CSG IMS

CSG IMS
SUPPLEMENTARY DOCUMENTATION

Release A -

for Version 1.3

OS9/6809

Clearbrook Software Group

CSG IMS

SUPPLEMENTARY DOCUMENTATION

Release A -

for Version 1.3

OS9/6809

CLEARBROOK SOFTWARE GROUP

INFORMATION MANAGEMENT SYSTEM

IMS Supplementary Documentation

RELEASE A

IMS Supplementary Documentation
Table of Contents

TABLE OF CONTENTS

IMS Supplementary Documentation
Foreword

This document contains information about the latest version of CSG IMS which is not covered in the reference manual. If you received this IMS Supplementary Documentation with a software update, take note of any revisions, corrections or changes and mark them in your Reference Manual if you wish.

Version 1.3 is the latest revision of the CSG IMS package. Since our policy is to support only the latest version of our software, please have your system updated as soon as you receive notification of a new release.

IMS Supplementary Documentation
Changes to the CSG IMS Language

TABLE OF CONTENTS

The following pages contain information about new features and functions as well as changes to existing functions. These descriptions *replace* corresponding descriptions in release B of the CSG IMS documentation.

#COLUMNS

USAGE:

```
     #COLUMNS
```

#COLUMNS (read "number of columns") is a function which returns a number and may only be used in numeric expressions (see EXPRESSION).

PURPOSE AND OPERATION:
#COLUMNS returns the number of columns that the user's terminal can display. This information is recorded in the UTD driver file for the device associated with the user's terminal. This function is useful for specialized display formatting.

EXAMPLE:

```
     message="Welcome to the Sleepy Hollow"
     GOSUB center_message
     message="Mailing List System"
     GOSUB center_message
     END

     LABEL center_message
     PRINT PADCENTER$(message,#COLUMNS)
     RETURN
```

will output:

```
                    Welcome to the Sleepy Hollow
                        Mailing List System
```

#ROWS

USAGE:
```
     #ROWS
```

#ROWS (read "number of rows") is a function which returns a number and may only be used in numeric expressions (see EXPRESSION).

PURPOSE AND OPERATION:
#ROWS returns the number of lines (or rows) that the user's terminal can display. This information is recorded in the UTD driver file for the device associated with the user's terminal. This function is useful for specialized display formatting.

EXAMPLE:
```
     LOCATE #ROWS,1
     CLEAR LINE
     PRINT "Enter your choice: ";
     choice=LIBRARY$(GETKEY)
     PRINT choice;
```

will output:
```
     Enter your choice:
```

on the bottom row of the terminal, then wait until a single character is pressed at the keyboard.

arrays

USAGE:
identifier(dim list)

where *dim list* is a list of numbers separated by commas and *identifier* is any valid CSG IMS identifier (see also).

PURPOSE AND OPERATION:
Arrays allow for easier manipulation of related items. An array is simply a collection of similar data items. An individual item of data in an array is called an element. Each element is referenced by an index. An array may have more than one index (or dimension). An array with one dimension (ie. a(n)) would be like a list. An array with two dimensions (ie. a(x,y)) would be like a table with rows and columns. An array with three dimensions (ie. a(x,y,z)) would be like a list of tables.

The size of an array is determined when the *identifier* is declared. A declaration like:

```
     INTEGER a(10)
```

will create an array consisting of a list of ten integers. a(1) refers to the first integer in the list, a(2) the second and so forth. The declaration:

```
     TEXT n(10,15) OF 20
```

will create an array of 150 elements (10x15), each element containing a text value of up to 20 characters.

Entire arrays may be passed as parameters between CSG IMS program modules. To accomplish this you must follow these steps:

- o Declare the array in the module from which the array is to passed. ie. INTEGER a(10,50).
- o Pass the array as a parameter to another module using the CALL statement. The array identifier must be followed by the open and close parentheses. ie. CALL spiff(a()).
- o The CALLed module must know the number of dimension that the array being passed to it has. This is specified by using "dummy" array indexes on the MODULE declaration line. ie. MODULE spiff(v(1,1)).

Notice that the array may be referred to by a different identifier in the CALLed module. This is due to the fact that array are passed by reference only. Even though an array may have a different identifier in the CALLing and CALLed modules, any changes made to the array by the CALLed program will appear in the array in the CALLing module.

EXAMPLE:

```
     MODULE foo
     INTEGER i
     REAL list(10),grid(4,8),threeD(2,2,2)
     CALL bar(list(),grid(),threeD())
     END

     MODULE bar(ary1(1),grid(1,1),threeD(1,1,1))
     END
```

In this example, any changes made in the module bar to the array ary() will also appear in the array list() in the module foo even though the arrays have different identifiers. Notice the use of the empty parentheses in the CALL statement and the use of "dummy" index values in the called module.

ESC#

USAGE:
```
     ESC#
```

ESC# (read "escape number") is a function which returns a number and may only be used in numeric expressions (see EXPRESSION).

PURPOSE AND OPERATION:
ESC# returns a permutated value of the ESCAPE function. When an ENTER statement is executed, the user may terminate entry of data by typing one of the following four keys: RETURN or ENTER (ASCII code 13), ESCAPE (ASCII code 27), UP ARROW (ASCII code 11), or DOWN ARROW (ASCII code 1Ø). The ASCII code of this key is then stored internally, and is the value of the ESCAPE function (see also) when it is evaluated. The ESC# function maps the value of ESCAPE into a number in the range of one to four. The mappings are as follows:

KEY STROKE	ESCAPE VALUE	ESC# VALUE
Escape	27	1
Up Arrow	11	2
Down Arrow	1Ø	3
Enter/Return	13	4

This function is useful in conjunction with screen form programs, since it may be used to go forward or backwards when ENTERing fields in a form.

EXAMPLE:
```
     LOOP
        LABEL enter_name
        ENTER maillist.name
        ON ESC# GOTO finished,enter_comment
        LABEL enter_country
        ENTER maillist.country
        ON ESC# GOTO finished,enter_name
        LABEL enter_comment
        ENTER maillist.comment
        ON ESC# GOTO finished,enter_country
     ENDLOOP
     LABEL finished
     RETURN
```

Note that the ON ESC# GOTO statement in the above example will simply continue execution at the next line when ESC# has a value of three or four (ie. RETURN or DOWN ARROW are pressed). When the escape key is pressed, ESC# will have a value of one causing execution to continue at the label *finished*. When the return key is pressed, ESC# will have a value of two causing execution to continue at the previous ENTER statement.

FMASK

USAGE:
```
     FMASK(n)
        or
     FMASK($)
```
where *n* is a number greater than zero and *$* is a text value representing the name of a field.

PURPOSE AND OPERATION:
FMASK returns the mask (see also) of field number *n* in the current file. If no mask was specified when the file was created, FMASK will return a null string (""). *n* must be greater than zero and less than or equal to the number of fields in the file. If *n* is less than one or greater than the number of fields in the file, an error number 44 will result.

If FMASK is called with a text argument ie. FMASK("name"), CSG IMS will return the mask of the named field in the current file. If the named field cannot be found in the current file, an error number 44 will result.

EXAMPLE:
```
     NOTE file "books" has fields dewey, cutter, title,
     NOTE and author
     INTEGER n
     OPEN "books"
     PRINT "Fieldname                   Default Mask"
     PRINT "-----------------------    -------------------------"
     SET TRAP TO trap44
     n=1
     LOOP
        PRINT FNAME(n); TAB(25);
        IF FMASK(n)="" THEN
           PRINT "none"
        ELSE
           PRINT FMASK(n)
        ENDIF
        n=n+1
     ENDLOOP
     LABEL modend
     END
     LABEL trap44
     RESUME AT modend
```

will result in:
```
     Fieldname                Default Mask
     ---------------------    ------------------------
     dewey                    ###.#
     cutter                   #####
     title                    none
     author                   none
```
See the example for FNAME for more information.

FNAME

USAGE:
```
     FNAME(n)
```
where *n* is a number greater than zero.

PURPOSE AND OPERATION:
FNAME returns the name of field number *n* in the current file. FNAME(1) would return the name of the first field, FNAME(2) would return the name of the second field and so on. If *n* is less than one or greater than the number of fields in the current file, an error number 44 will result.

EXAMPLE:
```
     NOTE file "books" has fields dewey, cutter, title,
     NOTE and author
     INTEGER n
     OPEN "books"
     PRINT "Fields in file 'books':"
     n=1
     SET TRAP TO trap44
     LOOP
        PRINT FNAME(n)
        n=n+1
     ENDLOOP
     LABEL modend
     END

     LABEL trap44
     RESUME AT modend
```

will result in:

```
Fields in file 'books:
dewey
cutter
title
author
```

LIST STATUS

USAGE:

```
     LIST STATUS
```

PURPOSE AND OPERATION:
LIST STATUS lists the status and values of the SET options.

EXAMPLE:

```
     IMS:LIST STATUS
     SCREEN        = ON
     INPUT         = OFF
     PRINT         = ON
     SINGLE        = OFF
     DATE          = M d, Y
     TOP MARGIN    = 3
     BOTTOM MARGIN = 62
     LEFT MARGIN   = 10
     RIGHT MARGIN  = 90
```

ON . . . GOSUB . . . RETURN

USAGE:

ON *n* GOSUB *label1, label2, . . . labeln*

where *n* is a numeric expression and *label1, label2, . . . labeln* is a list of zero or more labels (see LABEL).

PURPOSE AND OPERATION:
ON GOSUB performs an indexed multiway branch to subroutines. The numeric expression *n* is evaluated, and depending on its result, only one of the subroutines referenced by the list of labels will be chosen. If *n* is 1, the subroutine at *label1* is executed; if *n* is 2, the subroutine at *label2* is executed, and so on. If *n* is less than or equal to zero, or if *n* is larger than the number of labels in the list, none of the subroutines are called and execution continues at the first statement following the ON GOSUB.

EXAMPLE:

```
LOOP
   CLEAR SCREEN
   PRINT "1)  Enter check transactions"
   PRINT "2)  Enter deposit transactions"
   PRINT "3)  Print account statement"
   PRINT "4)  Quit this session"
   PRINT
   PRINT "Your selection: ";
   INPUT choice
   ON choice GOSUB checks,deposits,report,quit
ENDLOOP

LABEL checks
.
.
.
RETURN
.
.
.
LABEL quit
END
```

ON . . . GOTO . . .

USAGE:

ON *n* GOTO *label1, label2, . . . labeln*

where *n* is a numeric expression and *label1, label2, . . . labeln* is a list of zero or more labels (see LABEL).

PURPOSE AND OPERATION:
ON GOTO performs a simple indexed multiway unconditional branch. The numeric expression *n* is evaluated, and depending on its result, only one of the destinations referenced by the list of labels will be chosen. If *n* is less than or equal to zero, or *n* is greater than the number of labels in the list, execution continues at the first statement following the ON GOTO. If *n* is 1, execution continues at *label1*; if *n* is 2, execution continues at *label2*, and so on.

EXAMPLE:

```
     LABEL menu
     CLEAR SCREEN
     PRINT "1)  Enter check transactions"
     PRINT "2)  Enter deposit transactions"
     PRINT "3)  Print account statement"
     PRINT "4)  Quit this session"
     PRINT
     PRINT "Your selection: ";
     INPUT choice
     ON choice GOTO checks,deposits,report,quit
     PRINT "Invalid selection; press any key.";
     PRINT LIBRARY$(GETKEY)
     GOTO menu

     LABEL checks
     .
     .
     .
     GOTO menu
     .
     .
     .
     LABEL quit
     END
```

OPEN

USAGE:

```
     OPEN "pathlist" FOR READ  AS file tag
```

where *pathlist* is the name of a CSG IMS database, FOR READ is optional, and AS *file tag* specifies a file identifier and is optional.

PURPOSE AND OPERATION:
OPEN causes CSG IMS to OPEN the database named in *pathlist*, with *file tag* being the name for the file in the module. If the AS *file tag* clause is not present, the filename will be the *file tag*. The FOR READ clause causes IMS to allow only read type operations on the OPENed file. The current record of a file opened for read only will not be locked so that other processes will be able to access it.

You must OPEN a file before attempting to access the information in the file. An open file is referred to by its *file tag*. Files that are OPENed should be CLOSEd (see also) before exiting your program or exiting the Interactive Mode.

EXAMPLE:

```
     OPEN "DATA/vendor.list" AS VENDORS
```

causes CSG IMS to OPEN the file vendor.list.ida and vendor.list.iin in the directory DATA and creates the *file tag* VENDORS for that file.

```
     OPEN "members" FOR READ
     LIST
     CLOSE FILE members
```

causes CSG IMS to open the file members for read access only and creates the file tag members for that file. The file is then LISTed and subsequently CLOSEd.

SET SINGLE USER

USAGE:

```
     SET SINGLE USER  ON/OFF
```

PURPOSE AND OPERATION:
SET SINGLE USER permits the user to disable file header updating in applications where no other users will be accessing a file, significantly improving data access time.

```
     SET SINGLE USER ON
```

will disable header updating on all OPEN and subsequently OPENed files until a

```
     SET SINGLE USER OFF
```

is issued, returning operation to normal multi-user access.

CAVEATS:
While SINGLE USER is SET to ON, files may remain locked. Multi-user access to a file when SINGLE USER is ON could result in damage to the file structure.

SET TIMEOUT TO

USAGE:
    SET TIMEOUT TO *n*
where *n* is a number greater than or equal to zero and less than or equal to 65535.

PURPOSE AND OPERATION:
SET TIMEOUT TO *n* sets the length of time that CSG IMS will wait for a locked record to become unlocked before reporting an error. *n* represents the number of operating system ticks that CSG IMS will wait. On most OS9/68Ø9 Level Two systems a tick is 1/1ØØ of a second. TIMEOUT is initially set to zero when CSG IMS is executed. If *n* is zero a process will wait indefinitely to read a locked record. An *n* of one will cause CSG IMS to report error 252 (record locked) immediately on attempting to read a locked record. SET TIMEOUT to *n* affects only the file which is current at the time the SET is executed so each file can have a different timeout value.

SORT

USAGE:

SORT *file tag* *key clause* ON *exp* TO *filename* *range*

where *file tag* and *key tag* are optional and refer to the file to sort (see FILE TAG and KEY TAG). *exp* is the expression to sort by and *filename* is a text expression which evaluates to an IMS file name. *range* specifies the range of records to sort (see RANGE).

PURPOSE AND OPERATION:
SORT will sort an IMS data file by any expression consisting of field name(s) and/or operator(s). A new file will be created with the same structure as the file to be sorted. The records in the new file will be physically ordered according to the sort expression. No indexes will be created for the new file.

CAVEATS:
When sorting it is important to use as much memory as possible as it improves performance. When using CSG IMS in the interactive mode it is possible to use up to 32K bytes of memory (ie. IMSI #32K).

EXAMPLE:

```
     OS9:IMSI #32K
     IMS:OPEN 'gl.batch'
     IMS:SORT ON glcode TO 'gl.batch.srt' FOR glsource='AR'
     IMS:OPEN 'gl.batch.srt'
     IMS:LIST
```

This example causes IMS to create the file gl.batch.srt with the same record structure as gl.batch. After the SORT, gl.batch.srt will contain any records in gl.batch where the field glsource contains "AR" in sequenced according to the contents of glcode.

UNLOCK

USAGE:

```
     UNLOCK file tag
```

where *file tag* is optional and refers to an already OPENed file.

PURPOSE AND OPERATION:
UNLOCK releases control of the current record of the specified file. If *file tag* is not present, the current record of the current file is released. This is useful when a data base is being accessed by several users, in order to allow more than one user to read the same record without causing lockouts.

When CSG IMS reads a record in a file, that record becomes locked until it has been updated or a different record is read. Using UNLOCK forces CSG IMS to unlock the last read record allowing it to be accessed by other users.

EXAMPLE:

```
     OPEN 'maillist'
     FIND KEY name APPROX 'G'
     PRINT record, maillist.name
     UNLOCK FILE maillist
```

will find the first name in "maillist" starting with "G", print it out, then release that record so that other users can read or process it.

UTD

USAGE:

    UTD *function*

where *function* is one of the following universal terminal driver functions:

    ADDRESS *r,c*
    BOX *r1,c1,r2,c2*
    CLL
    CLS
    DOWN
    FULL
    HALF
    INIT
    LEFT
    NORMAL
    REVERSE
    RIGHT
    UP

where *r, c, r1, c1, r2,* and *c2* are numeric expressions.

PURPOSE AND USAGE:
The UTD function provides access to various facilities provided by the universal terminal driver (UTD). For complete information on the UTD, refer to the appendix and also the the UTD usage addendum in this document. The various functions are:

1. **ADDRESS *r,c***
   Absolute cursor address. This statement is identical to the LOCATE statement. See that entry in the reference manual for details.
2. **BOX *r1,c1,r2,c2***
   Box draw. This statement draws a simple rectangular box whose diagonal corners are at the absolute screen position *r1,c1,* and *r2,c2* respectively. The top left corner of the screen is assumed to be 1,1.
3. **CLL**
   Clear line from cursor to end of line. This is identical to the CLEAR LINE statement, described in the reference manual.
4. **CLS**
   Clear screen. This is identical to the CLEAR SCREEN statement, described in the reference manual.
5. **DOWN**
   Cursor down. This UTD function moves the cursor down by exactly one line.
6. **FULL**
   Full intensity. This statement switches the terminal to displaying characters in full intensity video attribute.

7. **HALF**
   Half intensity. This UTD function is the complement to the UTD FULL statement; it causes characters to be displayed in the half intensity video attribute.
8. **INIT**
   Terminal initialization. This UTD function initializes or resets the terminal to the default video mode(s) and/or attribute(s) required by the UTD.
9. **LEFT**
   Cursor left. This statement moves the cursor exactly one character position to the left.
10. **NORMAL**
    Normal video. This statement turns off the reverse video attribute, described below.
11. **REVERSE**
    Reverse video. This UTD function turns on the reverse video attribute for the terminal.
12. **RIGHT**
    Cursor right. This statement moves the cursor exactly one character position to the right.
13. **UP**
    Cursor up. This UTD function moves the cursor up exactly one line.

Changes to the Text Editor

Several small changes have been made to the text editor (tx) to make it even more useful.

OVERWRITE MODE

Overwrite Mode is selected from the new Option Menu (^O). When Overwrite Mode is on, a character typed on the keyboard will replace the character under the cursor. Indent is now controlled from the Option Menu.

FIND/REPLACE

The find/replace definition sequence has been improved. The Define function in the Find function menu has a slightly different operation than is described in the manual; it no longer prompts the user to find all occurrences, and returns to text mode when definition is complete. Additionally, when a replacement string is defined, performing a find next or find previous will prompt the user with the string "replace text (Y/N*/A) ?", where "A" stands for "all". Selecting "all" replaces all subsequent occurrences of the find search string without further prompting.

FUNCTION KEY SUPPORT

The user may now redefine the control keys used by tx and also make use of special function keys on most terminals. See the chapter on "Using the Universal Terminal Driver" for further information.

IMS Supplementary Documentation
Revision History

JULY 31, 1986 - Version 1.3 Released

- function key support added to UTD. Utilized by tx, imsF and imsR.
- LIST STATUS command added. Lists the current values of SET variables that are not otherwise accessible.
- imsD changed to create data files such that strings are at the end of the data record for compatibility with 68000 version. Only files created with imsD version 1.3 will be completely portable with 68000 versions of CSG IMS. Files created on earlier version *may* require conversion.
- FNAME(*n*) and FMASK(*n*) functions added.
- SORT statement added using heap sort and merging
- SET SINGLE USER ON/OFF added for faster single user operations.

MARCH 1986 - Version 1.2 Released

- array passing capability added.
- Options Menu added to tx.
- Find function in tx slightly enhanced.
- tutd and device utilities added to improve functionality of UTD.
- TVI 955 132 column support added to UTD.
- SET TIMEOUT TO n added to enhance multi-user file access capability.
- UNLOCK added to enhance multi-user file access capability.
- UP ARROW and DOWN ARROW added to ENTER function, to allows backtracking through fields in a form.
- #COLUMNS, #ROWS, ESC# functions added.
- ON n GOTO, ON n GOSUB and UTD function access added.

FEBRUARY 1986 - Version 1.1 Released

- Index Structure changed from BTree to concurrent B+Tree. Re-index of databases from version 1.0 required.

- o FOR READ clause added to OPEN statement for enhanced multi-user access capability.

JANUARY 1986 - Version 1.0 Released

IMS Supplementary Documentation
Errata

CSG IMS Manual Revisions and Corrections to Release B

Tutorial, Page 1
The fourth paragraph states that the DEL key is used to delete characters when a line of text is input. With the latest release of CSG IMS, only the editing programs (tx, imsF, imsR) use the DEL key for character deletion. All other portions the package use the standard OS9 backspace character for character deletion.

Tutorial, Page 4
As the maillist program is supplied, the screen shown opposite of page 4 will not be what is displayed when the program is first invoked. To get this screen, type ESC L (last record) and ESC P (previous record) when the maillist program first comes up.

Tutorial, Page 37
The alteration to the line "a1 = a1 + check_data.gross" should not be implemented if the check report is to operate correctly.

Tutorial, Page 61
The result given as the standard deviation is indeed the standard deviation of that list of six numbers. However, the program, as written, calculates the population deviation, which means the result printed should be 4.4091203907356, rather than 4.0249578279365.

Appendices, Page 1
The given list of files which should be in the CMDS directory should include all of the following files:

ims	the executive (menu)
imsI	the interpreter
imsC	the compiler
imsD	the file creator
imsF	the screen form editor
imsR	the report generator
imsR.statements	statements used by program generator
imsErrs	error message file
tx	text editor
mkterm	terminal driver editor
assoc	terminal association editor
nmall	UTD utility
device	UTD utility
tutd	UTD utility
tname	UTD utility
maillist	program for lesson 1

If the file csg_imsI appears in your execution directory, it may be deleted as it is no longer required.

Universal Terminal Driver Usage Reference

The universal terminal driver (UTD) is a system developed by Clearbrook Software Group (CSG) in order to allow for terminal independent display functions. The UTD in fact supports a fairly wide variety of terminal functions, but not all CSG software utilizes all of these functions.

The basic operation of the UTD is simply to reference a small table of conversion strings and numbers whenever a particular function is required of it. When the UTD is initialized, it reads this table from a file in the UTD directory in the execution directory. The file is named after the OS9 device the terminal is attached to. The files in the UTD directory are referred to as terminal-driver associations. Within the UTD directory is the UTD_DRIVER_FILES directory. This contains the basic tables for individual terminal models. These files are also know as driver files.

The UTD comes with six support programs necessary in order to utilitize the UTD. These are: **mkterm, assoc, tname, nmall, device,** and **tutd**. Their purpose and use is as follows:

1) **mkterm** -

Syntax:

```
OS9:mkterm [<source_driver> [<destination_driver>]]
```

Description:

This program allows a driver file to be created and/or modified. It is necessary if the driver for a particular terminal model is not on the distribution diskette, or if one wants to experiment with existing drivers to achieve different effects. For a complete explanation of how to use the program and how to set up a custom driver file, see the text below these descriptions.

Examples:

```
OS9:mkterm                  ; * edit a blank driver
OS9:mkterm tvi_910          ; * create TVI910 driver
OS9:mkterm tvi_910 qvt_102  ; * create QVT102 driver
OS9:                        * based on TVI910 driver.
```

2) assoc -

<u>Syntax</u>:

```
OS9:assoc <device> [<driver>]
```

<u>Description</u>:

This program is used to define a terminal-driver association. This program <u>must</u> be used when installing the software in order to let the UTD know what driver files each terminal uses. If two parameters are given, an association between those two items is made. If only one parameter is given, the existing association with that device is deleted. Each time the driver for an existing terminal-driver association is altered (I.E. - edited with mkterm), this command <u>must</u> be reapplied to it.

<u>Examples</u>:

```
OS9:assoc term vt100 ;* /term associated with VT100
OS9:assoc t1 qvt_102 ;* /t1 attached to a QVT102
OS9:assoc t1         ;* delete /t1 association
```

3) nmall -

<u>Syntax</u>:

```
OS9:nmall
```

<u>Description</u>:

This program is simply a utility for viewing data regarding the current state of the UTD. It will first list out the currently defined OS9 terminal devices (but not their associations), then it will list the UTD drivers that are defined.

<u>Examples</u>:

```
OS9:nmall

Universal Terminal Driver defined devices:
------------------------------------------

term

Universal Terminal Driver defined driver files:
-----------------------------------------------

qvt_102
tvi_910
vt100
```

4) **tname** -

Syntax:
```
OS9:tname {<device>}
```

Description:

This program, like nmall, is simply a utility for viewing data regarding the current state of the UTD. Specifically, tname is used to see what the current terminal-driver associations are for the list of OS9 devices given as parameters to the command.

Examples:
```
OS9:tname term t1      ;* ask for /term and /t1
term is associated with vt100
t1 is not defined
```

5) **device** -

Syntax:
```
OS9:device
```

Description:

This program is a utility designed to simplify the use of the UTD. When invoked, it will print the name of the OS9 device to which path 1 has been opened. I.E. - it tells you the name of your terminal.

Examples:
```
OS9:device         ; * ask for device name on path 1
TERM
```

6) **tutd** -

Syntax:
```
OS9:tutd
```

Description:

This program is another utility for use with the UTD. It is used to test whether a driver for a specific terminal model works correctly. It is self prompting, and the actions it performs should indicate whether or not a particular terminal function was correctly implemented in the driver file. Note that tutd expects to have the terminal it operates on to have been associated with a driver file.

Examples:
```
OS9:tutd        ; * self prompting from this point.
```

When CSG-supplied software utilizes the UTD, part of the installation process includes configuring the UTD to work with your particular terminals. To start off, find out what drivers were provided on the distribution diskette by typing nmall at the shell prompt. If you see the terminal model(s) for the terminal(s) attached to your system in the resultant list, simply associate them to the appropriate terminal devices.

For example, if a DEC VT100 is attached to "/term", and a QUME QVT102 is attached to "/t0", type the following commands:

```
        OS9:assoc term vt100
        OS9:assoc t0 qvt_102
```

This must be done for each terminal on your system on which the software will used.

If any of the driver files needed for the terminals on your system are not included, you will have to create them yourself using **mkterm**. Suppose one of the terminals you have for which the UTD has no driver file is a TeleVideo TVI910. Type the following command at the shell prompt:

```
        OS9:mkterm tvi_910
```

The program will print the following menu:

```
 1 - Number of rows.                          0
 2 - Number of columns.                       0
 3 - Terminal initialization.                 --
 4 - Cursor up.                               --
 5 - Cursor down.                             --
 6 - Cursor left.                             --
 7 - Cursor right.                            --
 8 - New line.                                --
 9 - Clear screen and home cursor.              --
10 - Clear from cursor to end of line.        --
11 - Reverse video.                           --
12 - Normal video.                            --
13 - Half intensity.                          --
14 - Full intensity.                          --
15 - Scroll screen up.                        --
16 - Scroll screen down.                      --
17 - Insert line at cursor.                   --
18 - Delete line under cursor.                --
19 - Menu 1
20 - Keystroke definitions

Your selection (or type exit) ?
```

The information for items 1-18 should be entered. To enter or alter a particular item, type the number of the item at the prompt, then press carriage return. You should define as many of these items as possible. CSG software using the UTD will try to emulate a particular terminal function if the UTD driver does not have that function defined. If it so happens that a terminal does not support a particular item, leave it blank. But there are some functions which the software must have defined if is to work correctly; using the tutd program will inform you which functions need to be defined and which are optional. Enter the appropriate data in the manner described in the following paragraphs.

When you have finished entry of items, or wish to save or quit, type the word "exit". **Mkterm** will prompt:

Do you want the current changes saved ?

Type the word "yes" in response. The program will then confirm the name of the driver:

Save as tvi_910 ?

Again, answer yes. This will terminate the current **mkterm** session.

Items 1 and 2 are the row and column dimensions, respectively. When these are selected for entry, simply enter an unsigned decimal integer. For the TVI910, there are 24 rows and 80 columns.

For items 3 through 18, a string describing that terminal function must be entered. This string is the sequence of ASCII characters which are required to perform that particular function on the terminal in question. This string may in fact be entered as a line of text; the corresponding ASCII characters are read from standard input. Since most terminal functions require the use of control characters, and the string is read with OS9 line editing features enabled, there are a number of mechanisms available to allow the entry of otherwise unprintable characters.

The first of these is a caret (^) placed in front of a character. When this appears, the corresponding control code of that character is substituted. For example, ^m is carriage return.

The second mechanism is the hex literal. This is specified with $XX where X is a hexadecimal character. This allows a non-printable non-control character to be defined. For example, $41 is A (capitol a), and $0d is carriage return. If X is not a hexadecimal character, it is assumed to be 0.

The third and final mechanism is the backslash (\). This is used to permit the specification of characters which otherwise have lexical significance to the above specification mechanisms. Thus, \^, \$, and \\ must be used to specify the ^, $, and \ characters in a string.

So, the strings for terminal items 3 through 18 on the TVI910 are:

3) **Terminal initialization -**
This function is simply a general purpose initialization and/or reset. If your terminal does not automatically wrap around at the end of a line, use this function to set wrap around on. It is not necessary to define it for the TVI910.

4) **Cursor up -**
This function string will move the cursor up by exactly one line; the column position will not be affected. If the cursor is at the top line of the display, the effect of this string is allowed to be undefined. For the TVI910, enter the string "^K". When you are prompted to enter the string, do not enter the double quotes.

5) **Cursor down -**
This function string will move the cursor down one line; the column position will remain unchanged. If the cursor is at the bottom line of the display, this string is allowed to have undefined results. The corresponding string for the TVI910 is "^J". Again, don't include the quotes.

6) **Cursor left -**
This function string will move the cursor one character to the left. The string for the TVI910 is "^H". At the leftmost column, the cursor should wrap around to the end of the previous line.

7) **Cursor right -**
This function string will move the cursor one character to the right. The string for the TVI910 is "^H". At the rightmost column, the cursor should wrap around to the start of the next line.

8) **New line -**
This function should position the cursor at the start of the next line. If the cursor is already at the bottom line of the display, the screen should scroll up one line as well. The corresponding string for the TVI910 is "^_".

9) **Clear screen and home cursor -**
This function should erase the entire display and place the cursor at the top left corner of the display. The string for the TVI910 is "^Z".

10) **Clear from cursor to end of line -**
This function erases the characters lying between the cursor and the end of the line the cursor is on, inclusive. The position of the cursor remains unchanged. The corresponding string for the TVI910 is "^[T". Note that the ^[is the UTD's notation for the escape code (ESC, $1b). Once again, don't include the quotes.

11) **Reverse video -**
This should initiate the reverse video attribute. It is not assumed that this attribute will be transparent, and it will not be assumed that it will be able to extend over more than one line while it is invoked. In all practicality, this could be defined in fact to be any video attribute; it is merely known to the UTD by this name. The string for the TVI910 is "^[G4".

12) **Normal video -**
This function should terminate the reverse video attribute. The same provisos for that item apply here as well. The string for the TVI910 is "^[G0".

13) **Half intensity -**
This function should initiate the half-intensity video attribute. It _is_ assumed that it will be transparent to the terminal; I.E. - any random character on the screen can be printed while this attribute is on. As for the name of half intensity, it is merely the UTD's reference to the attribute. It may in all practicality be any transparent attribute. The string for the TVI910 is "^[)".

14) **Full intensity -**
This function should terminate the half-intensity video attribute. The string for the TVI910 is "^[(".

15) **Scroll screen up -**
This function should scroll all the text on the screen up by one line. There is no required position of the cursor upon completion. The string for the TVI910 is "^[=7o ". Note that there is a space imbedded in this string. What this string does is address the cursor to the bottom right corner of the screen, then prints a space. Other terminals may or may not be able to use this strategy.

16) Scroll screen down -
This function should scroll the text on the screen down by one line. The placement of the cursor is allowed to be undefined upon completion. This function is not available on the TVI910.

17) Insert line at cursor -
This function should move all text lines from the cursor below down exactly one line. This includes the line the cursor is on. The line thus inserted should also be blank. The placement of the cursor upon completion need not be defined. The TVI910 does not support this function.

18) Delete line under cursor -
This function should move all lines immediately below the one on which the cursor is up exactly one line. The bottom line of the display should erased. The TVI910 does not support this function.

When you have completed filling in these items, choose item 19, to go to the following menu:

```
 1 - Box drawing lead in sequence   --
 2 - Box drawing termination        --
 3 - Box string: upper left corner  --
 4 - Box string: lower left corner  --
 5 - Box string: upper right corner --
 6 - Box string: lower right corner --
 7 - Box string: vertical bar       --
 8 - Box string: horizontal bar     --
 9 - Box string: downward T         --
10 - Box string: upward T           --
11 - Box string: left T             --
12 - Box string: right T            --
13 - Box string: crosshair          --
14 - Address the cursor.            -- [-;-;-] -- [-;-;-] --
15 - Menu 1
16 - Keystroke definition

Your selection (or type exit) ?
```

This menu allows you to define two kinds of functions: box drawing and cursor addressing. The 13 selections which relate to boxes are used by some CSG software for the purpose of simple graphics. While many terminals don't have special line or box drawing characters or modes, it is probably a good idea to define these items anyway. For instance, the TVI910 does not have any such special characters. What will be used for them instead are other ASCII characters, which are printed in half-intensity. Using such a method, it should still be possible then for simple terminals to support a kludge for true line and box drawing.

1) **Box drawing leadin sequence** -
This function initiates the gr^phics mode or special characters or video attribute or whatever is needed to start drawing box characters on the terminal screen. The TVI910 does not have any special graphics characters, but it does have a transparent video attribute (half intensity) which can be used to distinguish box characters from others. The string for the TVI910 then is "^[)".

2) **Box drawing termination** -
This function is the complement of the above function; it terminate the special character mode or video attribute or whatever is used to initiate the box drawing mode. For the TVI910, this means returning to full intensity, which is done with the string "^[(".

3) **Box string: upper left corner** -
This string should be the character sequence which prints a single character on the screen which should appear as the upper left hand corner of a box. Since the TVI910 has no special characters, use the string "+".

4) **Box string: lower left corner** -
This string should print a character which appears as the lower left hand corner of a box. Use "+" for the TVI91

5) **Box string: upper right corner** -
This should print a character which represents the upper right hand corner of a box on the display. For the TVI910 this should be "+".

6) **Box string: lower right corner** -
This should print a character which appears as the lower right hand corner of a box on the display. Use the string "+" for the TVI910.

7) **Box string: vertical bar** -
This should print a character which represents a vertical line on the screen. For the TVI910, this should be "|".

8) **Box string: horizontal bar** -
This should print a character which appears as a horizontal line. The string for the TVI910 should be "-".

9) **Box string: downward T** -
This should print a character which would logically result from the intersection of a horizontal bar and the bottom half of a vertical bar. The string for the TVI910 should be "+".

10) Box string: upward T -

This should print a character which would logically result from the intersection of a horizontal bar and the top half of a vertical bar. The string for the TVI910 should be "+".

11) Box string: left T -
This should print a character which would logically result from the intersection of a vertical bar and the left half of a horizontal bar. The string for the TVI910 should be "+".

12) Box string: right T -
This should print a character which would logically result from the intersection of a vertical bar and the right half of a horizontal bar. The string for the TVI910 should be "+".

13) Box string: crosshair -
This should print a character which would logically result from intersecting the vertical and horizontal bar characters. For the TVI910, use the string "+".

14) Cursor Addressing -
This function is among the more complex of the functions which terminals support. Selecting this item will cause a prompt for the "leadin sequence". This is the initial string by which the terminal knows it is supposed to address the cursor. For the TVI910, this string is "^[=".

You will next be prompted for a coordinate specification. This is a method of specifying how an actual cursor coordinate is sent to the terminal. It consists of: row or column selection (S), addressing type (T), and coordinate offset (O). The UTD assumes that the top left corner of the screen is address 0,0.

In order to enter a coordinate specification, it must be typed in the form S;T;O at the appropriate prompt. S may be R for row or C for column. T is a decimal integer in the range of 1 to 15. O is a decimal integer in the range 0 to 255. The addressing types supported by the UTD are:

1) Linear transformation; the coordinate is sent as a binary byte. The TVI910 is one terminal with this type of addressing.

2) ASCII string; the coordinate is converted to an unsigned string of ASCII digits and sent to the terminal. Any ANSI standard terminal uses this type of addressing.

3) BCD transformation; the coordinate is converted from a binary to a BCD byte. The column coordinate of the ADDS_25 is one terminal with this addressing type.

4) 8Ø/132 column irregular transformation; this type is characterized by the requirement that for a terminal with 132 columns, the middle sequence string for a column address of less than 8Ø is different than the leadin string when the column address is greater or equal to 8Ø. It causes the tilde ( ) to be the middle sequence string when the column coordinate is greater or equal to 8Ø; otherwise it is null. One terminal with this type is the TeleVideo TVI955 when it is in 132 column mode.

The TVI91Ø expects the row coordinate first, as a binary byte, and with an offset of 32 added to it. Thus, type R;1;32 when mkterm prompts for the coordinate specification.

Next, many terminals require a string between the two cursor coordinate specifications. This is not the case for the TVI91Ø, so in this instance simply press carriage return.

At this point the second coordinate specification must be entered, as described above. For the TVI91Ø, the second coordinate is the column, transformed linearly with an offset of 32. Thus, the specification is: C;1;32.

Finally, many terminals also require a string to indicate the end of the cursor addressing function. The TVI91Ø requires none, so simply press carriage return at this point.

When these items have been completed, choose item 16; the following menu should appear:

```
 1 - Abort (Default = ^A)                    --
 2 - Block mode (Default = ^B)               --
 3 - Insert character (Default = ^C)         --
 4 - Delete under cursor (Default = ^D)      --
 5 - End of text (Default = ^E)              --
 6 - Find/replace menu (Default = ^F)        --
 7 - Cursor left (Default = ^H)              --
 8 - Insert line (Default = ^I)              --
 9 - Cursor down (Default = ^J)              --
10 - Cursor up (Default = ^K)                --
11 - Cursor right (Default = ^L)             --
12 - New line (Default = ^M)                 --
13 - Next screen (Default = ^N)              --
14 - Options menu (Default = ^O)             --
15 - Previous screen (Default = ^P)          --
16 - Undelete character (Default = ^U)       --
17 - Delete line (Default = ^X)              --
18 - I/O menu (Default = ESCAPE)             --
19 - Start of text (Default = HOME)          --
20 - Delete character (Default = DEL)        --
21 - Menu 1
22 - Menu 2

Your selection (or type exit) ?
```

This menu allows you to define alternate keys for particular editor functions; I.E. - you can assign function keys to specific editor functions. Note that since these key definitions are made in the driver for a particular terminal model, all devices associated with that terminal driver will have the same key definitions. Note also that for the particular CSG software package you have received, not all of the functions given in the menu may be supported. Since the UTD is a general purpose item included in many CSG products, though, all of the possible key functions are listed.

The ability to assign editor functions to different function keys is very useful indeed. For instance, the cursor keys of a DEC VT100 terminal each output a 3-character code; re-defining the cursor up, cursor down, cursor left and cursor right functions (items 10, 9, 7, and 11 respectively) to these codes would allow the editors to use the cursor keys of the VT100. In addition, the HOME key and PF1 through PF4 keys could also be assigned specific editor functions.

To redefine an editor function, type the number of the desired item at the menu prompt and press return. Then simply type the string of ASCII characters which the function key, to which you are assigning the function, should produce and press return. This string is in the format defined previously; it may include the various character-conversion mechanisms.

Using the TVI910 as an example, it turns out that all the editor functions can be accessed with existing control keys and other dedicated function keys. However, it has been found that it is convenient if the BACK TAB key is assigned to the function of end of text (item 5). Here then are the descriptions of the various functions, as well as the redefinition of the BACK TAB key:

1) **Abort (Default = ^A)** -
This function interrupts any operation using OS9's QUIT function supported in SCF devices. Because of this, this function must be exactly one character, and no other function can be defined using a function key sending this character.

2) **Block mode (Default = ^B)** -
Only in tx does this enter block mode; in imsF (from the CSG IMS package) it enters the box drawing menu, and in imsR (from CSG IMS) it enters the border redefinition menu.

3) **Insert character (Default = ^C)** -
This function is used to insert a blank character; it is not used in tx.

4) **Delete under cursor (Default = ^D)** -
This deletes the character the cursor is currently on.

5) **End of text (Default = ^E)** -
This function moves the cursor to the end of the line, screen, then file. To assign end of text to BACK TAB (ESC I), type the string "^[I", as usual without the quotes.

6) **Find/replace menu (Default = ^F)** -
This initiates a function menu; on tx it is the find/replace menu, or field placement menus in imsF and imsR.

7) **Cursor left (Default = ^H)** -
This moves the cursor one character to the left, with cursor wrap around to the previous line.

8) **Insert line (Default = ^I)** -
This inserts a blank line, scrolling down the lines below the cursor. Not all CSG software has this function.

9) **Cursor down (Default = ^J)** -
This function moves the cursor down one line, scrolling the contents of the screen if necessary.

10) Cursor up (Default = ^K) -
This function moves the cursor up one line, scrolling the contents of the screen if necessary.

11) Cursor right (Default = ^L) -
This moves the cursor one character to the left, with cursor wrap around to the previous line.

12) New line (Default = ^M) -
This function places the cursor at the start of the next text line; in tx it creates a new text line as well.

13) **Next screen** (Default = ^N) -
This function causes the program to go to the next screen of text lines.

14) Options menu (Default = ^O) -
This initiates a menu which allows various operational parameters to be controlled. It is available only in tx.

15) **Previous screen** (Default = ^P) -
This function causes the program to go to the previous screen of text lines.

16) **Undelete character** (Default = ^U) -
This function inserts characters previously deleted with the DEL or ^D deletion functions. It is available only in tx.

17) **Delete line** (Default = ^X) -
This function deletes the line that the cursor is on.

18) **I/O menu** (Default = ESCAPE) -
This initiates a menu which is generally used for selecting input/output functions.

19) **Start of text** (Default = HOME) -
This function moves the cursor to the start of the line, screen, then file.

20) **Delete character** (Default = DEL) -
This function deletes the character to immediate left of the cursor.

After completing this procedure, you will have created a driver file for the TeleVideo TVI910 terminal. Be sure to save it when you exit. Once it is saved, you must associate the OS9 devices to which your TVI910 terminals are attached with tvi_910. Just to be sure you have defined the driver file correctly, run the tutd command, as described previously in this section. REMEMBER also that any time you edit this driver file, you must reassociate any terminals which use the driver.

CLEARBROOK SOFTWARE GROUP INC.
ABBOTSFORD, BRITISH COLUMBIA