

**Mini-micro CDS/ISIS  
CDS/ISIS PASCAL  
(Version 2.3)**

Division of Software Development and Applications  
Office of Information Programmes and Services

Unesco  
7, Place de Fontenoy  
75700 Paris

March 1989

(c) Unesco

(c) Unesco 1989

*All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of Unesco.*

First published 1989 by the  
United Nations Educational,  
Scientific and Cultural Organization,  
7, Place de Fontenoy  
75700 Paris, France

ISBN 92-3-102-605-7 (Unesco)

### *Introduction*

CDS/ISIS Pascal is a programming language designed to develop CDS/ISIS applications requiring functions which are not readily available in the standard package. In order to use CDS/ISIS Pascal you must therefore be familiar with the Pascal programming language and, obviously, with CDS/ISIS. You do not require, however, any detailed knowledge of CDS/ISIS internals, such as file structure and record format. The specificity of CDS/ISIS Pascal is in fact its library of pre-defined procedures, which provide access to most CDS/ISIS functions in a convenient and simple manner. For example, with a simple call to the MENU function you may display any CDS/ISIS system menu or any other menu you have designed for a specific application and obtain the choice made by the user; similarly, the CREATE and MODIFY functions allow you to create or edit a record interactively using the full CDS/ISIS data entry facilities.

Furthermore, in addition to providing you with a powerful and high-level interface to the CDS/ISIS software, the CDS/ISIS Pascal library will make your programs independent from the version of the system you use and give you the security of using well tested functions.

The "User guide" on page 1 describes the procedures for compiling and running CDS/ISIS Pascal programs; the language itself, which is a subset of standard Pascal, is described in "Language Specifications" on page 5; "The CDS/ISIS Pascal Library" on page 13 provides a full description of the CDS/ISIS Pascal library; and the section "Sample Programs" on page 35 describes the sample programs supplied on the distribution diskettes.



*Contents*

<b>Introduction</b> .....	<b>iii</b>
<b>Section 1: User guide</b> .....	<b>1</b>
CDS/ISIS Pascal .....	1
Compiling and executing a program .....	1
Files produced by CDS/ISIS Pascal .....	2
Compiler error messages .....	2
Run-time errors .....	3
<b>Section 2: Language Specifications</b> .....	<b>5</b>
Constants, Types and Variables .....	5
Expressions .....	5
Statements .....	6
Procedures and Functions .....	6
Scope of Identifiers .....	7
TEXT files .....	7
Comments .....	8
Designing and integrating user exits .....	8
MENU Exits .....	8
FORMAT Exits .....	10
<b>Section 3: The CDS/ISIS Pascal Library</b> .....	<b>13</b>
Overview .....	13
General procedures and functions .....	13
Screen management .....	13
Keyboard management .....	13
Type conversion .....	13
String utilities .....	14
Miscellaneous .....	14
CDS/ISIS procedures and functions .....	14
System .....	14
Data base .....	14
Master file .....	14
Inverted file .....	15
Searching .....	15
Formatting .....	15
Editing and data entry .....	15
Functional description .....	15
Procedure ASSIGN (s1, s2: string); .....	15
Procedure ATTR (fill: string; attr, lin, col, len: real); .....	16
Procedure AUTOTYPE (s1: string); .....	16
Procedure BOX (lin, col, h, w, frame: real); .....	17
Procedure CHATTR (attr, lin, col, len: real); .....	17

Function CHR (n: real): string; . . . . .	17
Procedure CLEAR; . . . . .	17
Procedure CLEARBOX (lin, col, h, w, attr: real); . . . . .	18
Procedure CLEARDATA; . . . . .	18
Procedure CLEARLN; . . . . .	18
Procedure CLEARMSG; . . . . .	18
Procedure CLOSE; . . . . .	18
Function CREATE: real; . . . . .	18
Procedure CURSOR (lin, col: real); . . . . .	18
Function DATAENTRY (var s1: string): real; . . . . .	18
Function DATESTAMP: string; . . . . .	20
Function DBN: string; . . . . .	20
Procedure DEFKEY (k: real; s: string); . . . . .	21
Procedure DELETE; . . . . .	21
Function DELTERM (t: string): real; . . . . .	21
Procedure DUMMYREC; . . . . .	21
Function EDIT (var s: string; max, line, col, wsize, attr: real; fill: string): real; . . . . .	22
Function ENCINT (n1, n2: real): string; . . . . .	23
Function ENCREAL (n1, n2, n3: real): string; . . . . .	23
Procedure EXEC (s1: string); . . . . .	23
Function FIELD (n: real): string; . . . . .	23
Function FIELDN (tag, occ: real): real; . . . . .	23
Function FILEXIST (f: string): real; . . . . .	23
Function FIND (var s1: string): real; . . . . .	24
Function FLDADD (tag, n: real; s: string): real; . . . . .	24
Function FLDCHA (n: real; s1, s2: string): real; . . . . .	24
Function FLDDDEL (n: real): real; . . . . .	24
Function FLDMOD (n1, n2, n3: real; s: string): real; . . . . .	24
Function FLDREP (n: real; s: string): real; . . . . .	25
Function FMTNAME : string; . . . . .	25
Function FORMAT (lw: real): real; . . . . .	25
Function GETCC : real; . . . . .	25
Function GETCL : real; . . . . .	25
Procedure GETFMT (s: string); . . . . .	25
Function INKEY: string; . . . . .	26
Function KBDKEY (var c: string): real; . . . . .	26
Function LANG: string; . . . . .	27
Function LINES: real; . . . . .	27
Function LOCK: real; . . . . .	27
Function MAXMFN: real; . . . . .	27
Function MENU (s: string): string; . . . . .	27
Function MODIFY (n: real): real; . . . . .	28
Procedure MSG (n: real); . . . . .	28
Function MSGTEXT (n: real): string; . . . . .	28
Function NEWREC: real; . . . . .	28
Function NFIELDS: real; . . . . .	28
Function NOCC (n: real): real; . . . . .	28
Function NXTLINE (var lin: string): real; . . . . .	29
Function NXTPOS (n: real): real; . . . . .	29
Function NXTPOST: real; . . . . .	30
Function NXTTERM: string; . . . . .	30

---

Procedure OPEN (s: string);	30
Function ORD (s: string): real;	30
Procedure PAGE (n: real);	30
Function PATH (s1: string; n: real): string;	30
Function POSITION (s1,s2: string; n: real): real;	31
Function POSTING (s1: string): real;	31
Function RECORD (n: real): real;	31
Procedure SAVESCR (n: real);	32
Function SEARCH (s: string): real;	32
Function SETLANG (s: string): real;	32
Function SETPOS (n1, n2: real): real;	33
Function SIZE (s: string): real;	33
Function SUBSTR (s: string; n1, n2: real): string;	33
Procedure UC (var s: string);	33
Procedure UNLOCK;	33
Procedure UPDATE;	34
Procedure UPDIF;	34
Function VAL (s: string): real;	34
Function WORKSHEET (s: string): real;	34
<b>Section 4: Sample Programs</b>	<b>35</b>
Program KEYB	35
Program DISPL	36
Program TEXT	36
Program THES	36



## Section 1

### *User guide*

#### A. CDS/ISIS Pascal

CDS/ISIS Pascal is an integral part of CDS/ISIS and consists of a compiler, an interpreter and a library. The compiler produces a pseudo-code which is then executed by the interpreter. Because the executable code is machine-independent, application programs written in CDS/ISIS Pascal are fully portable across the whole spectrum of computers supported by CDS/ISIS (i.e. an application developed on an IBM PC microcomputer will run without change on a VAX computer). Note, however, that some library procedures, are only available on certain computers. This fact is noted in the description of the relevant procedures. If portability is essential, you should avoid using these procedures and select alternate methods for obtaining the intended result.

You may design programs to run either in stand-alone mode (which provides the equivalent of a batch mode operation) or as user exits in certain CDS/ISIS functions. User exits provide a powerful way to extend the functionality of CDS/ISIS.

#### B. Compiling and executing a program

To enter a program you may use any familiar text editor which is available on your computer (e.g. EDLIN). Note that the compiler does not accept tabulation characters (ASCII code 9). The program must be stored in a file with the extension **PAS**, in the program directory defined in parameter 1 of SYSPAR.PAR (see the section "System installation" in the *CDS/ISIS Reference Manual*).

To compile a program, select option A from the main CDS/ISIS services menu xXISI. You will then be prompted to select an option from the following submenu:

```
C[ompile]  R[un]  Q[uit]  ?
```

Select the option as follows:

**C** to compile a program. You will then be prompted to enter the name of the program. Enter the program name (without the extension). You may suffix the program name with the list compilation switch as follows:

**/L** to obtain a listing of the source and compiled code. By default no listing is provided unless a compilation error is detected.

The switch may be entered in upper or lower case. For example: **myprog/l**

**R** to execute a program. As for option **C**, you will be prompted to enter the name of the program (unless you are executing the one just compiled).

**Q** to exit and return to the main menu xXISI.

Option **C** is used to compile both batch programs and user exits. To execute a batch program select option **R**. Note that this option is designed for running batch programs. You should not use it to execute user exits. The execution of a user exit is controlled by the CDS/ISIS program or function for which it was designed (see under "Designing and integrating user exits" on page 8).

### C. Files produced by CDS/ISIS Pascal

Two files are produced by the compiler (xxxxx is the program name):

**xxxxxx.LST** which contains the listing of the source program and a symbolic listing of the compiled code. Note that this file is only produced if you use the **/L** compilation switch or if a compilation error is detected.

**xxxxxx.PCD** which contains the executable code.

If any syntax errors are detected during compilation, the listing file will contain an appropriate error message, marking the point at which the error was found. The program line containing the error as well as the message is also displayed on the screen. A program which produced compilation errors will not be executed. The interpreter produces an error message to this effect if you try to do so.

Note that **PCD** files are stored in the program directory defined in parameter 1 of SYSPAR.PAR, whereas **LST** files are stored in the work files directory defined in parameter 4 of SYSPAR.PAR (see the section "System installation" in the *CDS/ISIS Reference Manual*).

### D. Compiler error messages

Error messages which may be produced by the compiler are listed below (note that when an error is encountered several error messages may be produced; in this case, it is the first of these which is significant):

- PROGRAM expected
- Identifier expected
- ; expected
- . expected
- := expected
- Program too large (the compiler cannot handle your program, try to reduce its size, for example by grouping repeating code in procedures or functions; or break it up into two programs, if possible)
- END expected (may also be caused by calling a function as a procedure)
- THEN expected
- DO expected
- BEGIN expected
- : expected
- Invalid array bound or CASE label
- Identifier already declared
- Unknown identifier
- Two many constants or identifiers

- Unknown type. Only REAL and STRING types are supported
- READ and WRITE procedures require an argument list
- ) expected
- ( expected
- Address table overflow
- Only INP can be specified for EOF function
- UNTIL expected
- Invalid REAL constant value
- Expression operands are of different types
- Left and right side of assignment statement are of different types
- Number of arguments or the type of one or more arguments do not match procedure or function declaration
- Unknown P-code instruction (indicates an error in the compiler)
- OF expected
- Invalid or missing array bound specification. Array bounds must be enclosed in []
- Array bounds not separated by ..
- OF missing in array declaration
- Invalid array type. Only REAL arrays are supported
- Invalid array bounds. Lower bound must be 1 and upper bound must be greater than lower bound
- [ expected
- Variable is not an array
- Array index must be a REAL expression
- ] expected
- Boolean operator expected (AND, OR, NOT)
- FOR variable cannot be an array or a STRING variable
- TO or DOWNTO expected
- REAL expression expected
- Too many procedures or functions
- FOR variable must be a local variable
- Invalid PROGRAM attribute

#### E. Run-time errors

- Stack overflow
- Invalid stack pointer (\*)
- Unknown P-code instruction (\*)
- Invalid procedure call (\*)
- Program too large
- Program had compilation errors
- Unknown pre-defined procedure call (\*)
- Invalid component name in POSTING function call (valid component names are: MFN, TAG, OCC, CNT)
- Too many STRING variables
- Invalid file name in ASSIGN procedure (valid file names are: INP, OUT)
- Reading past end-of-file on either INPUT or INP
- Maximum (16) procedure nesting level exceeded
- UPDATE procedure called without prior read (RECORD or NEWREC)
- Invalid parameter in PATH function call (check that the first parameter is SYS or DBN, and that the second parameter is within the corresponding range)

Note that errors marked with (\*) indicate a malfunction of either the compiler or the interpreter.

## Section 2

### *Language Specifications*

This chapter is not intended to provide a comprehensive description of the Pascal language, but only to indicate the restrictions of CDS/ISIS Pascal with respect to standard Pascal, or its deviations therefrom. For a complete language description you should refer to an appropriate Pascal programming manual.

#### A. Constants, Types and Variables

**CONST** and **TYPE** declarations are not supported.

Only three predefined types may be used:

**REAL**            Real numeric values. Constants of type real may be integers or decimal numbers optionally signed. Exponent notation is not supported.

**ARRAY [1..n] OF REAL**

Array bounds must be integer constants and the lower bound must be 1. Array constants or arrays of string are not supported.

**STRING**        Variable length character strings. Null string variables and constants (e.g. '') are supported.

All variables must be declared in a **VAR** clause before they are used.

#### B. Expressions

Arithmetic, boolean and string expressions follow standard PASCAL syntax (note that boolean expressions are supported in conditional statements although **BOOLEAN** type variables cannot be declared).

The following operators are supported:

<b>Unary</b>	+ -
<b>Addition</b>	+
<b>Subtraction</b>	-
<b>Multiplication</b>	*
<b>Division</b>	/
<b>Concatenation</b>	(string operator)
<b>Boolean</b>	AND OR NOT
<b>Relational</b>	= <> <= >= < > (note that in order to compare as equal, two strings must have the same length)

### C. Statements

The following statements are implemented (unless a specific restriction is mentioned, all statements implemented follow standard PASCAL syntax):

**Assignment statement:** The left and right side of the assignment statement must be of the same type.

#### IF statement

**CASE statement:** The CASE selector can be either a REAL or a STRING expression. When string expressions are used the corresponding case labels may be strings or single characters. However, in order to be matched properly, the expression must yield a string which has the same length as the case label constants. Multiple case label constants are not supported.

**FOR statement:** The control variable must be a local variable. It cannot be a parameter or a component of an array.

#### WHILE statement

#### REPEAT ... UNTIL statement

### D. Procedures and Functions

Both REAL and STRING functions are supported, ARRAY functions are not supported.

All REAL procedure and function parameters are passed by *value*, whereas STRING parameters are passed by *reference*. The VAR parameter prefix, however, should not be declared. Although the compiler does not issue a message if an assignment is made to a formal parameter passed by value, the value of the corresponding actual parameter in the calling procedure will not be affected.

Likewise, the compiler does not check whether the actual parameter, corresponding to a formal parameter passed by reference, is an expression and the procedure assigns a value to it. It is your responsibility to ensure that whenever a procedure assigns a value to a formal parameter passed by reference, the corresponding actual parameter is always a single variable.

For example:

```
Procedure A;  
  Var x: Real;  
      c: String;  
  
Procedure B(n: Real; s: string);  
  Begin  
    n:=5;  
    s:=s|'.';  
  End;
```

```
Begin
x:=1; c:='Water';
B(x,c);
{ the value of x at this point is still 1 }
{ while the value of c is 'Water.' }
End;
```

Also note that the compiler does not check whether a function is actually assigned a value within the function body. If a value is not assigned before the function terminates the returned value is unpredictable.

External procedures and functions are not supported.

### E. Scope of Identifiers

The scope of a variable or function identifier is defined in the same way as in standard PASCAL (e.g. a variable declared in a procedure is local to that procedure and is unknown in any outer procedures).

### F. TEXT files

In addition to the standard INPUT and OUTPUT files, two additional predefined text files may be used: INP (for input) and OUT (for output). INPUT and OUTPUT are always assigned to the standard input and output devices. A RESET or REWRITE statement is automatically issued for all these files. INP and OUT are assigned by default to the standard input and output device respectively, but may be reassigned through the predefined ASSIGN procedure (see below).

The standard procedures READ, READLN, WRITE, WRITELN have the same syntax as standard PASCAL. In WRITE and WRITELN statements the fieldwidth option is only supported for REAL expressions. In this case the following rules apply:

```
WRITE[LN] ( [OUT,] exp1 [: exp2 [: exp3]] );
```

where: *exp1* is the value to be written, *exp2* is the fieldwidth and *exp3* is the precision.

If only *exp2* is given the value is first converted to an integer (the value written is in fact TRUNC(*exp1*):*exp2*). If both *exp2* and *exp3* are given the value is written in decimal notation. If neither *exp2* or *exp3* are present, the value is written in scientific exponent notation.

The EOF function may be used on the standard input or the INP file. As in standard Pascal it returns a boolean value and its syntax is as follows:

```
EOF [(INP)]
```

The EOLN function is not supported.

## G. Comments

Comments are supported and must be delimited by braces { }. The alternate delimiters (\* \*) are not supported.

## H. Designing and integrating user exits

A user exit is designed for a specific CDS/ISIS function, which must be declared as an attribute in the PROGRAM statement. Unlike batch programs, user exits may declare program-level parameters which will be passed to the program by CDS/ISIS whenever the exit is activated. These parameters are fixed for each type of user exit and are checked by the compiler.

User exits may be written for the CDS/ISIS functions described below.

### 1. MENU Exits

A MENU exit is activated each time you select a specific option from a system menu. A menu option can be associated with a menu exit by assigning to that option action code E (see the section "The Menu editor" in the *CDS/ISIS Reference Manual*).

MENU exits have one parameter and are declared as follows:

```
Program xxxxx (s1: string) [MENU];
```

The parameter **s1** is both an input and an output parameter:

**on entry:** **s1** is a one-character string containing the option identifier which caused the program to be called;

**on exit:** you must set **s1** to one of the following:

**blank**           to cause the current menu to be redisplayed;

**s**                to cause the automatic selection of option **s** from the current menu (the menu is not redisplayed and control is passed directly to the calling program as if you had selected option **s** from the menu); in this case **s** must be the *external* identifier (i.e. the one actually displayed on the menu) of a valid option of the current menu. If **s** is not a valid option a beep is produced and the menu is displayed. Note that **s** must be different from than the one passed in **s1**, otherwise you will produce an infinite loop;

**.s**               to cause the direct execution of option **s**; in this case **s** must be the *internal* identifier of a valid option of the current menu. This is useful whenever you want to do some preliminary work *before* executing the standard option (see the example given below).

Note that if your program *autotypes* a string of characters (see AUTOTYPE procedure below), the action indicated by the output parameter *s1* is executed *before* the autotyped string.

The following example illustrates how you could control data entry access to the CDS data base by means of a password.

```
Program CHKPWD(s: string) [menu];

var dbname: string;

Function CHECK: string;
var cnt: real;
    pw: string;
begin
clear;
cursor(10,10); write('Enter password ');
cnt:=0;
repeat
attr(' ',5,10,25,6); readln(pw);
cnt:=cnt+1; { count number of trials }
if pw<>'x12y' then
begin cursor(12,25); writeIn(chr(7),'Invalid password'); end;
until (pw='x12y') or (cnt>2); { allow 2 trials at most }
if pw<>'x12y'
then begin check:=' '; pw:=inkey; end
else check:='.E';
end;

begin
dbname:=dbn;
if dbname='' then { force user to select data base if none currently
selected }
begin
clearmsg; write('Data base name? '); readln(dbname);
if dbname<>' ' then open(dbname);
end;
if dbn=''
then s:=' ' { if no data base selected redisplay menu }
else if dbn='CDS'
then s:=check { if CDS data base selected then check pass-
word }
else s:='.E'; { ok for all other data bases }
end.
```

When the above menu exit is associated with option **E** of the main services menu **xXISI**, it will request the password **x12y** each time the CDS data base is selected for data entry.

## 2. FORMAT Exits

A **FORMAT** exit is activated while processing a CDS/ISIS display format. It may be used to provide data to the standard formatting routine (see the section "The Formatting Language" in the *CDS/ISIS Reference Manual*). A **FORMAT** exit is invoked by coding the following in your format:

**&name(format)**

where:

- &** identifies this as a Format exit invocation;
- name** is the name of the CDS/ISIS Pascal program to be executed; and
- format** is a CDS/ISIS format.

**FORMAT** exits have four parameters and are declared as follows:

```
Program xxxx (s1: string; lw,occ: real; s2: string) [FORMAT];
```

where:

- xxxx** is the name of the program (which you will then use to invoke the exit from the format as mentioned above);
- s1** is the result of the execution of the format parameter supplied when the exit is invoked; it may be used to provide one or more parameters to your program and is helpful in designing generalized format exits; the maximum length of this string is 255 characters;
- lw** is the line width which is in effect when the exit is executed; it may be useful, in some cases, to know the line width if your exit outputs fixed length data;
- occ** is the occurrence counter. This is zero if the exit is invoked *outside* a repeatable group and is greater than zero if the exit is invoked from *inside* a repeatable group. In this case, CDS/ISIS will repeatedly call your exit for each execution of the repeatable group, the first time with **occ**=1, the second time with **occ**=2, etc. until you return an empty string (in **s2**) to signify that there is no more data to be output.
- s2** is the output data to be returned to the CDS/ISIS formatting program. CDS/ISIS will handle this data as if it was a field of the record being formatted. Note that the length of **s2** is not restricted.

If your format exit is designed to handle repeatable fields, you should check the **occ** parameter to know whether all the occurrences should be handled at once (if **occ**=0, your exit will only be called once), or one at the time. Alternatively, you may put restrictions on the use of the exit, e.g. decide that it must always be used within a repeatable group. For example, the following format exit can be used

within a repeatable group to number the occurrences of a repeatable field (the tag of the field to be numbered is passed as parameter):

```
Program NUMBER(s1: string; lw,occ: real; s2: string) [FORMAT];
begin
  if nocc(val(s1))>=occ
    then s2:=encint(occ,3)|'. '
    else s2:='';
end.
```

An example of usage of the above format exit (to number the occurrences of field 70) is given below:

```
Format :          (&number('70'),v70/)
                |          |
                |          |
                |          +----+
                |          |
                V          V
Output :         1. Jones, P.
                2. Brown, J.
                3. McKenzie, K.
```



## Section 3

### *The CDS/ISIS Pascal Library*

#### A. Overview

The CDS/ISIS Pascal library contains a collection of predefined procedures and functions which can be classified in two broad categories: general procedures and CDS/ISIS procedures. The following pre-defined procedures and functions are currently available.

#### 1. General procedures and functions

##### a. Screen management

ATTR	Set screen attribute
BOX	Draw box
CHATTR	Change screen attribute
CLEARBOX	Clear box
CLEARDATA	Clear data area (lines 1-21)
CLEARLN	Clear line
CLEARMSG	Clear message area (lines 22-24)
CLEAR	Clear screen
CURSOR	Set cursor position
GETCC	Get cursor position (column)
GETCL	Get cursor position (line)
PAGE	Select active page
SAVESCR	Save active page

##### b. Keyboard management

AUTOTYPE	Simulated keyboard input
INKEY	Single character input (with echo)
KBDKEY	Single character input (without echo)
DEFKEY	Define function key

##### c. Type conversion

CHR	Convert real to character
ENCINT	Convert real to string (integer)
ENCREAL	Convert real to string
ORD	Convert character to real
VAL	Convert string to real

**d. String utilities**

POSITION	Find position of string
SIZE	Size of string
SUBSTR	Substring
UC	Convert to upper case

**e. Miscellaneous**

ASSIGN	Assign file name
DATESTAMP	Get date and time
FILEXIST	Check existence of file
EXEC	Execute another program

**2. CDS/ISIS procedures and functions**

**a. System**

LANG	Current language
MENU	Display system menu
MSGTEXT	Get system message
MSG	Display system message
PATH	Get path defined in SYSPAR.PAR or dbn.PAR
SETLANG	Set dialogue language

**b. Data base**

CLOSE	Close data base
DBN	Current data base name
LOCK	Lock data base (VAX only)
MAXMFN	Next MFN to be assigned
OPEN	Open data base
UNLOCK	Unlock data base (VAX only)

**c. Master file**

CREATE	Create new record (interactive)
DELETE	Delete record
FIELDN	Find Field number
FIELD	Get Field contents
FLDADD	Add field
FLDCHA	Change field
FLDDEL	Delete field
FLDMOD	Modify field
FLDREP	Replace field
MODIFY	Modify record (interactive)
NEWREC	Create new record (batch)
NFIELDS	Number of fields in record

NOCC	Number of occurrences of field
RECORD	Get master file record
UPDATE	Update record

**d. Inverted file**

DELTERM	Delete dictionary term
FIND	Get dictionary term
NXTPOST	Get next posting
NXTTERM	Get next dictionary term
POSTING	Get posting component
UPDIF	Update Inverted file

**e. Searching**

NXTPOS	Get MFN of next record retrieved
SEARCH	Execute a CDS/ISIS search expression
SETPOS	Get MFN of retrieved record

**f. Formatting**

FMTNAME	Current format name
FORMAT	Execute display format
GETFMT	Define display format
LINES	Number of lines produced by FORMAT
NXTLINE	Get next line produced by FORMAT

**g. Editing and data entry**

DATAENTRY	Edit a worksheet page
DUMMYREC	Establish a dummy record
EDIT	Edit a string through the CDS/ISIS field editor
WORKSHEET	Select worksheet

A detailed description of all pre-defined procedures and functions is given below (in alphabetical order). Note that VAR parameters are supported for pre-defined functions and procedures.

**B. Functional description**

**1. Procedure ASSIGN (s1, s2: string);**

Assigns the standard file INP or OUT to a user-defined file or device.

s1 is a string expression equal to either INP or OUT

**s2** is a string expression yielding either a valid file name for the operating system being used (which may include drive and/or directory information) or a standard device name (e.g. PRN).

Reassigning a file already assigned is allowed. The current file is automatically closed. In some cases a dummy ASSIGN may be required to force the closing of OUT before reassigning this file to INP. A file assigned to INP must already exist. If in doubt, you should first check its existence by means of the FILEXIST function. Furthermore, you may obtain path information by means of the PATH function for files where this may be dependent on SYSPAR.PAR or dbn.PAR.

## 2. Procedure ATTR (**fill: string; attr, lin, col, len: real**);

Clears the screen area **len** characters long, beginning at line **lin**, column **col**. The screen attribute of the cleared area is set to **attr** and the area is filled with the first character of **fill**. The values of **attr** can be any of the standard CDS/ISIS attributes:

-2	Screen background	2	Bold
-1	Message area	3	Underlined
0	Normal	4	Blinking
1	Reverse video	5	Invisible

The actual attribute will depend on the values set for CDS/ISIS using option **A** of the system utility services menu xXM1.

ATTR will also position the cursor at line **lin**, column **col**.

## 3. Procedure AUTOTYPE (**s1: string**);

Inserts string **s1** into the internal keyboard input buffer. All subsequent calls to INKEY or KBDKEY will obtain input from this buffer so long as there are characters available. When the internal input buffer is empty, normal input from the keyboard is resumed.

If the internal buffer already contains one or more characters, **s1** will be inserted *before* these. Therefore you should not normally use AUTOTYPE more than once in a given program. If the characters to be autotyped are collected at various points of your program, use a string to hold them until the final sequence is obtained, then autotype this string before exiting.

If you use AUTOTYPE in a menu exit, note that the action indicated by the program parameter is executed *before* the autotyped string.

**s1** may contain encoded scan codes and/or control characters to identify keys which are not producing an ASCII code (e.g. F1, F2, etc.). Scan codes and control characters are encoded using the same rules for key definition parameters in SYSPAR.PAR (see the section "System installation" in the *CDS/ISIS Reference Manual*).

This procedure is particularly useful when you want to submit a query for processing and control to be returned to your program without any user intervention. For example:

```
autotype(' ');  
s:=search('adult * education');  
x:=inkey;
```

Normally, after executing a search, CDS/ISIS pauses to let the operator read the results. This pause is in fact an INKEY. As soon as you type a character, CDS/ISIS will in turn "autotype" it again (this allows you to enter a valid menu option, which will then execute immediately without redisplaying the menu). In the example above, there will be no pause after executing the search 'adult \* education' (because of the preceding AUTOTYPE). The subsequent INKEY is necessary to delete the autotyped ' ' from the internal buffer.

**4. Procedure BOX (lin, col, h, w, frame: real);**

Draws a rectangular box frame, **h** lines high and **w** characters wide, starting at line **lin**, column **col**. The frame is drawn with a single line (if **frame**=1) or a double line (if **frame**=2).

**VAX implementation note:** On VAX terminals, frames are implemented through the alternate (G0) character set. For single line frames the graphics set is designated [ESC(0)], and for double line frames the alternate graphics set is designated [ESC(2)]. As the latter is normally an optional feature, you should check whether it is actually supported by your terminal before using it.

**5. Procedure CHATTR (attr, lin, col, len: real);**

Changes the attribute of the screen area starting at line **lin**, column **col**, **len** characters long to **attr**. The text which may already be present on the screen in this area is not cleared. The values of **attr** are the same as for the procedure ATTR.

**Portability note:** This procedure is not available on VAX or WANG systems. If used on these computers, it will be ignored.

**6. Function CHR (n: real): string;**

This function returns a one-character string containing the character whose ASCII code is **n**. For example, CHR(65) returns 'A'. Note that **n** should be in the range 0 - 255; if not, the returned character is CHR(**n** modulo 256).

**7. Procedure CLEAR;**

Clear the entire screen. The screen attribute is set to -2 (screen background) and the cursor is positioned at line 1, column 1.

**8. Procedure CLEARBOX (lin, col, h, w, attr: real);**

Clears a rectangular box (**lin, col, h, w** as for procedure BOX). The area cleared is set to attribute **attr**. The values of **attr** are the same as for the procedure ATTR.

**9. Procedure CLEARDATA;**

Clears the screen data area (lines 1 to 21). The cleared area is set to attribute -2 (screen background) and the cursor is positioned to line 1, column 1.

**10. Procedure CLEARLN;**

Clears the current line starting from the cursor position.

**11. Procedure CLEARMSG;**

Clears the screen message area (lines 22 to 23). The cleared area is set to attribute -1 (message area attribute) and the cursor is positioned to line 22, column 1.

**12. Procedure CLOSE;**

Closes (and unlocks if necessary) the current data base, if any.

**13. Function CREATE: real;**

Performs the same function as the data entry option N of the data entry services menu xXE1 (creation of new record). The function value is the MFN of the created record.

In response to CREATE CDS/ISIS selects the current worksheet and displays it on the screen. Normal data entry proceeds until the operator exits. The record created is not available for processing after CREATE. If needed you must re-read the record using the RECORD function. If you want to interact with the operator while creating a record use the DATAENTRY function.

**14. Procedure CURSOR (lin, col: real);**

Positions the cursor at line **lin**, column **col**. Note that ATTR, CHATTR, CLEAR, CLEARDATA, CLEARMSG also position the cursor.

**15. Function DATAENTRY (var s1: string): real;**

This function allows you to control the creation and/or the editing of a record and interact with the operator. Before you use this function you must ensure that

a record is available. You do this by using either the RECORD function, if you are editing an existing record, or the NEWREC function, if you are creating a record. For record editing CDS/ISIS will use the first page of the currently selected worksheet (you may define the the worksheet to be used with the WORKSHEET function if required).

DATAENTRY edits one page at the time. If you need a multi-page worksheet, you must define each page as a separate worksheet and select the appropriate page before calling DATAENTRY. Because you gain control after each page, you may select the next page depending on the contents of the record, as in the following example:

```
rc:=record(n); { read record n }
if rc=0 then
  begin
    rc:=worksheet('a'); { select worksheet a }
    if rc=0 then
      begin
        rc:=dataentry(x); { perform data entry }
        if rc<>0 then check else
          case x of { check exit option }
            ' ': begin
                  t:=field(fieldn(10,1)); uc(t);
                  case t of { select next worksheet depending on field 10 }
                    'M': rc:=worksheet('b');
                    'S': rc:=worksheet('c');
                    'C': rc:=worksheet('d');
                  end;
                end;
            ....
          if (x=' ') and (rc=0) then rc:=dataentry(x);
          if (position(' EXN',x,1)>0) and (rc=0) then update else ckeck;
          end else writeln('Worksheet not found');
        end else writeln('Record not found');
```

In this example, record editing starts with worksheet A, then continues with worksheet B, C or D depending on the contents of field 10 (which could be, for example a type of record indicator).

Note that you may also perform field validation after editing a page.

On exit the function value will be set as follows:

- 0 Editing was successful
- 1 The worksheet is inadequate for editing the record (a field exceeds the length specified in the worksheet)
- 2 The editing caused the maximum record length to be exceeded
- 3 The worksheet was not found

If an error condition is reported the record is left in its original state.

**s1** will contain the exit option selected by the operator after editing the page. It is your responsibility to honor this request. The exit options which may be reported are listed below:

- blank** Next page (if any) or exit and update record
- B** Preceding page
- C** Cancel (exit without updating the record)
- T** Cancel and terminate range (in case you are offering to edit a range of record)
- E** Exit and update record (no further pages, if any should be displayed)
- D** Delete record
- N** Exit and update record, then create a new record

If one of the exit options requests record update, you should do so by using the UPDATE procedure. Note that the "blank" exit option may imply update if this was the last page of the worksheet.

While editing a record with this method you *must not access any other record*, as doing so will cause the changes done to the current record to be lost. You may, however, access the inverted file, if needed.

#### 16. Function DATESTAMP: string;

This function returns an 18-character string containing the (operating system) date and time of day, in the following format:

**MM-DD-YY HH:MM:SS**

where: **MM** is the month, **DD** the day, and **YY** the year; and **HH** the hour, **MM** the minutes, and **SS** the seconds (note that there are two spaces between the date and the time).

Because this function interfaces directly with the operating system, its accuracy depends on the proper setting of the operating system date and time. In particular, on those computers which do not automatically preserve the date and time when turned off, it is essential that these be reset manually to their correct value each time the computer is turned on.

#### 17. Function DBN: string;

Returns the name of the currently selected data base or a null string if none.

**18. Procedure DEFKEY (k: real; s: string);**

Defines *s* as the value of function key *k*. *k* is the scan code of the key being defined. This procedure is equivalent to defining a function key in SYSPAR.PAR (see the section "System installation" in the *CDS/ISIS Reference Manual*) For example:

```
defkey(67,substr(datestamp,1,8));
```

will cause function key <F9> (scan code 67) to autotype today's date.

**19. Procedure DELETE;**

Marks the current record as logically deleted (the current record is the last record read through the RECORD function or established through the NEWREC function). A call to DELETE without a prior RECORD or NEWREC is illegal and will cause program termination.

After DELETE the current record is no longer available.

**20. Function DELTERM (t: string): real;**

Deletes term *t* from the inverted file dictionary. Note that you may only delete those terms which have currently no postings.

The value returned is 0 if the operation was successfully performed and different from zero if it could not be performed (e.g. attempting to delete a term having one or more postings).

**21. Procedure DUMMYREC;**

Like NEWREC, it establishes an empty record suitable for data entry. DUMMYREC, however, does not increase the maximum MFN of the data base and the record established cannot be written to the data base. In fact, DUMMYREC does not require a data base to be open. An attempt to use UPDATE or DELETE on a record established with DUMMYREC will produce a run-time error.

The purpose of DUMMYREC is to facilitate the collection of data from a worksheet using the CDS/ISIS data entry facilities in a manner similar to the one used by CDS/ISIS with system worksheets, such as the print worksheet.

Worksheets to be used in conjunction with DUMMYREC, unlike actual system worksheets, which are created with the ISISUTL services, must be created with the ISISDEF services. This is because the tag of the fields defined in the worksheet identifies each piece of data entered (the fields of a system worksheet do not have tags as CDS/ISIS *knows* the fields by their relative position). To define these worksheets you may use an existing data base or a data base established for this purpose. However, in order to be retrieved even when no data base is cur-

rently selected, you should assign them a name with 'Y' in the second position (this will ensure that they will be stored in the menu path (defined in parameter 2 of SYSPAR.PAR)).

Once you have established a (dummy) record through DUMMYREC, you may collect data by means of the DATAENTRY function, after selecting the appropriate worksheet. You may then retrieve the data through the FIELD function.

**22. Function EDIT (var s: string; max, line, col, wsize, attr: real; fill: string): real;**

This function allows you to edit a field using the CDS/ISIS field editor. The parameters are described below:

<b>s</b>	the string to be edited (after EDIT s will contain the resulting edited string). Note that s may be initially empty.
<b>max</b>	the maximum length (up to 255 characters) to which the string should be allowed to expand during editing
<b>line, col</b>	the line and column position of the top left corner of the edit window
<b>wsize</b>	the size of the edit window (if wsize < max then the field will scroll)
<b>attr</b>	the screen attribute of the edit window
<b>fill</b>	the fill character to be used to fill empty positions of the edit window

EDIT clears the edit window and fills it with the contents of the string to be edited. The full range of function of the CDS/ISIS field editor can be used during editing. Note also that line 24 is used for the edit mode (insert/replace) message.

On exit the function value indicates the edit termination key as follows:

<b>0</b>	Enter	Edit terminated normally
<b>1</b>	F1	Help key was pressed
<b>2</b>	<TAB>	Tab key was pressed
<b>3</b>	F2	string was cleared
<b>4</b>	<ESC>	Esc key was pressed
<b>5</b>	<PgDn>	PgDn key was pressed

It is your responsibility to interpret the meaning of the various termination options. Note, however, that if the string was modified, <ESC> returns in s the *modified* version. If you want to implement the standard meaning of this key, i.e. to restore the field to its original value and ignore any changes made to it, you must save the original value before calling EDIT.

**23. Function ENCINT (n1, n2: real): string;**

Truncates the value of **n1** to an integer and converts it to its ASCII representation with a minimum precision of **n2** digits.

**24. Function ENCREAL (n1, n2, n3: real): string;**

Converts **n1** to its ASCII representation. **n2** is the minimum field width and **n3** the number of decimal places.

**25. Procedure EXEC (s1: string);**

Executes program **s1**. Note that control is not returned to the current program. EXEC may only be used in batch programs. If used in a user exit the results are unpredictable.

**26. Function FIELD (n: real): string;**

Returns the value of the **n**-th field in the current record. Note that **n** is *not* the tag of the field, but is the *field number* of the required field obtained through the function FIELDN.

For example:

```
s:=field(fieldn(10,1));
```

will return in **s** the value of the first occurrence of field 10 or an empty string, if field 10 is not present in the record.

**27. Function FIELDN (tag, occ: real): real;**

Returns the *field number* of a given field in the current record. The field number is the ordinal number of the directory entry corresponding to the **occ**-th occurrence of the field with tag **tag**. The field number of a non-existent field is 0.

**28. Function FILEXIST (f: string): real;**

This function returns 0 if file **f** exists. **f** should be a string expression yielding a valid file name for the operating being used. You may use the PATH function whenever the file path may depend on SYSPAR.PAR or dbn.PAR. For example:

```
n:=filexist(path('sys',4)|'myfile.wrk');
```

will return 0 if the file **myfile.wrk** exists in the work path defined in parameter 4 of SYSPAR.PAR.

**29. Function FIND (var s1: string): real;**

Searches **s1** in the dictionary of the current data base and returns 0 if it exists. If the term does not exist, **s1** will contain, on exit, the next higher term or an empty string if the end of the dictionary is reached. After **FIND** you may use **NXTTERM** to retrieve dictionary terms sequentially.

**30. Function FLDADD (tag, n: real; s: string): real;**

Adds a new field to the current record. **tag** is the tag of the field and **s** is the contents of the field to be added. The field is added before the **n**-th field currently existing in the record (you may specify **n=1** if the order is not important or **n=NFIELDS+1** to add a new occurrence of a repeatable field after the last occurrence currently present).

The value returned is 0 if the operation was successfully performed or 1 if it could not be performed (e.g. no room in record).

**31. Function FLDCHA (n: real; s1, s2: string): real;**

Replaces **s1** with **s2** in the **n**-th field of the current record (**n** must be obtained with the **FIELDN** function). The value returned is 0 if the operation was successfully performed or 1 if it could not be performed (e.g. **s1** was not found).

**32. Function FLDDDEL (n: real): real;**

Deletes the **n**-th field from the current record (**n** must be obtained with the **FIELDN** function). The value returned is 0 if the operation was successfully performed or 1 if it could not be performed (e.g. the field does not exist).

**33. Function FLDMOD (n1, n2, n3: real; s: string): real;**

Deletes the part of the **n1**-th field in the current record (**n1** must be obtained with the **FIELDN** function) starting at position **n2** of length **n3** and replaces it with **s1**. The value returned is 0 if the operation was successfully performed or 1 if it could not be performed (e.g. the field does not exist). An example is given below:

```
rc:=fldmod(fieldn(10,1),5,8,'xxx');
```

Contents of field 10

before fldmod

after fldmod

International Corporations

Intexxxl Corporation

**34. Function FLDREP (n: real; s: string): real;**

Replaces the contents of the **n**-th field in the current record with **s** (**n** must be obtained with the **FIELDN** function). The value returned is 0 if the operation was successfully performed or 1 if it could not be performed (e.g. the field does not exist).

**35. Function FMTNAME : string;**

This function returns the name of the currently selected display format. The following special names may also be returned:

- ” a null string indicates that no format is available (e.g. if no data base was yet opened)
- \* the current format was defined by the user during the current session (it does not therefore exist as a .PFT file)
- ALL the current format is the built-in format

**36. Function FORMAT (lw: real): real;**

Formats the current record according to the currently selected format using a line width of **lw** characters. The value returned is 0 if the operation was successfully performed or a formatting error code if a format error was found (the code returned is one of those listed in the section "The formatting language" in the *CDS/ISIS Reference Manual*).

Note that this function simply executes the format: the lines produced are stored in a work area. You may use the **LINES** function to determine how many lines were produced and the **NXTLINE** function to obtain the contents of each line.

**37. Function GETCC : real;**

Returns the current cursor column position.

**38. Function GETCL : real;**

Returns the current cursor line position.

**39. Procedure GETFMT (s: string);**

Defines the current display format. **s** may be either an actual format in the CDS/ISIS formatting language or the name of a predefine format. In the latter case the name must be preceded by an @ sign. For example:

```
GETFMT('v24/v56(0,4)');  
GETFMT('@xxx');
```

Note that the format defined with this procedure replaces the *current* format and remains in effect even after your program terminates. You may use the FMTNAME function to save the name of the current format, so that you may restore it before exiting, if required.

**40. Function INKEY: string;**

Returns the next character typed from the keyboard (or available in the internal AUTOTYPE buffer). If the character is actually typed from the keyboard, it is echoed to the screen, whereas autotyped characters are never echoed.

The string returned is always a 1-character string. Note that the <CR> key returns a space. After this function the cursor is always positioned at column 1 of the next line.

**41. Function KBDKEY (var c: string): real;**

Returns in *c* the next character typed from the keyboard (or available in the internal AUTOTYPE buffer). If *c* is an ASCII character, the function value is 0, otherwise it is the *scan code* of the key which was pressed or autotyped. In most cases, if the function value is greater than 0 then *c* is set equal to CHR(0). In some cases both the function value and ORD(*c*) are greater than zero. Some examples are given below:

Key pressed	value of <i>c</i>	function value
A	A	0
<F2>	CHR(0)	60
<ESC>	CHR(27)	1

You may use the following sample program to determine the exact values returned by this function (press x to exit from the program):

```
program SCANC;  
  
var c: string;  
    sc: real;  
  
begin  
  repeat  
    write('Press a key ');  
    sc:=kbdkey(c);  
    writeln('Scan code: ',sc:3,' ASCII code: ',ord(c):3,' char: ',c);  
  until (c='x') or (c='X');  
end.
```

**42. Function LANG: string;**

Returns a one-character string containing the currently selected language code.

**43. Function LINES: real;**

Returns the number of lines produced by the function FORMAT. Note that the execution time of this function is proportional to the number of characters produced by the format. You should use it therefore only when required (see also the NXTLINE function).

**44. Function LOCK: real;**

This function requests an exclusive write lock for the currently selected data base. If the lock is granted the function value is 0. In order to successfully obtain exclusive control, there must be no other user performing an update operation on the data base at the time the lock is requested. If the lock is granted, no other user will be permitted to perform any update operation until the lock is removed. Once granted, a lock remains in effect until: (a) an UNLOCK procedure (see below) is executed on the *same* data base; (b) the data base is (implicitly or explicitly) closed; or (c) the end of the session.

You must lock a data base whenever your program performs any output or update operations on the master or the inverted file, i.e. if you execute any one of the following procedures: NEWREC, UPDATE, DELETE, CREATE, MODIFY, DELTERM or UPDIF. An attempt to write to a data base without first issuing LOCK (or ignoring an unsuccessful lock request), will abnormally terminate the session.

**Portability note:** The LOCK function, as described above, is only implemented for VAX systems. On IBM PCs or WANG systems, where simultaneous write access to a data base is not supported, this function always returns 0 (i.e. write access is always granted). If your program is intended to work on both VAX and PCs then you must issue LOCK, whenever appropriate, otherwise your application will not be portable.

**45. Function MAXMFN: real;**

Returns the next MFN to be assigned in the current data base.

**46. Function MENU (s: string): string;**

Displays the system menu *s* and returns the user's selection. The string returned is always a 1-character string. The string *s* must contain the name of the menu to be displayed (not the file name), and this name must conform to the CDS/ISIS conventions for menu names, i.e. the first character must be the language code and the second character must be **X**, e.g. EXABC.

Note that CDS/ISIS will automatically select the language version of the menu corresponding to the currently set language. It is your responsibility to ensure that all the required language versions of the menu have been previously defined.

**47. Function MODIFY (n: real): real;**

Performs the same function as the **E** option of the data entry services menu **xXE1**. **n** is the MFN of the record to be edited. The value returned is 0 if the operation was correctly performed or different from 0 if it could not be performed.

**48. Procedure MSG (n: real);**

Displays the CDS/ISIS message **n** (in the currently selected language). The message is displayed starting at the current cursor position, and, after displaying the message the cursor will be positioned immediately after the message text.

If **n** does not correspond to a valid message number then CDS/ISIS will display **MSG-n [text not found]**.

**49. Function MSGTEXT (n: real): string;**

Returns the text of the CDS/ISIS message **n** (in the currently selected language). If **n** does not correspond to a valid message number, an empty string is returned.

**50. Function NEWREC: real;**

Initializes an empty new record and returns its MFN. You may then insert data in the record through the **FLDADD** function and write it to the master file using the **UPDATE** procedure.

The record established through **NEWREC** becomes the *current* record, i.e. all record-oriented procedures and functions subsequently executed, up to and including **UPDATE** or **DELETE**, will operate on this record.

**51. Function NFIELDS: real;**

Returns the number of fields in the current record. Note that *each* occurrence of a repeatable field is counted as a field. For example, if the record contains one occurrence of field 10 and 20 and 3 occurrences of field 30, then **NFIELDS** returns 5.

**52. Function NOCC (n: real): real;**

Returns the number of occurrences of the field with tag **n** in the current record (or 0 if the field does not exist).

### 53. Function NXTLINE (var lin: string): real;

Returns in **lin** the next line produced by the FORMAT function. The function value is 0 if a line is available in **lin**. A value different from zero is returned if no more lines are available. When called immediately after FORMAT, NXTLINE returns the first line; each subsequent call will return the next line. The function value should be checked to determine if **lin** contains valid data.

The following example shows the use of this function:

```
var rc,mfn,lrc,frc: real;
    lin: string;
- - - - -
rc:=record(mfn);
if rc=0 then
begin
  frc:=format(79);
  if frc=0 then
  begin
    lrc:=nxtline(lin);
    while lrc=0 do begin writeln(lin); lrc:=nxtline(lin); end;
  end
else writeln('Format error ',frc:1);
end;
```

Note that the following would produced the same results, although it would be slightly less efficient, especially if the format produced many lines (because of the LINES function):

```
var rc,mfn,lrc,frc,i: real;
    lin: string;
- - - - -
rc:=record(mfn);
if rc=0 then
begin
  frc:=format(79);
  if frc=0 then
    for i:=1 to lines do begin lrc:=nxtline(lin); writeln(lin); end
  else writeln('Format error ',frc:1);
end;
```

### 54. Function NXTPOS (n: real): real;

Returns the MFN of the next **n**-th record retrieved relative to the current record. NXTPOS may be used, after SETPOS or a previous NXTPOS, to obtain the MFN of the next record in the current set. The current set is determined by the last SETPOS issued. **n** may be greater than 1. For example NXTPOS(3) will provide the MFN of the 3rd record retrieved after the current one. Negative values of **n** are not supported.

NXTPOS returns 0 when no (or no more) records are available.

**55. Function NXTPOST: real;**

After FIND or NXTTERM, positions to the next posting for the term. The posting may subsequently be analysed with the POSTING function. A negative value is returned if no more postings are available.

**56. Function NXTTERM: string;**

Returns the next dictionary term. NXTTERM may be used after FIND or another NXTTERM to read the dictionary sequentially. A null value is returned when no more terms are available (i.e. the end of the dictionary is reached).

Using NXTTERM without a prior FIND produces unpredictable results.

**57. Procedure OPEN (s: string);**

Opens (i.e. makes available for processing) data base *s*. After this procedure is executed the previous data base, if any, is no longer available. Executing this procedure is equivalent to selecting option C (change data base) in the main services menu xXISI. If another data base is already selected when this procedure is executed, the former will be automatically closed (and unlocked, if necessary).

**Portability note:** On VAX system, the data base is open in read-only mode. If you intend to perform write operations to the data base you are opening, you must also issue a LOCK function.

**58. Function ORD (s: string): real;**

This function returns the ASCII code of the first character of *s*. For example, the value of ORD('A') is 65.

**59. Procedure PAGE (n: real);**

Selects page *n* ( $1 \leq n \leq 4$ ) as the active page. The page must have been previously saved with SAVESCR, otherwise the result will be unpredictable.

**Portability note:** This procedure is not available on VAX or WANG systems. If used on these computers it is ignored.

**60. Function PATH (s1: string; n: real): string;**

The value of this function depends on the value of *s1*, as follows:

*s1* = 'SYS'     the value returned is the value of parameter *n* defined in SYSPAR.PAR (in this case  $1 \leq n \leq 5$ );

**s1 = 'DBN'** the value returned is the value of parameter **n** defined in dbn.PAR (in this case  $1 \leq n \leq 10$ ). Note that in this case, the value returned is reliable only if a data base is currently selected. If not, the value returned is the path defined in parameter 5 of SYSPAR.PAR.

For example, if you wanted to read the FDT of the current data base, you could use the following ASSIGN procedure:

```
Assign('inp',path('dbn',10)|dbn|.fdt');
```

**61. Function POSITION (s1,s2: string; n: real): real;**

Searches string **s2** in string **s1** starting from position **n** and returns its position, or 0 if **s2** does not occur in **s1**.

For example, **position('abcd','c',1)** returns 3.

**62. Function POSTING (s1: string): real;**

Returns the value of the current posting component indicated by **s1**. The current posting must have been previously set by NXTPOST. The component returned is indicated by **s1** as follows:

<b>MFN</b>	Master file number
<b>TAG</b>	Field identifier
<b>OCC</b>	Occurrence number
<b>CNT</b>	Sequence number

**63. Function RECORD (n: real): real;**

Retrieves the master file record with **MFN = n**. The retrieved record becomes the *current record*, i.e. all record-oriented procedures and functions subsequently executed will operate on this record. The value returned indicates whether the record was successfully retrieved as follows:

-1	End of file
0	Record was retrieved
1	Record is marked for deletion
2	Record is physically deleted

No record-oriented operation should be performed if the returned value is -1. If the returned value is 1, the contents of the record at the time it was marked for deletion is available for inspection and the deletion flag is removed. If you want to leave the record marked for deletion, you should not write it back. Alternatively, you may reactivate the record by writing it back by means of the UPDATE procedure (after modifying it, if needed).

If the returned value is 2, CDS/ISIS provides you with an empty record, which you may fill with the FLDADD function and write to the master file using the UPDATE procedure. This record will retain its original MFN. If you do not write the record back it will remain deleted.

**64. Procedure SAVESCR (n: real);**

Saves the current screen in the internal buffer **n** ( $1 \leq n \leq 4$ ). The screen is not affected by this operation. A saved screen may be restored by using the PAGE procedure.

Note that a saved screen is only available for restoring within the program which saved it. You should not expect a screen saved by one program to be available to another program or to a subsequent re-execution of the program which saved it.

**Portability note:** This procedure is not available on VAX or WANG systems. If used on these computers it will be ignored.

**65. Function SEARCH (s: string): real;**

Performs a CDS/ISIS search on the currently selected data base using the search expression defined by **s** and returns the set number assigned to the search expression. Executing this function is equivalent to selecting option **S** of the ISISRET services.

Note that if the expression in **s** contains a syntax error the user may edit it interactively. If **s** is a null string (") the user is prompted to enter the search expression.

**66. Function SETLANG (s: string): real;**

Sets the current dialogue language. **s** is a one-character string containing the code of the language to be set. If you use this function, you must ensure that all necessary menus, worksheets and message files for the selected language exist.

The function returns 0 if the indicated language was successfully selected, or 1 if the language could not be set (e.g. if no message file was defined). If the value returned is 1 the current language is undefined. You should therefore re-instate a valid language before continuing. For example:

```
Program SETLNG;
var l: string;
begin
  l:=lang;           { save current language code }
  if setlang('q')=1
    then setlang(l); { reset saved language if 'q' is undefined }
end.
```

**67. Function SETPOS (n1, n2: real): real;**

The value returned by this function depends on the value of **n2** as follows:

**n2 < > 0** Returns the MFN of the next **n2**-th record retrieved by the **n1**-th search (in this case **n1** is a set number returned by a previous SEARCH). **n2** may be greater than 1. For example SETPOS(2,3) will provide the MFN of the 3rd record retrieved by set 2. For optimum performance subsequent records should be obtained through NXTPOS. Negative values of **n2** are not supported.

**n2 = 0** Returns the number of records retrieved (hits) by the **n1**-th search.

In both cases if **n1** is set to zero the search expression considered is the last one submitted.

**68. Function SIZE (s: string): real;**

Returns the current size of string **s**.

**69. Function SUBSTR (s: string; n1, n2: real): string;**

Returns the substring of **s** starting at position **n1** with a maximum length **n2**. A null string is returned if **n1** is outside the boundaries of **s**. If **n1 + n2 - 1** is greater than SIZE(**s**) only the last **n1 + SIZE(s) - 1** characters of **s** are returned.

**70. Procedure UC (var s: string);**

Converts string **s** to upper case. The conversion is performed according to the system table ISISUC.TAB.

**71. Procedure UNLOCK;**

Removes a data base lock established by a previous call of the LOCK function. Because the LOCK function requests exclusive write control of the data base, you should remove the lock as soon as this is no longer required. Note that the UNLOCK procedure need not be issued by the same program that executed the LOCK function.

**Portability note:** The UNLOCK function is only implemented for VAX systems. On IBM PCs or WANG systems, where simultaneous write access to a data base is not supported, this procedure has no effect. However, if your program is intended to work on both VAX and PCs then you must issue UNLOCK whenever appropriate to ensure portability.

**72. Procedure UPDATE;**

Rewrites the current record to the master file (the current record is the last record read through the RECORD function or established through the NEWREC function). A call to UPDATE without a prior RECORD or NEWREC is illegal and will cause program termination.

**73. Procedure UPDIF;**

Performs an Inverted file update run (which is equivalent to selecting option U of the ISISINV services menu xXG1).

**74. Function VAL (s: string): real;**

Converts s to a REAL. If s does not contain a valid numeric value the returned value is 0.

**75. Function WORKSHEET (s: string): real;**

Selects the data entry worksheet s. All subsequent data entry operations will use this worksheet. The value returned is 0 if the worksheet was retrieved correctly, different from zero if the worksheet was not found. The string s must contain the *name* of the worksheet, not the file name. Executing this function is equivalent to selecting option W of the ISISENT services menu xXE1.

Note that, once a worksheet is selected, it will remain in effect even after the program terminates.

## Section 4

### *Sample Programs*

This section describes the sample programs supplied on the SAMPLES diskette. We advise you to actually run the programs (they can all be run using option A of the main menu) as you study the listing.

#### A. Program KEYB

This (batch) program draws on the screen the numeric keypad of the IBM PC/XT keyboard and illustrates the application of the BOX and CLEARBOX procedures.

To simplify the algorithm, the program assumes that there are 20 keys numbered as follows:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

however, as keys 16 and 17 are long keys they are handled in a special way.

The arrays **l** and **c** contain the line/column position of the top left corner of each key. These are initialized based on the values of **ll** and **cl** which determine the position of the drawing on the screen. The string variables **top** and **bot** contain the labels to be displayed in the middle and at the bottom of the corresponding key. Because string arrays are not supported by CDS/ISIS Pascal, they are simulated here by using a single string variable, containing fixed length substrings: **top** contains 20 4-character strings, and **bot** 20 1-character strings.

The keyboard is drawn twice, key by key using the procedure DRAWKEY: the first time with attribute 0 and the second time with attribute 1. The visual effect obtained is that, after being drawn, the keys will change colour.

Note that the procedure DRAWKEY doubles the height of key 16 and the width of key 17, and does not draw keys 18 and 20 as these correspond to the second half of the long keys 16 and 17.

Note how the correct key labels are extracted from **top** and **bot** by using the SUBSTR function.

### B. Program DISPL

This program illustrates the use of the formatting procedures. It displays, sequentially, the records of the sample CDS data base. Four fields are displayed for each record: the MFN, the title (field 24), the authors (field 70) and the keywords (field 69). Each field is enclosed in a frame. After displaying a record the program issues the prompt 'N[ext record] Q[uit]' and terminates when the reply is Q or the end of the master file is reached (rc<0).

The procedure DISPLAY clears the boxes, defines the format for each field and then displays it using the procedure WRT.

### C. Program TEXT

This program performs a simple free text search on a data base. You are first asked to enter the tag of the field, then the character string to be searched. The searching is done using the POSITION function. As both the string to be searched and the contents of the field are translated to upper case before being matched, the search is case insensitive. As a record matches the search it is displayed and you may then continue or exit. The program pauses every 100 consecutive records not satisfying the search criteria.

### D. Program THES

This program provides a complete (monolingual) Thesaurus management facility. It is designed as a menu exit (but may also be run as a stand alone program). It assumes that a Thesaurus data base called THES exists.

The following global variables are used:

<b>maxt</b>	the maximum number of relations defined
<b>rel</b>	a string containing the (3-character) name of the relations
<b>invrel</b>	a string containing the (3-character) name of the inverse relations
<b>tag</b>	an array containing the tag of each relation
<b>term</b>	the current thesaurus term being processed
<b>q</b>	holds search terms selected through option Q (see below)

Submenus and prompts are displayed in the message area. The primary program options are described below:

**T** allows to select a term: an empty box is displayed at the top of the screen where the term should be entered; if it exists it is displayed, otherwise option **L** is automatically selected; the program terminates if no term is selected;

- L** displays the list of thesaurus terms in alphabetical order starting from the current term; you may page forward by pressing **P**, move from term to term with **Enter** (forwards) or **B** (backwards) , or select options **S**, **C**, **T** or **X**;
- S** selects a given term and displays the corresponding record (see the description of procedure **FINDTERM**);
- A** adds a relation to a term (see **ADDREL**);
- C** creates a new thesaurus term (see **CREATERM**);
- D** depending on the cursor position, deletes a term (see **DELTRM**) or a relation (see **DELREL**);
- Q** selects a term for searching; if other terms were previously selected, a '+' (logical **OR**) operator is inserted;
- ?** displays the current set of search terms selected with option **Q**, if any; these are displayed in a pop-up window;
- X** exits; if search terms were selected the search expression is displayed in edit mode, then executed, provided a data base was selected when **THES** was called.

The program uses a number of internal procedures and functions:

**FUC**: converts a string to upper case;

**ERRMSG**: beeps and displays an error message in the message area, then waits for a key to be pressed;

**DISPLT**: clears the data area and displays the term box; if called from option **S** it also displays the MFN of the term; the screen is saved in buffer 1;

**DISPLAY**: displays up to **maxl** relations of a term starting from a given tag and occurrence; the tag index and occurrence for each relation displayed is recorded in the arrays **dt** and **doc** respectively, and **nl** is set to the number of relations displayed;

**DECIDE**: allows to view the relations of a term and select one of the program options described above; if the selected option is **S**, the selected term is stored in **term**;

**FINDTERM**: searches a term and displays if it exists; sets **mfn** to the MFN of the corresponding record and sets **action** to the option selected through **DECIDE**;

**FLDUC**: returns a given field of the record converted to upper case;

**CHKREL**: checks whether a relation already exists;

**UPDINVF**: updates the Inverted file;

**CREATERM**: creates a new thesaurus term (provided it does not already exist); a new record is created and the inverted file updated immediately;

**ADDREL:** adds a relation to a term; it checks that the related term is already defined in the thesaurus and that a relation to the same term does not already exist; the inverse relation, if necessary is also added to the record corresponding to the related term;

**DELREL:** deletes a relation and, if necessary the corresponding inverse relation;

**DELTRM:** deletes a term, provided it has no relations; the inverted file is also updated;

**SHOWDICT:** displays the alphabetical list of thesaurus terms and allows to select one of the options **C**, **S**, **T** or **X**; you may page forward by pressing **P**, or move from term to term with **Enter** (forwards) or **B** (backwards).

```
Program KEYB;

{ Draws the numeric keypad of the IBM PC keyboard }

var l: array[1..20] of real;
    c: array[1..20] of real;
    ll,c1,i,j: real;
    top,bot: string;

Procedure DRAWKEY(i,a: real);

{ Draws a key with i=key number, a=attribute }

Var h,w: real;

Begin
  if (i<>18) and (i<>20) then
    begin
      if i=16
        then begin h:=6; w:=6; end
        else if i=17
              then begin h:=3; w:=12; end
              else begin h:=3; w:=6; end;
      if a<>0 then clearbox(l[i],c[i],h,w,a);
      box(l[i],c[i],h,w,1);
      cursor(l[i]+1,c[i]+1); write(substr(top,(i-1)*4+1,4));
      if substr(bot,i,1)<>' ' then
        begin cursor(l[i]+2,c[i]+1); write(substr(bot,i,1)); end;
    end;
end;

Begin
top:='Esc Num ScrLSysRHome ^ PgUpPrtS <- -> End v PgDn Ins Del';
bot:=' 789*456-123+0 . ' ;
ll:=3; c1:=30;
clear;
for i:=1 to 5 do for j:=1 to 4 do l[(i-1)*4+j]:=ll+(i-1)*3;
for i:=1 to 5 do for j:=1 to 4 do c[(i-1)*4+j]:=c1+(j-1)*6;

for i:=1 to 20 do drawkey(i,0);

for i:=1 to 20 do
begin
  drawkey(i,1);
  for j:=1 to 200 do; { wait loop }
end;

cursor(22,1);

end.
```

```
Program DISPL;
var i,j,k,n,rc,l: real;
    f,lin: string;

Procedure WRT (i,nl,lw: real);
var l,ll,rc: real;
begin
rc:=format(lw);
rc:=nxtline(lin); l:=i;
ll:=l+nl-1;
while rc=0 do
begin
cursor(l,2);
if l>ll then rc:=1
else begin writeln(lin); rc:=nxtline(lin); end;
l:=l+1;
end;
end;

Procedure DISPLAY;
Begin
cursor(2,74); write(i:4);

clearbox(2,2,1,28,2);
getfmt('md1,(v70/)' );
wrt(2,1,28);

clearbox(5,2,4,78,1);
getfmt('md1,v24/');
wrt(5,4,78);

clearbox(11,2,2,78,1);
getfmt('md1,v70+|; |/' );
wrt(11,2,78);

clearbox(15,2,6,78,1);
getfmt('md1,v69/');
wrt(15,6,78);

End;

begin
open('cds');
i:=0;
clear;
box(1,1,3,30,2); box(1,71,3,10,1);
box(4,1,6,80,1);
cursor(4,1);writeln('Title: ');
box(10,1,4,80,1);
cursor(10,1);writeln('Authors: ');
box(14,1,8,80,2);
```

```
cursor(14,1);writeln('Keywords: ');  
  
repeat  
  i:=i+1;  
  rc:=record(i);  
  if rc=0 then { Record exists }  
  begin  
    Display;  
    cursor(23,1); write('N[ext record] Q[uit]'); f:=inkey; uc(f);  
  end;  
  
until (f='Q') or (rc<0);  
end.
```

Program TEXT;

{ Performs a simple text search }

```
var tc,tag,rc,n,i,o,j,k,tot,nr,y: real;
    dbname, fld,s,r: string;

Begin
clear;
dbname:=dbn;
if dbname='' then
begin
msg(1); readln(dbname);
open(dbname);
end;
write('Data base name: ',dbname);
cursor(2,1); write('Tag to be searched? ');
cursor(3,1); write('String to be searched? ');

repeat
box(5,2,16,78,2);
r:='';
attr(' ',1,2,24,5);
readln(s);
if s<>'' then
begin
tag:=val(s);
attr(' ',1,3,24,50); readln(s);
uc(s);
i:=0; y:=0; nr:=0; tot:=0; tc:=0;
cursor(22,1); writeln('Records retrieved: ');
repeat
i:=i+1; rc:=record(i);
tc:=tc+1;
cursor(5,5); writeln(i:1);
if rc=0 then
begin
nr:=nr+1; n:=nocc(tag); o:=0;
while o<n do
begin
o:=o+1;
j:=fieldn(tag,o); fld:=field(j); uc(fld); k:=position(fld,s,1);
if k<>0 then
begin
Y:=Y+1;
k:=format(70); o:=6;
clearbox(6,3,14,76,1);
while (nxtline(fld)=0) and (o<19) do
begin cursor(o,5); writeln(fld); o:=o+1; end;
cursor(22,20); write(y:1,' '); r:=inkey; uc(r);
clearbox(6,3,14,76,0);
tc:=0;
end;
end;
end;
```

```
end;
end;
if tc=100 then
begin
cursor(23,1);
write('<CR> - continue for 100 more records X - Exit --> ');
r:=inkey; uc(r);
tc:=0;
end;
until (rc<0) or (r='X');
end;
until (s='') or (r='X');
end.
```

```
Program THES(option: string) [menu];

var dt: array[1..15] of real;
    doc: array[1..15] of real;
    tag: array[1..10] of real; { tag of relation }
    maxt: real; { max no. of tags (upper bound of tag) }
    maxl: real; { max no. of lines (upper bound of dt,doc) }
    rel,invrel: string; { Relation indicators }
    it,io: real; { current tag/occ }
    nl: real; { lines on this page }
    cl: real; { current line }
    term: string; { current term }
    q: string; { query }
    dbname: string; { current data base }
    mfn: real; { current mfn (in THES data base) }
    s,action,ft: string;
    i,k,kl,lq,rc: real;
```

```
Function FUC(s: string): string;
```

```
{-----}
{ Converts string s to upper case }
{-----}
```

```
var us: string;
begin
us:=s; uc(us);
fuc:=us;
end;
```

```
Procedure ERRMSG(t: string);
```

```
{-----}
{ Display error message t and pause }
{-----}
```

```
var s: string;
begin
clearmsg; writeln(chr(7),t);
write('Press ENTER to continue'); s:=inkey;
end;
```

```
Procedure DISPLT;
```

```
{-----}
{ Display top term box }
{-----}
```

```
begin
cleardata;
box(1,1,3,32,2); clearbox(2,2,1,30,2);
```

```

cursor(2,2); write(fuc(term));
if action='S' then
begin
  box(1,74,3,7,1);
  cursor(1,76); write('MFN'); cursor(2,75); write(mfn:5);
end;
savescr(1);
end;

Procedure DISPLAY(t,o: real);

{-----}
{ Display term relations starting from tag[t], occurrence o }
{-----}

var rc,fn: real;

begin
  nl:=0; it:=t; io:=o;
  if t=1 then displt else clearbox(5,1,15,80,0);
  while (it<=maxt) and (nl<maxl) do
  begin
    repeat
      fn:=fieldn(tag[it],io);
      if fn=0 then begin it:=it+1; io:=1; end;
    until (fn>0) or (it>maxt);
    if fn>0 then
    begin
      nl:=nl+1; dt[nl]:=it; doc[nl]:=io; io:=io+1;
      cursor(nl+4,1);
      write('_ ',substr(rel,(it-1)*3+1,3),' ',field(fn));
      end;
    end;
  end;
end;

Function DECIDE(l: real): string;

{-----}
{ Read action code (<CR>,B,F and P are handled here; other codes returned }
{-----}

var s: string;
    sc: real;

begin
  cl:=1;
  if nl>0 then
  begin
    clearmsg;
    writeln('+ Next B[ack] F[irst] P[age] S[elect] T[erm select] Q[query]');
    write ('?[display query] A[dd relation] D[delete] C[reate term] X[exit]');
    repeat

```

```

if c1<1 then c1:=1;
if c1>n1 then c1:=n1;
cursor(c1+4,1);
sc:=kbdkey(s); uc(s);
if s=chr(13) then s:= ' ';
case s of
' ': if c1>=n1 then c1:=1 else c1:=c1+1;
'B': c1:=c1-1;
'F': begin display(1,1); c1:=1; end;
'P': begin
display(dt[n1],doc[n1]);
c1:=1;
end;
end;
until position('?ACDLMQSTX',s,1)>0;
end;
decide:=s;
if s='S' then term:=field(fieldn(tag[dt[c1]],doc[c1]));
end;

```

Function FINDTERM(term: string): real;

```

{-----}
{ Search and display selected term }
{ Return 0 if term exists (action contains a valid action code) }
{ 1 if term does not exist (action is not set) }
{-----}

```

```

var rc: real;
t: string;
begin
t:=fuc(term);
rc:=find(t);
findterm:=rc;
if rc=0 then
if n1post<0
then findterm:=1
else begin
mf:=posting('MFN');
rc:=record(mf);
findterm:=rc;
if rc=0 then
begin
display(1,1);
action:=decide(0);
end;
end;
end;
end;

```

Function FLDUC(k: real): string;

```
{-----}  
{ Returns k-th field of record converted to upper case }  
{-----}
```

```
var f: string;  
begin  
  f:=field(k); uc(f);  
  flduc:=f;  
end;
```

Function CHKREL(t: string): real;

```
{-----}  
{ Check if a relation already exists }  
{-----}
```

```
var i,n: real;  
begin  
  n:=nfields; i:=1;  
  while (i<=n) and (flduc(i)<>t) do i:=i+1;  
  if i>n then chkrel:=0  
    else chkrel:=i;  
end;
```

Procedure UPDINVF;

```
{-----}  
{ Update inverted file (screen is clear because FST is displayed) }  
{-----}
```

```
begin  
  cleardata;  
  updif;  
end;
```

Procedure CREATERM;

```
{-----}  
{ Create new thesaurus term }  
{-----}
```

```
var tuc: string;  
    rc,np: real;  
begin  
  term:=''; clearmsg;  
  displt;  
  clearmsg; write('Enter new term');  
  rc:=edit(term,30,2,2,30,1,' ');  
  if term<>' ' then  
    begin
```

↑  
reverse video

```

tuc:=term; uc(tuc); rc:=find(tuc); np:=-1;
if rc=0 then np:=nxtpost;
if (rc=0) and (np>0)
then errmsg('Term already exists')
else begin
  mfn:=newrec;
  rc:=fldadd(tag[1],1,term);
  update; updinvf;
  action:='S';
end;
end
else action:='T';
end;

```

Procedure ADDRREL;

```

{-----}
{ Add new relation to a term }
{-----}

```

```

var r,rt,rtu: string;
    rc,i,rtag: real;

```

```

Function ADDIT: real;
var tt,fr: string;
    n,k: real;
    relmfn: real;

```

```

Procedure RELADD;
var rc: real;
begin
  n:=nbcc(rtag); k:=1;
  while (k<=n) and (flduc(fieldn(rtag,k))<rtu) do k:=k+1;
  rc:=fldadd(rtag,k+1,rt); update;
end;

```

```

begin
  if (find(rtu)<>0) and (substr(r,1,2)<>'SN')
  then begin
    addit:=1;
    errmsg('Related term does not exist');
  end
  else
  if (chkrel(rtu)<>0) and (substr(r,1,2)<>'SN')
  then begin
    addit:=1;
    errmsg('Relation already exists');
  end
  else
  begin
    rtag:=tag[(rtag-1)/3+1];
    reladd;
  end;
end;

```

```

ir:=substr(invrel,(rtag-1)*3+1,3);
if ir<>' ' then
  begin
    k:=nxtpost; relmfn:=posting('MFN');
    rtag:=tag[(position(rel,ir,1)-1)/3+1];
    rt:=field(fieldn(tag[1],1)); rtu:=rt; uc(rtu);
    k:=record(relmfn);
    reladd;
  end;
k:=record(mfn);
addit:=0;
end;
end;

```

```

begin
box(18,10,3,5,1); box(18,14,3,52,1);
r:=''; rt:='';
repeat
clearbox(19,15,1,50,1);
clearmsg; write('Enter relation code:');
for i:=2 to maxt do write(substr(rel,(i-1)*3+1,3),' ');
clearbox(19,11,1,3,1); rc:=edit(r,3,19,11,3,1,' '); uc(r);
rtag:=position(rel,r,1);
if rtag=0 then write(chr(7));
until (r='') or (rtag>0);
repeat
i:=0;
if rtag>0 then
  begin
    clearmsg;
    rc:=edit(rt,30,19,16,30,1,' '); rtu:=rt; uc(rtu);
    if rtu<>' ' then i:=addit;
  end;
until i=0;
action:='S';
end;

```

```

Procedure DELREL;
{-----}
{ Delete a relation }
{-----}

```

```

var rtag,rc,k,relmfn: real;
    rt,rtu,ir: string;
begin
rtag:=fieldn(dt[c1],doc[c1]);
rt:=field(rtag); rtu:=rt; uc(rtu);
rc:=flddel(rtag);
update;
ir:=substr(invrel,(dt[c1]-1)*3+1,3);
if ir<>' ' then

```

```
begin
  rc:=find(rtu);
  if rc=0 then
    begin
      k:=nxtpost;
      if k>=0 then
        begin
          relmfn:=posting('MFN');
          rtag:=tag[(position(rel,fr,1)-1)/3+1];
          rt:=field(fieldn(tag[1],1));
          rtu:=rt; uc(rtu);
          rc:=record(relmfn);
          if rc=0 then
            begin
              k:=chkrel(rtu);
              if k>0 then
                begin
                  rc:=flddel(k);
                  update;
                end;
            end;
          end;
        end;
      k:=record(mfn);
      action:='S';
    end;

Procedure DELTRM;

{-----}
{ Delete a thesaurus term }
{-----}

begin
  if nfields>1
  then begin
    errmsg('Cannot delete term with relations. Delete all relations first. ');
    action:='S';
  end
  else begin
    rc:=flddel(1);
    update; upinvf;
    action:='T';
  end;
end;

Procedure SHOWDICT;

{-----}
{ List dictionary }
{-----}
```

```

-----
var i,ii,k,sc: real;
    tp: array[1..16] of real;
    ts: array[1..16] of real;
    pg,ft: string;

begin
  ft:=term;
  repeat
    pg:=''; i:=1; sc:=find(ft);
    repeat
      tp[i]:=size(pg)+1; ts[i]:=size(ft);
      pg:=pg|ft;
      ft:=nxtterm; i:=i+1;
    until (i=17) or (ft='');
    i:=i-1;
    for k:=1 to i do
      begin cursor(k+4,5); writeln('_ ',substr(pg,tp[k],ts[k])); end;
    k:=1;
    repeat
      ii:=k;
      chattr(1,k+4,5,30); term:=substr(pg,tp[k],ts[k]);
      sc:=kbdkey(action); uc(action);
      if action=chr(13) then k:=k+1 else
      if action='B' then if k>1 then k:=k-1;
      chattr(0,ii+4,5,30);
      until (position('CPSTX', action,1)>0) or (k>1);
      page(1);
      until (position('CSTX',action,1)>0) or (term='');
    end;
  ----- Body of program THES -----
begin
  maxt:=7; {Number of defined relations }
  rel:= ' SN USEUF BT NT RT '; { Name of relations }
  invrel:= ' UF USENT BT RT '; { Name of inverse relation }
  for i:=1 to maxt do tag[i]:=i; { Tag of relation }

  maxl:=15; q:='';
  dbname:=dbn; { save currently selected data base }
  if dbname<>'THES' then open('THES');
  clear;
  if maxmfn=1 then action:='C' else action:='T';

  repeat
  case action of

```

```
'T': { Term selection }  
  
begin  
  clearmsg;  
  write('Select term');  
  term:=''; dispit;  
  cursor(2,2); readln(term);  
  if term='' then action:='X' else  
    if (substr(term,size(term),1)='$') or (findterm(term)<>0)  
      then action:='L';  
end;  
  
'L': { List of thesaurus terms }  
  
begin  
  uc(term);  
  rc:=find(term);  
  page(1);  
  clearmsg;  
  writeln('* [Next]          B[previous]          P[age]         S[elect]');  
  write ('C[reate term]  T[erm select]      X[extit]');  
  savescr(1);  
  showdict;  
  if term='' then action:='L';  
end;  
  
'S': { Display term relations }  
  
begin  
  rc:=findterm(term);  
  if rc<>0 then action:='L';  
end;  
  
'A': { Add a relation }  
  
  addrel;  
  
'C': { Create a new term }  
  
  createrm;  
  
'D': { Delete a term or a relation }  
  
  if cl=1 then deltrm else delrel;  
  
'Q': { Select term for searching }  
  
begin  
  s:=field(fieldn(tag[dt[cl]],doc[cl]));  
  if size(s)+size(q)+3>255  
    then begin  
      write('ú');  
      action:='?';  
    end;
```

```
end
else begin
  if q<>' ' then q:=q|' + ' ;
  q:=q|s;
  action:=decide(c1+1);
end;
end;

'?: { Display current query }
begin
  savescr(2);
  box(16,8,6,66,2); clearbox(17,9,4,64,1);
  cursor(17,9); lq:=size(q);
  if lq=0 then write('No search terms currently selected') else
  begin
    k:=1; k1:=17;
    repeat
      if lq>64 then i:=64 else i:=lq;
      writeln(substr(q,k,i));
      k:=k+i; lq:=lq-i;
      k1:=k1+1; cursor(k1,9);
    until lq=0;
  end;
  clearmsg; write('Press any key to continue');
  s:=inkey;
  page(2);
  action:=decide(c1+1);
end;
end;
until action='X';

if (dbname<>'THES') and (dbname<>'') then
begin
  open(dbname);
  if size(q)>0 then
  begin
    clear;
    clearmsg; write('Edit search expression or press Enter');
    rc:=edit(q,254,2,1,254,0,' ');
    if size(q)>0 then rc:=search(q);
  end;
end;
option:=' ';
end.
```

