"JUMP"-"/W"="    --- COMPUTER REPEATS THE QUESTION ---"
   18:"JUMP"-"/"=""
   19:"JUMP"-#C#= "    --- NEXT FRAME => "#C#
   20:#D#=""
*****************
```

```
**************** [CDS-TU4]=  734 CHARACTERS
    1:"SHOW"#A#"<ER"=""
    2:#A#"<END"#B#">"#C#=#A##C#^
    3:"SHOW"-#A#="WRITE    "#A#
    4:"WAIT"="ARROW    2001"
    5:#A#"<EXACT>"#B#=#A##B#^
    6:#A#"<SPELL>"#B#=#A##B#^
    7:#A#"<OR>"#B#=#A#","#B#^
    8:"ANS"#A#"<UN>"#B#="UN "#A##B#^
    9:"UN "#A#"<NO>"="NO"
   10:"UN "="OK"
   11:"ANS"#A#"<OK>"#B#="ANSWER   "#B#
   12:"ANS"#A#"<NO>"#B#="WRONG    "#B#
   13:"ANSWER"-#A#="ANSWER   "#A#
   14:"CAL"#A#"Y"#B#="CAL"#A#"N"#B#^
   15:"CAL"-#A#="CALC     "#A#
   16:"JUMP"-"/WAIT"="JUDGE    NO"
   17:"JUMP"-"/"#A#="BRANCH   "#A#
   18:"JUMP"-#A#="NEXTNOW "#A#
   19:"IF"#A#" EQ "#B#="IF"#A#"$EQ$"#B#^
   20:"IF"#A#" NE "#B#="IF"#A#"$NE$"#B#
   21:"IF"#A#" GT "#B#="IF"#A#"$GT$"#B#
   22:#A#"IS TRUE"#B#=#A#"$LT$0"#B#^
   23:"IF"#A#"Y"#B#="IF"#A#"N"#B#^
   24:"IF"-"["#A#"]"-"JUMP"-"/"#B#="BRANCH   "#A#","#B#",X"
   25:"IF"=""
   26:"@"="UNIT     "#FRAME#
   27:"Q"="ENDARROW"
   28:"  "#A#=#A#^
   29:#D#=#D#
*****************
```

7.1 System overview

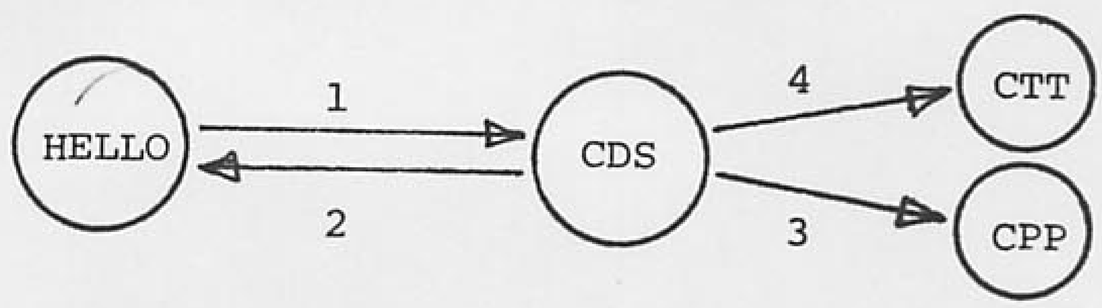
The CAI Design System is composed of four interrelated computer programs, HELLO, CDS, CPP and CTT. When run on the RSX11M operating system, CDS stays active at all times, and it requests activation of CPP or CTT when required to service the demands of CDS users. HELLO is activated by users on TT-type ports who wish to enter CDS.

CDS is installed and runs as a high priority task, to ensure adequate service to on-line users. CPP and CTT run at lower priorities.

CDS services a number of simultaneous users; the exact number is determined by the individual who assembles CDS initially.

7.2 Intermodule communication

The RSX11M operating system provides a facility for sending messages or data between operating programs (called tasks). Through this, HELLO tells CDS that a port is requesting sign-on, and CDS communicates with both CPP and CTT, to request printing or translation.



1) HELLO sends a messages to CDS giving the TT number when a new user is attempting a sign-on. CDS attaches that port and runs it from then on. (HELLO terminates)
2) CDS may respond to HELLO if ALL LINES ARE BUSY, by returning an appropriate message.
3) CDS activates CPP (the "printer") to print whole lessons, and sends a message giving the filename to be printed. These messages may stack up when several printouts are requested, but CPP will receive them as it is ready to handle them.
4) CDS activates CTT (the "translator") to translate lessons, to translate individual frames, and to control the writing of lessons onto magnetic tape for transfer to other computer systems.

Neither CPP nor CTT send messages back to CDS.

7.3 System security

CDS was programmed on RSX11M version 2. However, RSX11M version 3 provides password protection of User-Identification-Codes (UICs), which presumably allows the creator of a CDS installation to protect himself or herself against unwanted programmer access to courseware or datasets.

Assuming that the computer installation itself is not physically vulnerable to pilferage of data or courseware by those who might circumvent RSX11M's security measures, CDS also has some intermediate-strength controls of its own.

All CDS files are kept in a single (password-protected) UIC. Any user requesting access to CDS is given a user-name, user-group, and access-code by the CAI Coordinator. CDS requests each of these at sign-on and checks their validity. An incorrect access-code, or invalid sign-on name or user-group causes termination of the sign-on process. On dial-up lines, CDS hangs up the phone in these cases.

Once a user has successfully signed onto CDS, additional codes must be given to gain access to courseware files. These codes (CHANGE-CODE and INSPECT-CODE) are specified by the creator of a file, and are validated by CDS for each user requesting editing-access to the file. Improper codes cause denial of access by CDS.

In the case of printouts and translations, and writing of magnetic tapes, the CHANGE-CODE or INSPECT-CODE must be correctly given before CDS allows access to these facilities. Messages received by CPP and CTT are checked, and must have originated with CDS to be processed. This completes the cycle of protection.

7.4 Module linkage

This document includes full descriptions of modules which comprise HELLO, CDS, CPP and CTT. At the overall level, these four programs communicate via messages sent through RSX11M. Internally, however each program is self-sufficient. Using symbols defined in the CDS macro library (CDSMLB.MAC), they can each access CDS courseware files.

Within each program, there is a set of independent "processes". At inception of execution, one of these processes is started. (For CDS itself, a particular process is begun for each of the ports defined.) Control passes from one process to another. In CDS, subroutines are called via the JSR instruction, which leaves a return address on the stack. However, the stack must be clear when control switches from one terminal to another, and thus subroutines are short processes compared to the main processes which may request i/o.

Data transfer from one process to another is via global variables. Data transfer from a process to a subroutine it calls is generally via registers R1 through R5, though some global variables may also be involved.

7.5 Terminal interface

RSX11M provides driver programs to run two classes of ports which CDS deals with; these are TT-ports and XL-ports, driven by TTDRV and XLDRV respectively.

TT ports:

Input and output are handled on a line by line basis, through QIO$S macros. CTRL-C, CTRL-U, CTRL-Z, CTRL-O are handled by the system, not by CDS. A user at a TT port may talk directly with RSX11M, and to enter CDS must type "HELLO" (which starts the HELLO program running).

CDS can request single-character reading from TT ports when required.*

XL ports:

Version 2 of RSX11M provides only primitie support for XL ports. Output is handled on a full-line basis via QIO$S macros. Input is handled on a character-by-character basis (again via QIO$S), with echoing, CTRL-U and CTRL-Z all handled by CDS. All reads from XL ports are single-character. (At the user-level, however, both types of ports appear the same.)

We understand that version 3 of RSX11M supports XL ports in the same manner as TT ports, though this has not been verified by actual operation.

*Note: TT ports are processed on a full-line basis for most WAIT functions. However, when there is an entry in the KEYCODE table for a particular port, TT ports are processed on a character-by-character basis (just as an XL port would be) so that the KEYCODE interrupts may be properly generated when a specific key is pressed by the student. During this processing, CTRL-C is disabled (CDS uses the #IO.RAL code to read), and CTRL-U, CTRL-H (backspace), DEL, ESC and CTRL-Z are processed by CDS just as they would be for XL ports.

7.6 Internal course format

Courseware files are RSX11M direct-access ("random") files. (Thus must be transferred to tape in "image" mode.) Each file is composed of at least 2 records. A record consists of 2 disk blocks (1024 bytes total). The overall format is:

```
{record 1 --- courseware information
{record 2 --- frame directory (including prototypes)
{records 3 ... --- individual frames
```

Descriptions of the formats of individual records follow.

INFORMATION RECORD (record 1):

Symbols defined in CDSMLB.MAC, the CDS macro library, are used to access information in this record.

(symbol)	(byte value)	(description)
BUFTTL	4	Title of lesson is stored here+2, and continues to byte 99 which will be zero.
BUFAU	100.	Name of author or source is stored here +2.
BUFDAT	150.	last edit date (all dates are stored month,day,year; 3 bytes)
BUFCRE	153.	Creation date (see note above)
BUFLDU	156.	Last-date-used (see note above)
BUFAS	160.	Author's sign-on name
BUFDIR	170.	Directory record number
BUFNEX	172.	Next-available record number
BUFTIM	178.	Last-edit time (minutes)
BUFCC	180.	Change-code (10 bytes)
BUFIC	190.	Inspect-code (10 bytes)
BUFTIS	200.	Editing time (minutes)
BUFTIL	202.	Student time (completions)(minutes
BUFRUN	204.	Student runs (completions)
BUFSR	206.	Sign-on-records flag (nz=sign-on) (zr=courseware
BUFTUT	208.	Total user time (minutes)
BUFAUC	210.	Author's user group (10 bytes)
BUFTIE	220.	Student time (to escape)(minutes)
BUFESC	222.	Student runs (escapes)
BUFRUL	224.	Ruleset name (auto-transl flag)
(next available...	234.)	

FRAME DIRECTORY (record 2):

The directory is composed of five-word entries containing the names of frames and the disk record number for each.

Thus:
words 0-3: first frame name, in ASCII, upper-case
word 4 : disk record number for first frame
words 5-8: second frame name, in ASCII, upper-case
word 9 : disk record number for second frame
...

End of directory is indicated by a DISK RECORD NUMBER of zero.

When frames are deleted from a courseware file by an author, the FRAME NAME portion of the respective entry is zeroed, and the record number is kept. The entry is then moved to the end of the directory and saved. Later on, when a new frame is created (no matter what its new name), the new frame name will occupy the previously-zeroed frame name in the slot, and the old record number will be reused.

INDIVIDUAL FRAMES (records 3 thru end of file):

A frame is stored in ASCII. Each line of the frame is preceded by its line length (a single byte). A line length of zero (a zero-byte) indicates end of frame. For example:

```
byte 0 : 7
byte 1 : "S"
byte 2 : "H"
byte 3 : "O"
byte 4 : "W"
byte 5 : " "
byte 6 : "H"
byte 7 : "i"
byte 8 : 4
byte 9 : "W"
byte 10: "A"
byte 11: "I"
byte 12: "T"
byte 13: 0
```

Represents the two-line frame:

```
SHOW Hi
WAIT
```

7.7 Instruction format

CDS instructions are stored internally in exact ASCII representations of the lines entered by an author. Each line begins with a function specification (SHOW, WAIT, ANSWER, etc.) and may contain an optional "tag" portion giving additional information necessary to execution of the command. No compressed formats are used either internally or externally.

7.8 External courseware storage

Magnetic tape storage of CDS courseware is managed by the FLX (FILEX) file-exchange program. Courseware is stored in "image" format to preserve an exact bit-by-bit representation of the original.

Courseware to be transferred to other computer systems can also be put on magnetic tape by CTT. The format for these files is ASCII. Each physical record on the tape contains one logical record (a line) in ASCII, terminated by CR/LF pairs. All records are at least 14 bytes long on the tape. Physical tape marks (end of file marks) terminate each courseware file written to the tape, and end of tape is indicated by three tape marks in a row (alternately, by an "empty" file).

EXAMPLE:

```
{To move courseware from tape to disk:
        FLX cw0:/rs/im=mt0:xxx.cds/do/im
(moves courseware file "xxx" onto disk).

{To clear tape before writing courseware:
        FLX mt0:/ze

{To move courseware from disk to tape:
        FLX mt0:/do/im=cw0:xxx.cds/rs/im
(moves courseware file to tape).
```

*

7.9 Runtime memory usage

CDS may be operated in a mapped or unmapped RSX11M system. A single-user CDS will run in less than 100000 bytes (octal), including all i/o subroutines loaded from the RSX11M library. At UOP we are currently running a 3-user CDS in 76000_8 bytes, using a common FCSRES library.*

HELLO, CTT and CPP may be assigned to specific partitions, or to a system-controlled partition (SCP) in a mapped system.

CDS is composed of a main segment, which includes data areas for the users, and of 12 overlays. The overlays share a common load address. When i/o is to be performed for a process in an overlay, the overlay number and return address are saved in the appropriate user's area (at WAITER and OVERLAY), and the i/o is initiated. When i/o completes, of course, some other user may have loaded another overlay, and the main CDS loop must reload the appropriate overlay for the user whose i/o is now complete. This is described in the flow-charts for the LOOP processor.

In a single-user CDS system, HELLO, CDS, CPP and CTT could all share overlapping memory. With CTT and CPP running as low-priority tasks, RSX11M would swap them into CDS's area whenever CDS was waiting on i/o operations for the user's terminal.

SINGLE-USER SYSTEM:

In the single-user system, HELLO and CDS are assembled in such a way that when no CDS users are present, no tasks are running. When a user desires to run CDS, s/he types "HELLO". HELLO activates CDS, which runs the user. CTRL-Z causes CDS to exit, leaving the system idle again. The single-user CDS is swapped by RSX11M when other tasks need the partition (if this option was specified as sysgen time). A single-user system, with file-control-system (FCS) routines resident in CDS's partition, takes less than 100000_8 bytes of main memory (16k words).

MULTIPLE-USER SYSTEMS:

In multiple-user systems, CDS is resident and running at all times. When a user desires to run CDS, s/he types "HELLO". HELLO sends a message to CDS, which then attaches the appropriate TT terminal and runs. CTRL-Z causes CDS to detach the terminal. Multiple-user CDS is not swapped by RSX11M, because it may have input or output active on more than one terminal at a time (and RSX11M cannot swap such a program). In addition, CDS may not be "shuffled" (task SHF) if it is loaded into a system-controlled-partition (SCP), due to the multiple-input/output. Thus, it is always loaded when the SCP is "idle", to ensure it is at the bottom of the partition and will not prevent shuffling of other tasks. Because CDS cannot share its space with CPP or CTT in the multiple-user system, an additional partition is needed (or additional SCP space) for these tasks in multiple-user systems.

*Note: Each additional "port" we define on a multiple-user version of CDS adds about 7000_8 bytes to CDS's overall memory requirement.

7.10 Required resources

The RSX11M operating system, and the PDP-11 computer system are adequate to the tasks required by CDS, and for CAI in general. However, potential implementation of CDS or CDS-like software on other computers and operating systems should not be attempted without forethought. This section outlines the basic requirements a system must meet to allow CDS implementation.

7.10.1 Operating-system requirements

1) The system must allow a single program to access several terminals (ports) at once, and to have input and output going simultaneously on any or all of them. Completion of i/o may be signalled by the setting of a bit, a flag, or activation of an asynchronous routine (a "trap" in RSX11M parlance).

2) Single-character input must be supported.

3) "Transparent" input is desirable: the operating system should not screen any "special" characters on input.

4) A "break" character should be supported, so a user may terminate long outputs. The break condition should be transmitted to the CDS program for action. (RSX11M does NOT support this feature.)

5) "Transparent" output must be provided: no characters may be removed or altered on output.

6) Several programs (tasks) must be permitted to access the same courseware file at one time. (Only one of them needs "write" permission, and that will generally be CDS itself.)

7.10.2 Hardware requirements

1) CDS requires between 70000_8 and 100000_8 bytes (octal) of user memory. In a single-user version, this may be shared by RSX11M's swapping mechanisms with other programs. (This is 16k words of memory.)

2) CDS will run any ASCII terminal, printing or CRT. No special graphics are provided or required.

3) CDS utilizes RK05 disk packs for courseware and dataset storage. No determination has been made of the difficulty of using other devices.

4) CDS can utilize any number of RK05 courseware devices.