
APPGEN

'C' TOOLKIT

REFERENCE MANUAL

Appgen Business Software, Inc. provides this manual 'as is' without any warranty of any kind, either expressed or implied warranties of merchantability or fitness for a particular purpose. Appgen Business Software may make improvements and/or changes in the product(s) and/or program(s) described in this manual at any time without notice.

Proprietary Rights Notice

All Rights Reserved. This document contains information which is proprietary to and considered a trade secret of Appgen Business Software. It is expressly agreed that it shall not be reproduced in whole or in part, disclosed, divulged, or otherwise made available to any third party either directly or indirectly. Reproduction of this document for any purpose is prohibited without the prior expressed written authorization of Appgen Business Software, Inc.

Copyright Notice

Copyright © 1999 by
Appgen Business Software, Inc.
1300 Veterans Memorial Highway
Hauppauge, NY 11788
(516) 471-3200
(800) 231-0062

Trademarks

Appgen is a registered trademark ® of Appgen Business Software, Inc. All other trademarks and registered trademarks are the property of their respective owners.

APPGEN

'C' TOOLKIT

INTRODUCTION	v
CHAPTER 1: OVERVIEW	1-1
CHAPTER 2: PROGRAM ORGANIZATION	2-1
2.1 Include Files	2-1
2.2 Libraries	2-2
CHAPTER 3: ACCESSING APPGEN FILES	3-1
CHAPTER 4: SUB-ROUTINES	4-1
4.1 Calling Sub-Routines	4-1
4.2 Argument Vector in Sub-Routines	4-1
4.3 Global Variables in Sub-Routines	4-2
CHAPTER 5: STAND ALONE PROGRAMS	5-1
5.1 Calling Stand Alone Programs	5-1
5.2 Argument Vector in Stand Alone Programs	5-2
5.3 Global Variables in Stand Alone Programs	5-2
5.4 Exiting From Stand Alone Programs	5-4
CHAPTER 6: COMPILING PROCEDURES	6-1
6.1 Makefile Targets	6-1
6.2 Make.XX.Head	6-2
6.3 Make.XX.Tail	6-2
6.4 Make.Base.Head	6-3
6.5 Compiling Sub-Routines	6-3
6.6 Compiling Stand Alone Programs	6-5
6.7 Summary	6-7

CHAPTER 7: 'C' LIBRARY ROUTINES	7-1
7.1 Data Base Utilities	7-1
Table of Errors	7-1
db_create - create data base file	7-2
db_delete - delete data base file	7-2
db_delrec - remove a record from a file	7-3
db_lock - data base file lock	7-4
db_unlock - data base file unlock	7-4
db_newrec - add a new record to a file	7-5
db_open - open a data base file	7-6
db_close - close a data base file	7-6
db_read - get a record by key	7-7
db_release - unlock and release the current record	7-8
db_rewind - move file pointer to beginning	7-9
readnext - sequential read facility	7-9
db_stat - return statistics about records and fields	7-10
db_write - make changes in the current record permanent	7-11
delete - delete a field	7-12
extract - extract a field	7-13
insert - insert a field	7-14
replace - replace a field	7-15
char_extract - extract a character	7-16
int_extract - extract an integer	7-16
long_extract - extract a long	7-16
str_extract - extract a string	7-16
sstr_extract - extract a string	7-16
7.2 Screen Utilities	7-18
Bell - Ring bell on terminal	7-18
clrflld - erase a field on terminal screen	7-19
clrscr - clear entire screen	7-20
dispdata - format and display a string	7-21
displaym - display message on screen	7-22
draw_box - draw a box on the screen	7-23
getdata - get input with checking	7-24
getfld - take input of given width at position on screen	7-25
padget - display a field and prompt for input	7-26
putfld - display a string of a given width at a given position	7-27
title - display the title information at top of screen	7-28
S_cook - restore original ioctl state of terminal	7-29
S_uncook - set raw term modes	7-29
S_exit - de-initialize screen handler routines	7-30

S_flush - force all queued output to the screen	7-31
S_read - get the next character	7-32
S_refresh - redraw the screen	7-33
7.3 Window Utilities	7-34
w_clrflld - erase a field	7-34
w_clrscr - clear window	7-35
W_exit - remove a window	7-36
w_getdata - get input with checking	7-37
w_getfld - get input	7-39
W_init - open window	7-40
w_padget - get input with size checking	7-41
w_putfld - display string	7-42
7.4 Function Evaluator	7-43
F_string - return a pointer to a function	7-43
Func_Eval - evaluate a function	7-44
Func_Parse - parse a function	7-45
Func_Exec - execute a function	7-46
7.5 Conv - Data Conversions	7-47
Atoi - string to integer conversions	7-47
Atol - string to long conversions	7-47
date_ck - verify that date is valid date	7-48
date_in - convert a date to the Julian internal date	7-49
date_out - convert Julian date to the external format	7-50
date_4out - convert Julian date to the external format	7-51
toascii - convert to ASCII	7-52
tolower - convert a character to lower case	7-52
toupper - convert a character to upper case	7-52
Lower - convert a string to lower case	7-52
Upper - convert a string to upper case	7-52
7.6 Type - Character Classification	7-53
isalnum - is character alphanumeric	7-53
isalpha - is character alphabetic	7-53
isascii - is character ASCII	7-53
isctrl - is character control	7-53
isdigit - is character digit	7-53
isgraph - is character graphic	7-53
islower - is character lower case	7-53
ismeta - is character meta	7-53
isprint - is character printable	7-53

TABLE OF CONTENTS

ispunct - is character punctuation	7-53
isspace - is character space	7-53
isupper - is character upper case	7-53
7.7 HP - High Precision Math Utilities	7-55
hp_exp - natural exponential function	7-55
hp_log - natural logarithmic function	7-55
s_to_hp - convert short to hp	7-56
hp_to_s - convert hp to short	7-56
l_to_hp - convert long to hp	7-56
hp_to_l - convert hp to long	7-56
a_to_hp - convert string to hp	7-56
hp_to_a - convert hp to string	7-56
hp_add - add hp to hp yielding hp	7-58
hp_sub - sub hp from hp yielding hp	7-58
hp_mul - mul hp by hp yielding hp	7-58
hp_div - divide hp by hp yielding hp	7-58
hp_mod - hp modulo hp yielding hp	7-58
hp_pwr - raise hp to power yielding hp	7-58
hp_adds - add hp to short yielding hp	7-59
hp_subs - sub short from hp yielding hp	7-59
hp_muls - multiply short by hp yielding hp	7-59
hp_divs - divide hp by short yielding hp	7-59
hp_mods - hp modulo short yielding hp	7-59
hp_pwrs - raise hp to power yielding hp	7-59
hp_round - round hp	7-60
hp_copy - copy hp	7-60
hp_negate - negate hp	7-60
hp_cmp - compare hp	7-60
hp_isneg - is hp negative	7-60
hp_isshort - is hp a short	7-60
hp_islong - is hp a long	7-60
hp_hasfract - does hp have fractional part	7-60
hp_int - truncate hp to integer	7-60
7.8 Strings Utilities	7-62
dechar - remove character from string	7-62
derep - replace character in string	7-62
detab - remove tabs from string	7-62
scat - concatenate strings	7-62
sncat - concatenate strings	7-62
scmp - compare strings	7-62
strccmp - compare strings	7-62

sncmp - compare strings	7-62
scopy - copy strings	7-62
sncpy - copy strings	7-62
slen - string length	7-62
schr - occurrence of character	7-62
srchr - occurrence of character	7-62
spbrk - occurrence of character	7-62
stok - occurrence of token	7-62
strmtok - occurrenc of token	7-62
7.9 Other Utilities	7-65
Chain_Away - chain to a new program	7-65
formdata - format a data field for display or printing	7-66
getvar - look for Appgen environment variable	7-67
mmove - general memory to memory mover	7-68
pformat - perform printf type formatting	7-69
trap - ignore certain signals	7-70
CHAPTER 8: IMPORTING FOREIGN DATA STRUCTURES	8-1
8.1 Overview of Importing	8-2
8.2 Program Organization	8-5
8.3 Executing the Program	8-12
8.4 Reviewing Program Results	8-12
APPENDICES	
Appendix A - Sub-Routine Listing	A-1
Appendix B - Stand Alone Program Listing	B-1
Appendix C - Example 1 Data	C-1
Appendix D - Import.C Listing - Example 1	D-1
Appendix E - Import.C Listing - Example 2	E-1

TABLE OF CONTENTS ---

CHAPTER 1: OVERVIEW

Subroutines and stand alone programs are useful for executing complex calculations or processes that would be cumbersome to perform using only Appgen functions. Many examples of subroutines and stand alone programs can be found in the appgen/src directory. The subroutine named arma3 that is listed in Appendix A is provided as an example to illustrate the techniques of managing data in subroutines. The stand alone program named file_copy that is listed in Appendix B is provided as an example of a stand alone program. Subroutines and stand alone programs can perform any of the operations listed below:

- Create, delete, read, write, and lock records in Appgen files.
- Open Appgen files that are not opened by the calling Appgen program.
- Access other files that are not Appgen files.
- Extract, delete, insert, and replace attributes and their multi-values in Appgen files.
- Display messages or data on the terminal screen.
- Send output to or receive input from other devices, i.e. printers, terminals, etc.
- Return after execution to the UNIX shell (stand alone programs only).

Examples of subroutines and stand alone programs can be found in the directory appgen/src. A partial list of the subroutines and stand alones is shown below:

Subroutines

Purpose

align	print an alignment pattern on a printer
ar05sub	construct special headings for the aged trial balance
dollar	construct a string of words for a dollar amount
prdd	payroll calculations

Stand Alone Programs

ar_stmnts	calculate Accounts Receivable statements
Compute_st	calculate General Ledger financial statements
file_copy	general purpose file copy routine
prg_ropen	purge the Accounts Receivable Open file
import1,import2	model import programs

CHAPTER 2: PROGRAM ORGANIZATION

This section describes the Appgen include files, global variables, and other declarations that are used in both subroutines and stand alones. The Appgen libraries are described in general terms. Additional descriptions and examples of the Appgen library functions can be found in the various subroutines and stand alones.

2.1 INCLUDE FILES

The Appgen library routines rely on several files of Appgen variable and constant declarations. In order to use the Appgen library routines, the appropriate header files should be included at the beginning of the subroutine or stand alone program.

The usage of the Appgen header files, located in the appgen/include directory, will vary depending on the Appgen library routines used. The subroutines and stand alone programs in the appgen/src directory include various combinations of header files. To use the Appgen library calls used in a particular Appgen subroutine or stand alone program, you should include the same header files.

The appgen.h header file must be included in the subroutine or stand alone program in order to use any of the Appgen libraries. Several Appgen header files frequently included in subroutines and stand alones are discussed below. The appgen.h, vm.h, and db.h header files are the minimum include files needed and should be included in the order listed below.

- | | |
|----------|--|
| appgen.h | Contains declarations used by all Appgen programs. The UNIX system include file, <code>stdio.h</code> , is included in <code>appgen.h</code> and should not be listed again in the include section of the subroutine or stand alone program. |
| vm.h | Contains declarations used by the Appgen Virtual Machine software to standardize certain system calls among various computers. |
| db.h | Contains declarations used by Appgen data base routines. |
| hp.h | Contains declarations needed for the Appgen high precision math functions. |

2.2 LIBRARIES

The Appgen data base functions are required to access and manipulate Appgen data files. These Appgen functions provide the necessary tools for:

- Creating, deleting, opening, closing, and locking of files.
- Creating, deleting, reading, writing, and locking of records.
- Extracting, deleting, inserting, and replacing of attributes and their multi-values.
- Extended precision arithmetic.

Appgen string functions are supplied for portability and consistency. Minor differences in system supplied libraries can cause indeterminate results during program execution.

Appgen High Performance math functions are provided to make coding easier, portable, and consistent.

All the above functions, and also the others described in Chapter 7 of this manual, are found in the two libraries appgen.a and strings.a to be found in your base Appgen directory.

CHAPTER 3: ACCESSING APPGEN FILES

The types of files that can be accessed in both subroutines and stand alone programs include:

- Appgen data files
- Appgen Screen/Menu PDEFs
- Appgen Report/Posting PDEFs
- Files not having an Appgen format

Both types of Appgen PDEFs listed above contain Header (PROG.NAME) records. The information in the Header records can be accessed in subroutines or stand alone programs with the use of Appgen data base functions. The record layout shown below lists the data fields of a Report Header record.

ATTRIBUTE	VALUE	DESCRIPTION
0	0	PROG.NAME
1	0	FUNCTION NAME
2	1..100	DATA FILES (MULTI-VALUED)
3	1..n	HELP TEXT
4	0	REPRINTS ALLOWED (Y/N)
5	0	REPORT TITLE
6	0	REPORT TYPE (PRINTER/SCREEN/BOTH)
7	0	MAXIMUM WIDTH
8	0	RETURN TO PROGRAM
9	0	RETURN TO PDEF
10	0	SUBSEQUENT PROGRAM
11	0	EXECUTION PROGRAM
12	0	NOT USED
13	0	SUBSEQUENT PDEF
14	0	LINES PER PAGE
15	0	BEFORE FUNCTION
16	0	HEADING TYPE (P/R)

ACCESSING FILES

The record layout for a Posting Header Record is as follows:

ATTRIBUTE	VALUE	DESCRIPTION
0	1	PROG.NAME
1	0	FUNCTION NAME
2	1..100	DATA FILES
3	1..n	HELP TEXT
4	0	NOT USED
5	0	NOT USED
6	0	NOT USED
7	0	DELETE (Y/N)
8	0	RETURN TO PROGRAM
9	0	RETURN TO PDEF
10	0	SUBSEQUENT PROGRAM
11	0	EXECUTION PROGRAM
12	0	NOT USED
13	0	SUBSEQUENT PDEF
14	0	POST STATUS
15	0	BEFORE FUNCTION
16	1..n	REFERENCE ONLY (Y/N)

The record layout for a Screen/Menu Header record is as follows:

ATTRIBUTE	VALUE	DESCRIPTION
0	1	PROG.NAME
1	0	PROGRAM NAME
2	1..100	DATA FILES (MULTI-VALUED)
3	1..n	MULTI-VALUED DESCRIPTION
4	0	TOP OUT ALLOWED
5	0	BACKSLASH ALLOWED
6	0	RETURN TO PROGRAM
7	0	RETURN TO PDEF
10	0	MODES ALLOWED
15	0	VERIFY ADD MODE

The attribute and value numbers shown above are required to process data fields with Appgen data base functions. All fields in an Appgen file must be accessed by attribute number and value number; no symbolic names are provided.

CHAPTER 4: SUB-ROUTINES

Section four provides an overview of subroutines and several specific examples from the Appgen subroutine, arma3. This section explains the basics of calling subroutines, passing data to the subroutine as arguments on the function line, and returning a value to the function line after executing the subroutine call.

4.1 CALLING SUB-ROUTINES

Subroutines are executed through the use of the 'U' function. The 'U' function can be used on any Appgen PDEF function line. Function lines exist in the following Appgen PDEF records:

1. Screen/Menu PDEF - Prompt Items (Before Function, Default Function, Key Function, After Function, Verify Function, Fork Function)
2. Report & Posting PDEFs - Header record (Before Function), Selection Item (Before Function, Default Function, Key Function, After Function, Verify Function, Fork Function)
3. Report PDEF - Report Format Item (Before Function, Key Function, After Function)
4. Posting PDEF - Posting Item (Key Function, Before Function, Posting Function, After Function)

The syntax of the subroutine call is:

U(a0,a1,a2,...,a9)

where a0 is the name of the subroutine enclosed in quotes; and a1, a2, ..., a9 are comma-separated optional arguments to the subroutine. The maximum number of arguments allowed is ten including the name of the subroutine. Subroutines automatically return to the calling program. The return value of the 'U' function is explained in the next section.

4.2 ARGUMENT VECTOR IN SUB-ROUTINES

Function line arguments are used to pass various types of data to the subroutine. A few examples of the type of data that can be passed as arguments to subroutines are:

1. An index into the list of data files opened by the calling Appgen program.
2. A record key from one of the files listed in the Header record of the calling PDEF.

3. The attribute and value numbers that are to be extracted or replaced in a data file.
4. A file name to be opened, read, or written.

Assuming that the user has chosen the name *progm* for the subroutine, the subroutine should be declared in the source code as:

```
progm ( argc, argv, dest )
int argc;
char *argv[], *dest;
```

The function processor which executes the subroutine requires that the subroutine be defined as an integer type function. The default function type for 'C' language functions is an integer type. The number of arguments to the 'U' function is contained in the *argc* variable. This number includes the name of the function, *progm*, in this case. The variable *argv* is a vector of pointers to the character strings received by the function processor from the function line in the PDEF. These pointers or arguments to the subroutine are known in the subroutine source code as *argv[0]*, *argv[1]*, *argv[2]*, ... *argv[9]*, with *argv[0]* being the name of the subroutine. Up to ten arguments may be passed to the subroutine including *argv[0]*, the name of the subroutine.

The use of *dest* is optional in subroutines. The variable *dest* is a pointer to a buffer used by the function processor. The buffer contents are the return value of the 'U' function call. In the example below from *arma3* the *dest* buffer contains a flag indicating whether the subroutine was successful. If the statement is executed, a value of 'N' is copied into the *dest* buffer and 'N' is the return value of the 'U' function.

```
scopy (dest, "N");
```

If *dest* is not modified in the subroutine, the function processor will return a NULL value after processing the 'U' function.

4.3 GLOBAL VARIABLES IN SUB-ROUTINES

The declaration of Appgen constants and variables depends on the data structures and Appgen library routines being used. Two Appgen variables, *argvec* and *db_*, are instrumental in managing data files in subroutines:

argvec

is a global variable used by Appgen library routines. It is declared outside of the current source file in *appgen.h*. When subroutines are compiled, they are linked to an Appgen program in which *argvec* is already defined. It is not necessary to redefine *argvec* in subroutines. The declaration and usage of the *argvec* variable in stand alone programs is discussed in Section 5.3.

db_ DB is a structure defined in db.h that describes the Appgen file format. In order to access an Appgen file, the subroutine must declare a DB pointer. Subroutines access DB structures differently than stand alones because the data files listed in the Header record of the current PDEF remain open when the subroutine is called. The global variable db_ allows subroutines to access the Appgen data files opened by such Appgen programs as *Maint*, *Printem*, *Selectem*, or *Postem*. In stand alone programs, the db_ vector of DB pointers is undefined.

In subroutines the db_ vector is assigned values when an Appgen program opens the data files listed in the PDEF Header record. The db_ vector can be referenced in subroutines by declaring the db_ vector as follows:

```
extern DB *db_[];
```

Any Appgen data files that are to be used in the subroutine should be opened by listing the file names in the Header record of the current PDEF. The data files listed in the Header record of the current PDEF will be opened automatically when the current Appgen program is chained-to. The same data files will also be closed automatically when the next program is chained-to. If the data files listed in the Header record are closed during the subroutine, they should be reopened since the calling Appgen program will attempt to close all the data files that it opened initially.

The db_ vector elements are defined differently for subroutines that are called from within *Printem* than for subroutines that are called from within *Maint*, *Selectem*, or *Postem*. The differences are outlined below.

		<i>Printem</i>	<i>Maint/Selectem/Postem</i>
db_[0]	->	current PDEF	File 1
db_[1]	->	File 1	File 2
db_[2]	->	File 2	File 3
db_[3]	->	File 3	File 4
.	.	.	.
.	.	.	.
.	.	.	.

The arrows above indicate that the DB pointer on the left references one of the DB structures listed on the right. In a subroutine that is called from *Printem*, the first DB pointer in the db_ vector, db[0], points to the current PDEF. Subsequent db_ elements are DB pointers to the data files listed in the Header record of the current PDEF. In a subroutine that is called from *Maint*, *Selectem*, or *Postem*, the DB pointers reference only the Appgen files listed in the Header record of the PDEF. An example of the usage of the db_ vector in a subroutine is taken from arma3 that is listed in Appendix A. This subroutine is called from Prompt Item 12 in PDEF.AR040000, a *Maint* PDEF. The Appgen files listed in the Header record of PDEF.AR040000 are shown below with their respective location in the db_ vector:

HEADER RECORD FILE LIST	db_ element
1. AR-PTRANS	db_[0]
2. AR-CUSMAS	db_[1]
3. AR-OPEN	db_[2]
4. AR-GLDIST	db_[3]
5. AR-COMPNY	db_[4]

The DB pointers shown above are used with the Appgen library functions to access the corresponding Appgen files. The statement below is from arma3 and contains a for loop statement. In each execution of the loop a multi-valued item is extracted from a multi-valued list of items in attribute 27 of the AR-CUSMAS file. After each extract() the mutli-valued item is in the character array named dest.

```
for(vnum = 1; extract(db_[1], 27, vnum, dest, 79) > 0; vnum += 1)
```

The statements below are also from arma3 and are executed with each iteration of the for loop shown above. The multi-valued item just extracted is combined with the customer number, argv[2] to form the key to the AR-OPEN file. The records in the AR-OPEN file are multi-part, consisting of the customer number, a '*' separator character, and the document number. All official Appgen applications use the '*' character as a separator character in multi-part keys. The parts of the record key are combined in the character array named temp.

```
    sprintf (temp, "%s*%s", argv[2], dest);
    if (db_read (db_[2], temp, 0 ) != 0)
    {
        putfld (MESGLINE,0,80,temp);
        putfld (-1, -1, 80, ": document not on AR-OPEN");
        getfld (-1, -1, 0, "");
        continue;
    }
```

The second statement above reads the AR-OPEN file with the key field contained in temp. As demonstrated by the second statement, the subroutine should allow for all possible control paths that the subroutine may take. In the event that db_read() was unable to read the record, an error message is displayed on the terminal screen and subroutine control returns to the for loop statement. The putfld() and getfld() routines accept as arguments a row number, column number, field width, and name of a character string array, in that order. The -1 value that is used with putfld() and getfld() is interpreted as the current row or column that the screen cursor is on. In the sprintf() statement above, the contents of argv[2] above are determined by the 'U' function call. The subroutine arma3 is called from Prompt Item 12 in the *Maint* PDEF.AR040000 by the function below:

```
U("arma3", E, E(1) )
```

The arguments are interpreted in the subroutine as follows:

Argument	Value	Description
argv[0]	arma3	the name of the subroutine
argv[1]	E	document number entered by user
argv[2]	E(1)	customer number entered by user

The arguments received by the subroutine from the function processor are pointers to character strings. If the arguments are needed in an integer format, as for use with an Appgen library routine, an Atoi() conversion is necessary to convert the character strings to integer values. An example of an Atoi() conversion of a subroutine argument is as follows:

```
inum = Atoi ( argv[2] );
```

The Appgen standard library routine Atoi() is documented in Section 7. Since subroutines return automatically to the function line in the calling Appgen PDEF, exit() statements should not be placed in the subroutine. This would cause the subroutine to fall to the shell without necessary Appgen exiting procedures being performed, such as data files being closed. The subroutine can contain return() statements to cause subroutine control to return to the function line without executing any further statements.

CHAPTER 5: STAND ALONE PROGRAMS

Section five first describes the procedures for calling stand alone programs and the Appgen chaining process. This section also describes the use of Appgen global variables. Methods of exiting stand alone programs are discussed in Section 5.5.

5.1 CALLING STAND ALONE PROGRAMS

Stand alone programs are executed by listing the name of the stand alone program on one of the "Program Name" lines of the three Appgen PDEF records listed below.

1. Screen/Menu PDEF - Header record (End Program). In this situation the stand alone program is listed in the Header record of a PDEF and is executed after the PDEF routine.
2. Screen/Menu PDEF - Menu Item (Chain Program). The stand alone program in this case is executed when the user selects the particular Menu Item that lists the stand alone program as the Chain Program.
3. Report/Posting PDEF - Header record (Execution Program, Subsequent Program, Return To Program). The order of execution of the Report/Posting PDEF programs is Execution Program then Subsequent Program. The Return To Program is executed if the Execution Program fails for any reason. The stand alone program could be listed as any of these programs, for example in a combined reporting and posting process.

Listing the stand alone program in one of the places indicated above causes the stand alone program to be executed by the Appgen Chain_Away() routine. Chain_Away() first closes all data files that may have been opened by the previous execution program. Any data files or PDEF files that are to be read by the stand alone program must first be opened by calling db_open. Before an exit() or Chain_Away() from the stand alone program, any Appgen files that were opened must be closed with the db_close() routine as described in Section 5.5. Although Chain_Away() closes all open files with the close() system call, the db_close() routine should be performed first in the stand alone program to correctly finalize the usage of an Appgen file. After closing files Chain_Away performs an execl() which transforms the current process into a new process. Additional information on the execl() routine is available in your UNIX Reference Manual.

The above PDEF records, with the exception of the Report/Posting PDEF Execution Program, also allow the specification of a chain-to PDEF. The chain-to PDEF name is passed as an argument to the stand alone program as discussed in the next section.

5.2 ARGUMENT VECTOR IN STAND ALONE PROGRAMS

The argument vector for stand alone programs is not defined by the user, but by the Appgen Chain_Away() routine that executes the stand alone program. The stand alone program should be declared in the source code file as:

```
main ( argc, argv )
int argc;
char *argv[];
```

The argc variable is the number of arguments to the stand alone program. The argv variable is a vector of pointers to the character strings received by the stand alone program from the Chain_Away() routine. Chain_Away() executes the stand alone program by performing an exec() statement such as:

```
exec ("prog", "XX000000", argvec[2], argvec[3], argvec[4], argvec[5], argvec[6],
argvec[7], argvec[8]);
```

In the statement above, the items in parentheses are passed as arguments to the stand alone program. The arguments to the stand alone program are known in the program source code as argv[0], argv[1], ..., argv[8]. In the example above argv[0] is *prog*, the name of the stand alone program, and argv[1] is *XX000000*, the name of the chain-to PDEF. A chain-to PDEF must be specified when the stand alone program is called. If the program does not use a PDEF, this argument will be ignored; but it must not be null. The arguments argv[2] through argv[8] are assigned the current values of the Appgen global variables that are described in the next section.

5.3 GLOBAL VARIABLES IN STAND ALONE PROGRAMS

argvec

The declaration of argvec is necessary in stand alone programs because other Appgen programs require the declaration of argvec. Stand alone programs must declare argvec before main() with the statement:

```
char **argvec;
```

This statement declares argvec as a pointer to a vector of pointers. The pointers identify character arrays which contain the following information about an Appgen process:

argvec [0]	PROGNAME	-	the name of the program
argvec [1]	PDEFNAME	-	the name of the PDEF or RDEF to execute
argvec [2]	SECURITY	-	reserved for future use
argvec [3]	INSTALLNAME	-	the name of the installation which appears at the top of all screens
argvec [4]	SYSINITIALS	-	the application initials

argvec [5]	COMPANYNUM	- the company number
argvec [6]	TTYNAME	- the tty name
argvec [7]	KEYFILE	- the name of the file containing the list of record keys generated by <i>selectem</i>
argvec [8]	RECKEY	- the name of the file containing the selection prompt responses

The middle column above contains symbolic names that can be substituted in either subroutines or stand alone programs for the items in the first column. When a stand alone program is executed, the `Chain_Away()` routine assigns the current values in `argvec[2]` through `argvec[8]`. The current values of `argvec` are the values from the previous execution program, usually an Appgen program. Stand alone programs must initialize `argvec` within `main()` as follows:

```
argvec = argv;
```

Once the `argvec` variables have been given the values of the stand alone program arguments, `argv`, the symbolic names above can be used in the stand alone program. For example, in the stand alone program named `file_copy` that is listed in Appendix B, the `argvec` variable `PDEFNAME` is used to open the chain-to PDEF:

```
printf(p_name, "PDEF.%s", PDEFNAME);
if ((pdef=db_open (p_name)) == (DB*) NULL)
{
    sprintf(buffer,"Open failed on [%s] in [%s]",p_name,PROGNAME);
    error ();
}
```

The first statement creates a character string in the character array named `p_name`. The format of the character string is `PDEF.XX000000`, where `XX000000` is the character string named by the `argvec` variable named `PDEFNAME`, and `XX` are the application initials. In the second statement the value returned by `db_open()` is assigned to the DB pointer, `pdef`. If the value returned by `db_open` is null, a message is displayed that the PDEF could not be opened. DB pointers are defined in general terms in Section 4.3 and are described below in the context of stand alone programs.

DB

Because all data files are closed before the stand alone program is executed, the `db_vector` described in Section 4.3 cannot be used in the stand alone program to access an Appgen file. Stand alone programs must declare a DB pointer in the following manner:

```
DB *fr_file;
```

This statement taken from the `file_copy` program declares `fr_file` as a pointer to a DB structure. If more than one data file is opened in the stand alone program, then a vector of DB pointers could be used. For example, the following statement declares a vector of three DB pointers:

```
DB *fp[ 3 ];
```

The DB pointers are assigned values by opening Appgen data files or PDEF files. An example of opening a PDEF file using the argvec variable PDEFNAME is shown in the discussion of argvec above.

5.4 EXITING FROM STAND ALONE PROGRAMS

Before returning from either a stand alone program or a subroutine, it may be necessary to file any changes made to Appgen files by executing a `db_write()`. The `db_write()` statement below is from the stand alone program `file_copy`:

```
db_write(to_file);
```

Before returning from a stand alone program all data files must be closed. The stand alone program must contain provisions for all possible exits from the program in order to assure that all data files are closed before returning from the program. An example of closing an Appgen file is as follows:

```
db_close(to_file);
```

Stand alone programs will fall to the shell unless they specify an `exec()` or `Chain_Away()`. Stand alone programs can also return to the UNIX shell by executing an `exit()` statement such as:

```
exit(0);
```

An example of the use of the Appgen `Chain_Away()` routine is:

```
Chain_Away("Menu", "XX000000");
```

This statement calls the `Chain_Away()` routine which performs an `exec()` to transform the current process into a new process. As described in Section 5.2, the arguments to the `exec()` statement within `Chain_Away()` contain the two arguments shown above, as well as the global argvec variables. For example, the `exec()` statement within the `Chain_Away()` routine would have the format:

```
exec("Menu", "XX000000" argvec[2], argvec[3], argvec[4], argvec[5], argvec[6],  
argvec[7], argvec[8]);
```

This `Chain_Away()` statement would call the *Menu* program to display the menu screen of PDEF.XX000000. Examples of calls to `Chain_Away()` can be found in the Appgen stand alone programs.

CHAPTER 6: COMPILING PROCEDURES

Subroutines and stand alone programs are compiled through the use of the UNIX 'make' command and the makefile. The program 'make' reads the makefile to determine how the components of a program depend on each other and how to process the components to create an up-to-date version of the program. In reviewing Sections 6.1 through 6.4 below, it is recommended that the reader have listings of the following files for reference: make.XX.head, make.XX.tail, make.base.head, make.base.tail, and makefile. Additional information on the 'make' and 'cc' commands is available in your UNIX Reference Manual.

6.1 MAKEFILE TARGETS

To understand the Appgen makefile it is necessary to understand the target entries in the makefile. The target entries in the makefile have the following format:

```
TARGETNAME:    DEPENDENCIES
                SHELL COMMANDS
```

TARGET is the name of the file to be created; DEPENDENCIES are other files that must be created before the TARGET can be created; and SHELL COMMANDS are the UNIX commands that are executed to create the TARGET. The shell commands MUST begin on the line immediately following the list of dependencies and MUST be preceded on the line by a single tab character. The list of dependencies on the first line of the target entry can be continued on subsequent lines by placing a backslash character '\ ' at the end of the line that is to be continued. The backslash MUST be followed immediately by a newline character and MUST NOT be at the end of the last line of the dependencies. Continuation lines of dependencies can contain spaces or tabs at the beginning of the line for alignment.

The Appgen makefile is created from the concatenation of files such as make.base.head, make.XX.head, make.XX.tail, and make.base.tail in the appgen directory. The initials 'XX' are used throughout this section and refer to the application initials, for instance GL (General Ledger) or AR (Accounts Receivable).

The make.XX.head, make.XX.tail, makefile, and possibly make.base.head files will need to be modified in order to compile a new subroutine or stand alone program. The make.XX.head files and the make.base.head file contain macro definitions that are described below. The make.XX.tail and the make.base.tail file contain target entries for the programs that are to be compiled and linked. The make.XX.tail files are only needed for stand alone programs. If the application is not a standard application, then the make.XX.head and make.XX.tail files will have to be created with a UNIX text editor such as 'vi', according to the formats described below.

6.2 MAKE.XX.HEAD

The make.XX.head file contains a sequence of entries known as macro definitions of the form string1 = string2. Subsequent appearances of \$(string1) are replaced by string2. These macros become part of the makefile when the makefile is remade as described in Sections 6.5 and 6.6 below. A macro definition line can be continued on subsequent lines by placing a backslash character '\ ' at the end. The backslash MUST be followed immediately by a newline character, and there MUST NOT be a backslash on the last line of the macro definition. Subsequent lines of the macro definition may contain spaces or tabs at the beginning of the line for alignment. There are two types of macro definitions in make.XX.head, XXOBJ, and XXTARGS:

XXOBJ The \$(XXOBJ) macro defines the object files for the subroutines within an application. These object files must be created and linked to the Appgen object files in order to create executable Appgen programs. The \$(XXOBJ) macros are summarized by the \$(APPLOBJ) macro in the makefile. The \$(APPLOBJ) macro is described in Section 6.4 below.

XXTARGS The \$(XXTARGS) macro defines the executable stand alone programs that must be created. The programs defined by \$(XXTARGS) are summarized by the \$(TARGETS) macro in the makefile. The \$(TARGETS) macro is described in Section 6.4 below.

6.3 MAKE.XX.TAIL

The make.XX.tail file contains target entries for stand alone programs. These target entries are added to the makefile when the makefile is remade as described in Sections 6.5 and 6.6. These target entries list the program object file, the APPL.a library, and the strings.a and appgen.a libraries as dependencies. The shell commands link the dependency files to create the executable stand alone program.

The shell commands in the target entry MUST begin on the line immediately following the list of dependencies and MUST be preceded on the line by a single tab character. The list of dependencies MUST begin on the first line of the target entry and can be continued on subsequent lines by placing a backslash '\ ' at the end of the line that is to be continued. The backslash MUST be followed immediately by a newline character and MUST NOT be at the end of the last line of the dependencies. Subsequent lines of the dependencies list may contain blanks or spaces at the beginning of the line for alignment.

6.4 MAKE.BASE.HEAD

The make.base.head file contains two macro definitions called \$(APPLOBJ) and \$(TARGETS) that summarize all of the \$(XXOBJ) and \$(XXTARGS) entries listed in the make.XX.head files. The \$(APPLOBJ) object files are required to create the APPL.a library which is needed to create the executable Appgen programs. The \$(TARGETS) macro specifies all of the executable programs to be created.

The make.base.head file is concatenated to the makefile when the makefile is remade as described in Sections 6.5 and 6.6. If a new make.XX.head file is created, the \$(XXOBJ) and \$(XXTARGS) macros must be added to the make.base.head file. The \$(APPLOBJ) macro summarizes all of \$(XXOBJ) files, and the \$(TARGETS) macro summarizes all of the \$(XXTARGS) files. The \$(XXOBJ) and \$(XXTARGS) macros are defined in detail in the makefile when the make.XX.head files are concatenated to the makefile.

6.5 COMPILING SUB-ROUTINES

The procedures for compiling subroutines are summarized and described below. The procedures assume that the source files and object files are placed in the appgen/src directory, and the executable program files are placed in the appgen/bin/directory.

1. Modify make.XX.head.
2. Modify the XX.u text file.
3. Modify make.base.head if necessary.
4. Remake the makefile.
5. Execute make.

Steps one, two, and four above are necessary only when compiling new subroutines for the first time. The make.XX.head and XX.u text files that appear in steps one and two are concatenated to the makefile in step four. If the application is new, then the make.base.head file must be modified as described below.

STEP 1 - MODIFY MAKE.XX.HEAD

The first step in compiling a new subroutine is to modify the make.XX.head file in the appgen directory. The pathname of the subroutine object file should be added to the XXOBJ definition in the make.XX.head file. The XXOBJ macro specifies the subroutine object files that are linked with the Appgen program object files to create executable programs. An example of an XXOBJ definition is shown below.

```
XXOBJ = src/progm.o src/progr.o
```

In this example the object files src/progm.o and src/progr.o would be linked in the makefile to the Appgen object files.

STEP 2 - MODIFY XX.u FILE

All subroutines must be listed in a XX.u text file in the appgen directory. In the makefile the .u files are automatically sorted together and run through a program called usertab to produce the source files one through three below. The source files below are the part of the Appgen code that maps the name of the subroutine to the Appgen function processor. Each line in an XX.u file has the name of a subroutine followed by one or more numbers; the numbers correspond to the programs in the makefile as follows:

```
1 = Maint_User.c
2 = Post_User.c
3 = Print_User.c
4 = Select_User.c
```

For example to include a subroutine called *prog*m in *Maint* for use in an application whose initials are XX, add a line to the file XX.u as follows:

```
prog
```

The number one in the above example corresponds to the *Maint_User.c* program as shown in the table above. The *Maint_User.c* program contains code that directs the function processor to find the *prog*m subroutine in the *Maint.c* program.

STEP 3 - MAKE.BASE.HEAD

The names of all XX.u files must appear in the PACKLIST line of make.base.head. When subroutines are created for a new application, add the name of the application's .u file to the PACKLIST line.

The \$(XXOBJ) macro that is defined in the make.XX.head file must be added to the make.base.head file if the application is new. The \$(XXOBJ) macro should be added to the end of the definition of the APPLOBJ macro in the make.base.head file. The APPLOBJ macro in the make.base.head file can be continued on another line by placing a backslash character at the end of the line that is to be continued. The backslash MUST NOT be at the end of the last line of the macro definition.

STEP 4 - MAKE NEWMAKE

The fourth step in compiling a subroutine is to recreate the makefile. The makefile is located in the appgen directory and is created from the concatenation of files such as make.base.head, make.XX.head, make.base.tail, and make.XX.tail. To recreate the makefile, go to the appgen directory that contains the makefile and type 'make newmake' at the UNIX shell prompt.

STEP 5 - EXECUTE MAKE

The final step in compiling a subroutine is to type 'make' at the UNIX shell prompt in the appgen directory. The subroutine will be created in the appgen/bin directory. Any other source code files that have been modified will also be compiled. To compile a single subroutine named *prog*, that is linked to *Maint*, type 'make bin/Maint' at the UNIX shell prompt.

6.6 COMPILING STAND ALONE PROGRAMS

The procedures for compiling stand alone 'C' programs are summarized and described below. These procedures assume that the source file and the object file are placed in the appgen/src directory and that the executable program file is placed in the appgen/bin directory.

1. Modify make.XX.head.
2. Modify make.XX.tail.
3. Modify make.base.head if necessary.
4. Remake the makefile.
5. Execute make.

Steps one, two, and four above are necessary only when compiling new stand alone 'C' programs for the first time. The make.XX.head, make.XX.tail, and make.base.head files that appear in steps one through three are concatenated to the makefile in step four. If the application is new, then the make.XX.head file must be modified as described below.

STEP 1 - MODIFY MAKE.XX.HEAD

The first step in compiling a new stand alone program is to modify the make.XX.head file in the appgen directory. The relative pathname of the executable stand alone program should be added to the XXTARGS definition in the make.XX.head file. The form of the XXTARGS definition is shown below.

```
XXTARGS = bin/prog1 bin/prog2 bin/prog3\  
         bin/prog4
```

The above definition adds bin/prog1, bin/prog2, bin/prog3, and bin/prog4 to the list in the makefile of executable programs that must be created. The macro can be continued on another line by placing a backslash at the end of the line to be continued. The backslash MUST be followed immediately by a newline character and MUST NOT be at the end of the last line of the macro definition. The XXOBJ macro in the make.XX.head file does not need to be changed when compiling stand alone programs because the stand alone program object files are not linked to the Appgen program object files.

STEP 2 - MODIFY MAKE.XX.TAIL

The second step in compiling a new stand alone program is to add a new target entry to the make.XX.tail file. For example, if the new stand alone program is named prog.m.c in the appgen/src directory, then the target entry shown below should be added to the make.XX.tail file in the appgen directory.

```
bin/progm : src/progm.o $(LIBFILES)
           $(LFLAGS) -o $@ src/progm.o $(LIBS)
```

The list of dependencies MUST begin on the first line of the target entry and can be continued on subsequent lines by placing a backslash '\ ' at the end of the line to be continued. The backslash MUST be followed immediately by a newline character and MUST NOT be at the end of the last line of dependencies. Subsequent lines of the list of dependencies can contain either tabs or spaces at the beginning of the line for alignment. The macros \$(LIBFILES) and \$(LIBS) are defined in the makefile as object file libraries. The executable stand alone program bin/progm cannot be created until the object files src/progm.o, and \$(LIBFILES) are created.

The shell commands MUST begin on the line immediately following the list of dependencies and MUST be preceded on the line by a single tab character. The shell command to create the target is \$(LFLAGS), a macro defined in the makefile as a 'cc' command. The 'o' option to the 'cc' command =specifies '\$@' as the name of the executable program to be created. The characters '\$@' translate to the full target name, bin/progm. The executable program is created from linking the object files src/progm.o and \$(LIBS).

STEP 3 - MAKE.BASE.HEAD

The \$(XXTARGS) macro that is defined in the make.XX.head file must be added to the make.base.head file if the application is new. The \$(XXTARGS) macro should be added to the end of the definition of the TARGETS macro in the make.base.head file. The TARGETS macro in the make.base.head file can be continued on another line by placing a backslash character at the end of the line that is to be continued. The backslash MUST be followed immediately by a newline character and MUST NOT be at the end of the last line of the macro definition.

STEP 4 - MAKE NEWMAKE

The fourth step in compiling a stand alone program is to recreate the makefile. The instructions are the same as in step four for compiling subroutines. (See Section 6.5.)

STEP 5 - EXECUTE MAKE

The final step in compiling a stand alone program is to type 'make' at the UNIX shell prompt in the appgen directory. The stand alone program will be created in the appgen/bin directory. Any other source code files that have been modified will also be compiled. An example of a command to compile a single program named *prog* is:

```
'make bin/progm'
```

6.7 SUMMARY

Stand alone 'C' programs differ from subroutines in that stand alone programs are called as a Chain_Away () program and not through the use of the Appgen function processor. The compilation procedures for stand alone programs differ from subroutines in three respects:

1. Compiling stand alone programs does not require creation or modification of a XX.u text file.
2. The make.XX.tail file must be modified or created when compiling a new stand alone program.
3. The XXTARGS macro in the make.XX.head file must be modified or created for new stand alone programs whereas the XXOBJ macro must be modified or created for new subroutines.

STAND ALONE PROGRAMS ---

CHAPTER 7: 'C' LIBRARY ROUTINES

7.1 DATA BASE UTILITIES

The following table lists the values of the variable, *errno*, for various data base utilities. These utilities are documented on pages 7-2 thru 7-17. The variable *errno* is only defined if the routine returns one of the values shown in the Return Value column, indicating an error has occurred.

<u>Routine Name</u>	<u>Return Value</u>	<u>Value of errno</u>	<u>Meaning</u>
db_create	0	EACCES	the file is locked
	0	EMFILE	too many open files
	0	EPERM	corrupt file header
db_delete	-1	EACCES	file not locked by current process
	-1	EBADF	bad file
db_delrec	-1	EACCES	record not locked by current process
	-1	EBADF	bad file
db_lock	-1	EACCES	file has opens
	-1	EBADF	bad file
db_newrec	-1	EACCES	if record exists and is locked
	-1	EBADF	bad file
	1	EEXIST	if record exists and is not locked
	-1	EINVAL	null key
	-1	EPERM	file not open
db_open	0	EACCES	file locked
	0	EMFILE	too many open files
	0	EPERM	corrupt file header
db_read	-1	EACCES	if read with lock and record already locked
	-1	EBADF	bad file
	-1	ENOENT	not on file
	-1	EPERM	file not open
db_release	-1	EBADF	bad file
db_rewind	-1	EBADF	bad file
db_write	-1	EACCES	record not locked by current process
	-1	EBADF	bad file
db_unlock	-1	EBADF	bad file
readnext	0	EBADF	bad file
	0	EPERM	file not open

NAME

`db_create`, `db_delete` - file level data base utilities

SYNOPSIS

```
#include <db.h>
```

```
db_create( fname, hsize, trunc )  
char *fname;  
long hsize;  
int trunc;
```

```
db_delete( db )  
DB *db;
```

DESCRIPTION

`db_create` builds a new data base file if the named file does not exist. If the file exists and appears to be a DB file, `db_create` will open it and attempt to set the file lock. If this is successful and `trunc` is non-zero, the file will be rebuilt to be empty and with the specified hash size. In any case, the pointer returned refers to an open and locked data base file or is **NULL**, indicating an error.

`db_delete` removes the file associated with the indicated DB pointer. The file lock must be set.

RETURN VALUES

`db_create` returns a pointer to a data base file or **NULL** on error (see ERRORS below).

`db_delete` returns zero if successful, -1 on error (see ERRORS below).

ERRORS

`db_create`:
EACCES - file locked
EMFILE - too many open files
EPERM - corrupt file header

`db_delete`:
EACCES - file not locked by current process
EBADF - bad file

SEE ALSO

`dbopen`, `dblock`

NAME

`db_delrec` - remove a record from a file

SYNOPSIS

```
#include <db.h>
```

```
db_delrec( db )  
DB *db;
```

DESCRIPTION

If the **current record** in *db* is **locked**, it may be removed from the file by *db_delrec*. Any process which read that record (without locking) before the deletion will continue to use the old record until no such *db_reads* are active, at which time the space associated with the record will be available for reuse.

RETURN VALUE

db_delrec returns zero if successful, -1 when some error occurs (see ERRORS below).

ERRORS

EACCES - record not locked by current process
EBADF - bad file

SEE ALSO

`db_newrec`, `db_read`

NAME

db_lock, db_unlock - data base file locking

SYNOPSIS

```
#include <db.h>
```

```
db_lock( db )  
    DB *db;
```

```
db_unlock( db )  
    DB *db;
```

DESCRIPTION

db_lock sets the openlock on the file if no other open exists on the file. This lock prevents further opens. If the *db_lock* returns zero, you are guaranteed exclusive access to the entire file. *db_unlock* resets the openlock.

RETURN VALUE

These functions all return zero if all goes well; a non-zero value indicates some error (see ERRORS below).

ERRORS

EACCES - file has opens
EBADF - bad file

SEE ALSO

db_create

CAVEATS

The *datalock* call has been deleted from the system.

NAME

`db_newrec` - add a new record to a file

SYNOPSIS

```
#include <db.h>

db_newrec( db, key, size )
    DB *db;
    char *key;
    long size;
```

DESCRIPTION

`db_newrec` will create a record in the file associated with `db` having key `key` provided that no such record exists at the time of the call. The record will become the **current record** in the file, will be **locked**, and will consist of just the key (attribute zero). The `size` parameter provides optimization information if it is known exactly how large to make the record. Use `size` of zero if the ultimate size of the record is unknown at the time `db_newrec` is called.

The size of a record is the sum of all bytes in all fields (including the header and the key) plus one byte for each separator. There is an attribute separator after each attribute except the last one. There is a value separator after each value in a multi-valued attribute except the last value.

Use a size of zero to build the record with the RECSIZE parameter in the file header.

If a record already exists with the key specified, `db_newrec` reads the record, locks it, and returns one.

RETURN VALUE

`db_newrec` returns:
0 (zero) if successful,
1 if an existing record was read and locked (see ERRORS below),
-1 if some error occurs (see ERRORS below).

ERRORS

EACCES - if record exists and is locked
EBADF - bad file
EEXIST - (when 1 returned) if record exists and is not locked
EPERM - file not open

SEE ALSO

`db_write`, `db_read`, `dbar` in the Utilities section of the Appgen Development System Manual.

NAME

db_open, db_close - database file level interface

SYNOPSIS

```
#include <db.h>

DB *db_open( fname )
    char *fname;

DB *db_open( fname )
    char *fname;

db_close( db )
    DB *db;
```

DESCRIPTION

These functions implement the interface to existing data base files. *db_open* returns a pointer that may be used to access the data base file named in the call. *_db_open* is used by slave processes when the master process may have the file locked. It ignores the file lock and should be used carefully. If *fname* is NULL, a virtual file pointer is returned, which may be used to store and retrieve **DB**-structured data for which no system file need exist.

db_close ends the attachment to the file and no reference to the **DB** pointer is valid after it is closed.

RETURN VALUES

db_open returns NULL if the file cannot be open or is corrupted. See **ERRORS** below. *db_close* returns 0 if all goes well, -1 if you feed it a bad **DB***

ERRORS

- EACCES - file locked
- EMFILE - too many open files
- EPERM - corrupt file header

SEE ALSO

db_create

CAVEATS

All open **DB**'s must be closed before exiting.

NAME

`db_read` - get a record by key.

SYNOPSIS

```
#include <db.h>

db_read( db, key, lock )
    DB *db;
    char *key;
    int lock;
```

DESCRIPTION

`db_read` attempts to find and, if the parameter *lock* is non-zero, lock the record identified by *key*. The record, if found, becomes the **current record** in the file associated with *db*.

RETURN VALUE

If all goes well, `db_read` returns zero. A non-zero return value (-1) indicates that an error has occurred (see ERRORS below).

ERRORS

EACCES - if read with lock, record already locked
EBADF - bad file
ENOENT - not on file
EPERM - file not open

CAVEATS

If `db_read` returns an error indication, any previous **current record** on that file is released and the file is left without any **current record**.

SEE ALSO

`db_release`, `db_write`

NAME

`db_release` - unlock and release the current record.

SYNOPSIS

```
#include <db.h>
```

```
db_release( db )  
    DB*db;
```

DESCRIPTION

A file may have associated with it a **current record** and a certain amount of overhead is associated with maintaining that association. The record may also be **locked**, and the lock should not be maintained any longer than necessary. `db_release` cancels the association with the **current record** and unlocks that record if necessary.

RETURN VALUE

`db_release` returns zero unless some error occurs, in which case it returns -1 (see ERRORS below).

ERRORS

EBADF - bad file

CAVEATS

Prior to Release 1.5, `pdbwrite` released the record after writing it. Any code which relied on this behavior for correctness or friendliness should be examined and should perhaps use `db_release`.

SEE ALSO

`db_read`, `db_write`

NAME

`db_rewind`, `readnext` - sequential read facility

SYNOPSIS

```
#include <db.h>

db_rewind( db )
    regDB*db;

char *readnext( db, lock )
    regDB*db;
    int lock;
```

DESCRIPTION

db_rewind resets the sequential read pointer to the first record in the file, releasing any current record. It returns zero normally, -1 on error (see ERRORS below).

readnext performs a fast, 'random' read. It reads the physically next live data block in the file, optionally locking it. The physical order of the records within the file cannot be predicted, and thus the logical order in which *readnext* receives them is 'random'. Returns a pointer (into a static buffer) to the key of the record found, or NULL when the end of the file or an error is encountered (see ERRORS below).

CAVEATS

Intermixing *readnexts* and *db_writes* can have unpredictable results. When a record is written, its physical location in the file can change. So it is possible that a record could be read more than once or not at all by *readnext*. If you need to look at all of the records in a file and write some or all of them, use *readnext* to get a list of all the keys. Then read each record in the list using *db_read()*, make your changes and *db_write()* the record.

ERRORS

EBADF - bad file
EPERM - file not open

NAME

`db_stat` - return statistics about records and fields

SYNOPSIS

```
#include <appgen.h>
#include <vm.h>
#include <db.h>
#include <db_int.h>
```

```
db_stat( db, attr, val )
    DB*db;
    int attr,val;
```

DESCRIPTION

`db_stat` determines the size and composition of a field in the **current record** in `db`. If both `attr` and `val` are non-negative, the number of characters in the field they indicate is returned (including value marks if `val` is zero). If `val` is negative, the number of values present in attribute `attr` is returned. Using a negative value for `attr` will cause `db_stat` to return the number of attributes actually present in the record (including the key).

ERRORS

If the record, attribute, or value does not exist, `db_stat` returns -1.

NAME

`db_write` - make changes in the current record permanent.

SYNOPSIS

```
#include <db.h>
```

```
db_write( db )
        DB*db;
```

DESCRIPTION

If the **current record** in `db` is locked (was read with the lock parameter set or created by `db_newrec`) and has been modified (by *replace*, *insert*, or *delete*) then those changes are written to the file and become permanent and visible to any reads occurring after the write completes. Any processes which read the record before the write call will continue to use the old data.

RETURN VALUE

`db_write` returns zero normally, a non-zero (-1) value indicates an error (see ERRORS below).

ERRORS

EACCES - record not locked by current process
EBADF - bad file

CAVEATS

The current implementation of `db_write` does not release the **current record**. Prior to Release 1.5, `pdbwrite` did release the record after writing it.

SEE ALSO

`db_read`, `db_newrec`, `db_release`.

NAME

delete - delete a field

SYNOPSIS

```
#include <appgen.h>
#include <vm.h>
#include <db.h>
#include <db_int.h>
```

```
delete( db, attr, val )
      DB*db
      int attr,val;
```

DESCRIPTION

delete removes a field from the **current record**. It removes an attribute if *val* is zero or a value if *val* is greater than zero. All subsequent attributes or values are renumbered.

ERRORS

Returns zero normally, -1 on error. It is not an error to delete a non-existent field.

CAVEATS

The key of a record (attribute 0) may not be deleted.

NAME

extract - extract a field

SYNOPSIS

```
#include <appgen.h>
#include <vm.h>
#include <db.h>
#include <db_int.h>
```

```
extract( db, attr, val, buf, max )
    DB*db;
    int attr, val;
    char buf[];
    int max;
```

DESCRIPTION

extract retrieves a data field from the **current record** of *db*. *Attr* of zero indicates the key. A positive *val* indicates a particular value in a multi-valued attribute. A *val* of zero indicates an entire attribute. A nonexistent field is not an error and will appear to be null (zero length). The extracted string will always be null terminated, so at most (*max* - 1) characters will be placed in *buf*.

RETURN VALUE

The return value is the length of the field before any transaction to *max* - 1 characters, so overflow may be detected.

-1 is returned on error.

NAME

insert - insert a field

SYNOPSIS

```
#include <appgen.h>
#include <storage.h>
#include <vm.h>
#include <db.h>
#include <db_int.h>
```

```
insert( db, attr, val, buf )
    DB*db;
    int attr, val;
    char buf[];
```

DESCRIPTION

insert creates a new field in the **current record** of *db*. The new field is defined by *attr*, *val*, and will contain the data from *buf*. If such a field already exists it is renumbered one higher. Returns 0 normally, -1 on error.

CAVEATS

insert is not allowed to operate on the key.

If *buf* is empty and is inserted before the first value of an empty attribute, nothing happens. This differs from the behavior of previous releases (prior to R1.6). Previous releases would have placed a value mark in the attribute which is interpreted as two null values.

NAME

replace - replace a field

SYNOPSIS

```
#include <appgen.h>
#include <vm.h>
#include <db.h>
#include <db_int.h>
```

```
replace( db, attr, val, buf )
    DB*db;
    int attr, val;
    char buf[];
```

DESCRIPTION

replace modifies a data field. It places the string in *buf* into the field specified by *attr* and *val* in the **current record** in *db*. Any fields between the end of the record (or attribute) and the specified field will be created. Returns 0 normally, -1 on error.

CAVEATS

It is legal to *replace* the key of a record; however, the record will then be unlocked and may not be written.

NAME

`char_extract`, `int_extract`, `long_extract`, `str_extract`, `sstr_extract` - extract and convert a field

SYNOPSIS

```
#include <appgen.h>
#include <vm.h>
#include <db.h>

char char_extract( db_p, att, val )
    DB*db_p;
    int att;
    int val;

int int_extract( db_p, att, val )
    int DB*db_p;
    int att;
    int val;

long long_extract( db_p, att, val )
    long DB*db_p;
    int att;
    int val;

char *str_extract( db_p, att, val )
    DB*db_p;
    int att;
    int val;

char *sstr_extract( db_p, att, val )
    DB*db_p;
    int att;
    int val;
```

DESCRIPTION

These utilities are used by at least *Printem* and *Postem*. Although these routines are rather trivial, their frequent use in various programs warrants their inclusion here.

int_extract extracts the proper attribute and converts that string to an integer;

long_extract extracts the proper attribute and converts that string to a long integer;

char_extract extracts the proper attribute and returns the first character of the attribute string.

str_extract extracts a copy of the proper attribute into a malloc 'ed buffer and returns a pointer to the buffer. A free of the buffer space should be executed when it is no longer needed.

sstr_extract extracts the proper attribute into a static buffer and returns a pointer to the buffer. Any subsequent calls to *sstr_extract* overwrite the contents of the buffer.

7.2 SCREEN UTILITIES

NAME

Bell - Ring bell on terminal.

SYNOPSIS

```
#include <appgen.h>  
#include <errno.h>  
#include <ctype.h>  
#include <signal.h>
```

```
Bell( n )  
    int n;
```

DESCRIPTION

Bell signals the user via audible or visible bell. Arguments should be zero for an audible bell or non-zero for a visible bell.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`.
If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

NAME

`clrflld` - erase a field on terminal screen

SYNOPSIS

```
#include <appgen.h>  
#include <errno.h>  
#include <ctype.h>  
#include <signal.h>
```

```
clrflld( row, col, width )  
    regint row;  
    regint col;  
    regint width;
```

DESCRIPTION

clrflld erases a field on the CRT or display screen at the specified *row*, of the specified *width*, beginning in column *col*. If -1 is used as the row or column number, *clrflld* uses the current row or column number of the screen cursor.

CAVEATS

If your computer is an AT&T PC7300, in order to use any of the screen input and output routines, you must also include `sys/window.h`. If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`clrscr`, `S_exit`

NAME

`clrscr` - clear entire screen

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

`clrscr()`

DESCRIPTION

clrscr clears the CRT or display screen.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`.
If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`S_exit`

NAME

dispdata - format and display a string

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
dispdata( r, c, f, d, w )
    int c, r, w;
    char *f, *d;
```

```
v_dispdata( r, c, f, d, w, mode )
    int c, r, w, mode;
    char *f, *d;
```

DESCRIPTION

dispdata displays data *d* in format *f* on row *r*, beginning in column *c*, for width *w*. The formats accepted by *dispdata* are found in Appendix D of the Appgen Reference Manual. *Dispdata* uses *formdata* to format the data and *putfld* to display the formatted data on the display device.

v_dispdata works the same as *dispdata* with the added argument *mode*. *mode* is the video attribute (0 thru 7 - see page 27-1 in the Appgen Development System Manual) that determines how the data *d* will appear on the screen.

SEE ALSO

formdata, *putfld*, *S_exit*

NAME

displaym - display message on screen

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
displaym( msg, prompt, resp )
char *resp;
char *msg;
int prompt;
```

DESCRIPTION

displaym displays the message text *msg* on the last row of the screen using *putfld*. If *prompt* is non-zero, a response is prompted for using *getfld* (at the current cursor position) and is then placed in *resp*.

SEE ALSO

formdata, putfld, S_exit

NAME

`draw_box` - format and display a box or line

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

`draw_box(top row, bottom row, beginning column, width, vmode)`

`clr_box(top row, bottom row, beginning column, width)`

DESCRIPTION

`draw_box` displays a box (or line) using the terminal's graphic line drawing characters as defined in the Appgen Terminfo data base. The box will extend from *top row* to *bottom row*, from *beginning column* to *width plus beginning column*. It will display using video attribute *vmode* (0 through 7 - see Chapter 27.1 Video Attributes in the Appgen Development System Manual). `clr_box` erases a previously drawn box. (`clr_box` erases the character positions of a box with the same coordinates regardless of the existence of a box.)

To draw a horizontal line, set the *top row* and *bottom row* to the same row.

To draw a vertical line, set the *width* to zero.

SEE ALSO

Appgen Development System Manual Operator's Reference - Appgen Terminal Information Data Base.

NAME

getdata - prompts for operator input with default value and checking

SYNOPSIS

```
getdata( target, default, row, column, format, max, min, req, mode )  
char *target, *default, *format, *max, *max, req;  
int row, column, mode;
```

DESCRIPTION

getdata prompts the operator for input at *row* and *column*, first displaying *default* at *row* and *column* using the video attribute *mode*. If the operator simply hits <return>, the default value is stored in *target*. If the operator makes an entry, it is checked against *format*; if it is a valid entry, it is redisplayed using *format*. The entry is also checked against *max* and *min*. If the entry is invalid, the operator will be reprompted until the entry is acceptable. If *req* is 'Y', an entry is required. The final entry is stored in *target*.

NAME

`getfld` - take input of given width at position on screen

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>

getfld( r, c, w, str )
    int r, c;
    int w;
    char *str

v_getfld( r, c, w, str, mode )
    int r, c, w, mode;
    char *str;
```

DESCRIPTION

`getfld` is the Appgen input editor. It prompts the user for input of length `w` on row `r` beginning with column `c`. If the row or column is negative, `getfld` uses the current row or column position of the screen cursor. The Appgen Data Editor commands are implemented in `getfld`. If `str` is non-null, it is placed at `r` and `c` and used as a default value. The input, if any, will be placed in `str`. A pointer to `str` is returned. Most of the work of `getfld` is actually done by `padget`.

`v_getfld` works the same as `getfld` with the added argument `mode`. `mode` is the video attribute (0 thru 7 - see page 27-1 in the Appgen Development System Manual) that determines how the data `str` will appear on the screen.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`. If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`putfld`, `padget`, `S_exit`, Appgen Development System Manual - Appendix C - Appgen Data Editor.

NAME

padget - display a field and prompt for input

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
char *padget( r, c, w, str, ps )
    regint r;
    regint c;
    int w;
    regchar *str;
    char *ps;
```

```
v_padget( r, c, w, str, ps, mode )
    int w, mode;
    char *ps;
```

DESCRIPTION

padget displays the string *ps* at row *r* and column *c* up to the $\text{MIN}(\text{strlen}(\text{str}), w)$. It positions the cursor at row *r* and column *c* again and prompts the user for input of length *w*. The input is placed in *str* and a pointer to *str* returned. If *r* or *c* is negative, the current row or column position of the cursor is used.

v_padget works the same as *padget* with the added argument *mode*. *mode* is the video attribute (0 thru 7 - see page 27-1 in the Appgen Development System Manual) that determines how the data *str* will appear on the screen.

CAVEATS

If your computer is an AT&T PC7300, you must also include *sys/window.h*. If your computer has XENIX version 3.0, you must also include *sys/ioctl.h*.

SEE ALSO

S_exit

NAME

putfld - display a string of a given width at a given position

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
putfld( row, col, width, str )
      reg      int      row;
      reg      int      col;
      reg      int      width;
      reg      int      *str;
```

```
v_putfld( r, c, w, str, mode )
      int r, c, w, str, mode;
```

DESCRIPTION

putfld prints *str*, padded to the specified *width* on the terminal screen on the *row* specified, and beginning at column *col*. If -1 is used as the row or column number, *putfld* uses the current row or column number of the screen cursor.

v_putfld works the same as *putfld* with the added argument *mode*. *mode* is the video attribute (0 thru 7 - see page 27-1 in the Appgen Development System Manual) that determines how the data *str* will appear on the screen.

CAVEATS

If your computer is an AT&T PC7300, you must also include *sys/window.h*. If your computer has XENIX version 3.0, you must also include *sys/ioctl.h*.

SEE ALSO

getfld, S_exit

NAME

title - display the title information at top of the screen

SYNOPSIS

```
title( prognum, installname, progname, sysin )
      char      *prognum
      char      *installname
      char      *progname
      char      *sysin
```

DESCRIPTION

title displays the following information on row zero of the screen: the PDEF number, the installation name, and the program name.

NOTE: *sysin* is no longer used in *title* but is left in the function for consistency.

NAME

`S_cook`, `S_uncook` - restore original `ioctl` state or set raw term modes

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

`S_cook()`

`S_uncook()`

DESCRIPTION

`S_cook` restores the original `ioctl` state and should be performed before executing shell commands. The `ioctl()` command performed by `S_cook` sets the cooked terminal modes. The cooked modes allow input and output processing with special settings such as ERASE, KILL, INTR, QUIT, and EOT. The original `ioctl` state is required when executing commands that suspend the current process such as `fork`, `exec`, or `sh`.

`S_uncook` should only be performed after executing an `S_cook`. The `ioctl` command executed by `S_uncook` sets the raw terminal modes. The raw modes allow input and output processing without special settings such as ERASE, KILL, INTR, QUIT, and EOT. The raw terminal modes are required when executing screen control routines from the `appgen.a` library such as `putfld` and `getfld`.

`S_uncook` is called automatically by the screen package initialization. The screen package initialization is called automatically the first time a process uses any screen package routine. `S_cook` is called automatically by `S_exit`.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`. If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`ioctl(3)`, `system(3)` in the UNIX/XENIX manual, `S_exit`, `putfld`, `getfld`

NAME

`S_exit` - deinitialize screen handler routines

SYNOPSIS

```
#include <appgen.h>  
#include <errno.h>  
#include <ctype.h>  
#include <signal.h>
```

`S_exit()`

DESCRIPTION

`S_exit` should be called before any possible exit from user written code in which any of the screen package routines, such as `putfld`, `getfld`, `S_read`, `S_flush`, `S_refresh`, etc. are called. Any of the screen package routines will perform the screen package initialization if the initialization has not already been performed. The deinitialization functions must be performed by calling `S_exit`.

`S_exit` calls `S_cook`.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`.
If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`S_cook`, `putfld`, `getfld`, `S_read`, `S_flush`

NAME

`S_flush` - force all queued output to the screen

SYNOPSIS

```
#include <appgen.h>  
#include <errno.h>  
#include <ctype.h>  
#include <signal.h>
```

```
S_flush()
```

DESCRIPTION

S_flush writes all queued data to the screen. The data queued is normally written to the screen at intervals that minimize the overhead involved in writing to the screen. Use *S_flush* to force output to the screen without waiting.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`.
If your computer has XENIX version 3.0, you must also include `sys/ioctl.h`.

SEE ALSO

`putfld`, `S_exit`

NAME

`S_read` - get the next character

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
char S_read()
```

DESCRIPTION

`S_read` gets the next character from the `S_read` buffer. The `S_read` buffer, when empty, is refilled from the system input buffer. If the system input buffer is empty, `S_read` waits until data is typed in at the keyboard. `S_read` is used to read a single character that was input at the terminal. To read several characters at a time, use the `getfld` function. The `S_read` function can be used in the creation of a data editor such as `getfld`.

RETURN VALUE

Returns a character.

CAVEATS

If your computer is an AT&T PC7300, you must also include `sys/window.h`. If your computer has XENIX version 3.0, you must also include `sys.ioctl.h`.

SEE ALSO

`getfld`, `S_exit`

NAME

S_refresh - redraw the screen

SYNOPSIS

```
#include <appgen.h>
#include <errno.h>
#include <ctype.h>
#include <signal.h>
```

```
S_refresh()
```

DESCRIPTION

S_refresh redraws the display to reflect the current state of the screen as known to the screen package. This may be necessary if there has been non-Appgen screen package terminal input/output such as the performance of a *printf* statement.

CAVEATS

If your computer is an AT&T PC7300, you must also include *sys/window.h*.
If your computer has XENIX version 3.0, you must also include *sys/ioctl.h*.

SEE ALSO

S_exit

7.3 WINDOW UTILITIES

NAME

`w_clrfld` - erase a field on the current window

SYNOPSIS

```
w_clrfld( r, c, w,m )  
    int   r;  
    int   c;  
    int   w;  
    int   m;
```

DESCRIPTION

`w_clrfld` clears a field to the specified width 'w' in the current window at row 'r' and beginning at column 'c' in video mode 'm'. If -1 is used as the row or column number, `w_clrfld` uses the current row or column of the screen cursor.

Note: The current window coordinates will be added to the 'r' and 'c' specified to determine the absolute address on the screen image.

NAME

`w_clrscr` - clears the current window screen image

SYNOPSIS

`w_clrscr()`

DESCRIPTION

`w_clrscr` clears the entire area within the current open window.

NAME

W_exit - remove the current window image and restore the previous screen

SYNOPSIS

W_exit()

DESCRIPTION

W_exit will close the current window and redraw all of the portions that it overlays.

NAME

w_getdata - display a field and prompt for input on the current window

SYNOPSIS

```
w_getdata ( target, dflt, r, c, format, max, min, req, m )
    char *target;
    char *dflt;
    int r;
    int c;
    char *format;
    char *max;
    char *min;
    char req;
    int mode;
```

DESCRIPTION

w_getdata is a subroutine that prompts the user for information in the current window then redisplay that information in a specified format.

target : the target string in which the verified and formatted input will be stored
c : the column at which input will begin
r : the row at which input will begin
format : the format to which input will be converted for storage and redisplay; possible formats are:
A: an alpha-numeric string
D: a date in the form mmddy or mmdd
Nx,yz: a numeric string with an optional minus sign, up to x digits before the decimal point, y digits after the decimal point, in a field width of z digits. If a period replaces the comma, no commas will be inserted on redisplay. Note that only a string of digits with an optional minus sign may be input by the operator. Note also that x and y are single digit fields and z is a two digit field.
\$x,yz: identical to the N format except that a dollar sign is printed on output
S: a 9 digit social security number
P: a 7 or 10 digit telephone number
Y: a single character boolean with possible absolute address on the screen image.

- req: a character containing Y or N to specify whether or not a response is required
- default: either a null string (if no default is specified) or a string containing the default just as if the operator had typed it in
- max: the maximum value the response is to have
- min: the minimum value the response is to have

Max and Min are digit strings which are the maximum and minimum string lengths for "A" formats, and the maximum and minimum values for "N" and "\$" formats (in the same format as the input would be).

- m: the video mode that is used for display

Note: The current window coordinates will be added to the 'r' and 'c' specified to determine the absolute address on the screen image.

NAME

`w_getfld` - take input of a given width on the current window

SYNOPSIS

```
char *w_getfld( r, c, w, s, m )
    int    r,    c;
    int    w;
    char   *s;
    int    m;
```

DESCRIPTION

`w_getfld` prompts the user for input of length 'w' on row 'r' beginning with column 'c' in video mode 'm'. If -1 is used as the row or column number, `w_getfld` uses the current row or column of the screen cursor.

Note: The current window coordinates will be added to the 'r' and 'c' specified to determine the absolute address on the screen image.

NAME

W_init - initialize and open an APPGEN window

SYNOPSIS

```
W_init( beg_row, end_row, col, width, mode, desc )  
    int    beg_row;  
    int    end_row;  
    int    col;  
    int    width;  
    int    mode;  
    char   *desc;
```

DESCRIPTION

W_init will initialize and open a window. The window will appear on the screen with *beg_row*, *end_row*, *col* and (*col* + *width*) as window coordinates. The window border will appear outside of the window coordinates in the form of asterisks. The actual work area will be inclusive of the coordinates.

NAME

`w_padget` - display a field and prompt for input on the current window

SYNOPSIS

```
char *w_padget(r, c, w, s, ps, m)
    int    r;
    int    c;
    int    w;
    char   *s;
    char   *ps;
    int    m;
```

DESCRIPTION

`w_padget` displays the string 'ps' at row 'r' and column 'c' up to the MIN(slen(s),w). It positions the cursor at row 'r' and column 'c' again and prompts the user for input of length 'w' in video mode 'm'. If -1 is used as the row or column number, `w_padget` uses the current row or column of the screen cursor.

Note: The current window coordinates will be added to the 'r' and 'c' specified to determine the absolute address on the screen image.

NAME

`w_putfld` - display the string on the current window

SYNOPSIS

```
w_putfld( r, c, w, str, m )  
    int    r;  
    int    c;  
    int    w;  
    char   *str;  
    int    m;
```

DESCRIPTION

`w_putfld` prints 'str', padded to the specified width in the current window at row 'r' and beginning at column 'c' in video mode 'm'. If -1 is used as the row or column number, `w_putfld` uses the current row or column of the screen cursor.

Note: The current window coordinates will be added to the 'r' and 'c' specified to determine the absolute address on the screen image.

7.4 FUNCTION EVALUATOR

NAME

F_string - return pointer to a function

SYNOPSIS

```
F_string( fstr )  
    char *fstr;
```

DESCRIPTION

F_string returns a pointer to a C function that returns the characters in the string *fstr* one at a time. It is used in conjunction with the other function evaluator calls described below. *fstr* is a string containing an Appgen function as described in the Appgen Development System Manual.

NAME

Func_Eval - evaluate a function

SYNOPSIS

```
#include <time.h>
#include <appgen.h>
#include <func.h>
#include <ctype.h>
#include <Func_Eval.h>
#include <assert.h>
#include <hp.h>
```

```
Func_Eval( func, dest )
    char (*func) ();
    char *dest;
```

DESCRIPTION

Func_Eval evaluates an Appgen function of the form described in the Appgen Development System Manual. *Func_Eval* places the results of the function in *dest*. *func* is a pointer to a function which returns a char. Use *F_string* described above.

CAVEATS

The results are indeterminate if the function is syntactically or semantically incorrect.

EXAMPLE

```
scopy( fstr, "A(1,4)" );
Func_eval( F_string(fstr), dest );
```

SEE ALSO

Func_Parse and Func_Exec

NAME

Func_Parse - parse a function

SYNOPSIS

```
#include <time.h>
#include <appgen.h>
#include <func.h>
#include <ctype.h>
#include <Func_Eval.h>
#include <assert.h>
#include <hp.h>
```

```
P_NODE Func_Parse( func )
char ( *func ) ();
```

DESCRIPTION

Func_Parse used in conjunction with *Func_Exec* can be more efficient than *Func_Eval* for repetitive operations since the function need only be parsed once. *Func_Parse* returns a non-null P_NODE if the function being parsed is valid; otherwise a null is returned.

EXAMPLE

```
strcpy( fstr,"A(1,4)" );
Func_Parse( F_string( fstr ), dest );
```

SEE ALSO

Func_Exec and Func_Eval

NAME

Func_Exec - execute a function

SYNOPSIS

```
#include <time.h>
#include <appgen.h>
#include <func.h>
#include <ctype.h>
#include <Func_Eval.h>
#include <assert.h>
#include <hp.h>
```

```
Func_Exec( ptr, dest )
    P_NODE *ptr;
    char *dest;
```

DESCRIPTION

Func_Exec executes a function pointed to by the P_NODE referenced by *ptr*. This function must have been previously parsed by *Func_Parse*. The result of the function is placed in *dest*.

SEE ALSO

Func_Parse and Func_Eval

7.5 DATA CONVERSION UTILITIES

NAME

atoi, atol - string to integer conversions

SYNOPSIS

```
#include <appgen.h>  
#include <ctype.h>
```

```
int atoi( s )  
    reg char *s;
```

```
long atol( s )  
    reg char *s;
```

DESCRIPTION

Convert the string *s* to an integer or a long ignoring beginning white space.

RETURN VALUE

Returns zero if *s* is completely non-numeric. It will convert any beginning numeric characters (ignoring white space) up to the first non-numeric character.

SEE ALSO

Section 7.6 TYPE

NAME

date_ck - verify that date is valid date

SYNOPSIS

```
#include <appgen.h>  
#include <ctype.h>  
  
int date_ck( datebuff )  
          char *datebuff;
```

DESCRIPTION

date_ck verifies that *datebuff* is a valid date. The date stored in *datebuff* is expected to be in the format mmddyy or mmddyyyy. If the UNIX environmental variable AGINIT contains the string AGDATE = "1", *datebuff* is expected to be in the format ddmmyy or ddmmyyyy (see *International Dates* in the Utilities section of the Appgen Development Reference Manual).

RETURN VALUE

date_ck returns a 0 if and only if the date is a valid calendar date, -1 otherwise.

SEE ALSO

date_in, date_out, date_4out

NAME

date_in - convert a date to the Julian internal date

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>

char *date_in( buff )
char *datebuff;
```

DESCRIPTION

date_in converts the date in *datebuff* to an internal Julian date. The internal date is an integer representing the number of days (plus or minus) since 12/31/67 with 12/31/67 being day 0. *date_in* returns a pointer to a character string which contains the digits of the internal date. The date stored in *datebuff* is expected to be in the format mmddyy or mmddyymm. If the UNIX environmental variable AGINIT contains the string, AGDATE = "I", then *datebuff* is expected to be in the format ddmmyyyy (see *International Dates* in Appendix 'F' section of the Appgen Development System Manual).

RETURN VALUE

date_in returns the internal Julian date for any string which can be interpreted as a date. It returns "0" for anything else.

SEE ALSO

date_ck, date_out, date_4out

NAME

date_out - convert an internal Julian date to the external date format

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>

char *date_out( sdays )
char *sdays;
```

DESCRIPTION

date_out converts the character string *sdays*, representing an internal Julian date to the external date format, *mmddy*. If the UNIX environmental variable *AGINIT* contains the string, *AGDATE = "I"*, then the external date will be in the format *ddmmyy* (see *International Dates* in Appendix 'F' of the Appgen Development System Manual).

RETURN VALUE

date_out returns a pointer to a character string containing the external date format.

SEE ALSO

date_ck, date_in, date_4out

NAME

date_4out - convert an internal Julian date to the external date format

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>

char *date_4out( sdays )
char *sdays;
```

DESCRIPTION

date_4out converts the character string, *sdays*, representing an internal Julian date to the external date format, mmddyyyy. If the UNIX environmental variable AGINIT contains the string, AGDATE = "I", then the external date will be in the format ddmmyyyy (see *International Dates* in Appendix 'F' of the Appgen Development System Manual).

RETURN VALUE

date_4out returns a pointer to a character string containing the external date.

SEE ALSO

date_ck, date_in, date_out

NAME

toascii, *tolower*, *toupper*, *Lower*, *Upper* - character translation

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>
```

```
int toascii( c )
    int c;
```

```
int tolower( c )
    int c;
```

```
int toupper( c )
    int c;
```

```
*char Lower( instr )
    char * instr;
```

```
*char Upper (instr )
    char * instr;
```

DESCRIPTION

toascii returns its argument with all bits turned off that are not part of a standard ASCII character.

tolower and *toupper* have as domain the range of *getc*: the integers from -1 through 255. If the argument of *tolower* represents an uppercase letter, the result is the corresponding lowercase letter. If the argument of *toupper* represents a lowercase letter, the result is the corresponding uppercase letter. All other arguments in the domain are returned unchanged.

Lower and *Upper* apply *tolower* and *toupper* respectively, to all characters in the string, *instr*. The argument is returned for convenience.

CAVEATS

toascii, *tolower*, and *toupper* are macros. One should take care not to pass them anything as arguments which might have side effects (such as a pointer to a pointer argument).

SEE ALSO

Section 7.6 TYPE

7.6 TYPE - CHARACTER CLASSIFICATION

NAME

isalnum, *isalpha*, *isascii*, *isctrl*, *isdigit*, *isgraph*, *islower*, *ismeta*, *isprint*, *ispunct*, *isspace*, *isupper* - character classification

SYNOPSIS

```
#include <ctype.h>
```

```
int isalnum( c )  
    int c;
```

```
isalpha( c )  
    int c;
```

```
    .  
    .  
    .
```

DESCRIPTION

A portable ctype (see your UNIX manual) implementation. This one conforms to the SYS3 conventions, with the exception that all macros are defined over all input values. These macros classify ASCII-coded integer values by table lookup and/or range checking. Each is a predicate returning nonzero for **true** or zero for **false**.

isalnum returns **true** if *c* is an alphanumeric (letters and digits).

isalpha returns **true** if *c* is a letter.

isascii returns **true** if *c* is an ASCII character, code less than 0200.

isctrl returns **true** if *c* is a delete character (0177) or ordinary control character (less than 040).

isdigit returns **true** if *c* is a digit [0-9].

isgraph returns **true** if *c* is a printing character, like *isprint* except false for space.

islower returns **true** if *c* is a lowercase letter.

ismeta returns **true** if *c* is a 'meta' character - bit 7 on (codes from 0200 to 0377).

isprint returns **true** if *c* is a printing character, code 040 (space) through 0176 (tilde).

ispunct returns **true** if *c* is a punctuation character (neither control nor alphanumeric).

isspace returns **true** if *c* is a space, tab, carriage return, new-line, vertical tab, or form feed.

isupper returns **true** if *c* is an uppercase letter.

7.7 HIGH PRECISION MATH UTILITIES

NAME

hp_exp, *hp_log* - natural exponential and logarithmic functions

SYNOPSIS

```
#include <hp.h>

hpf hp_exp( x, dest )
    hp x;
    hp dest;

hpf hp_log( x, dest )
    hp x;
    hp dest;
```

DESCRIPTION

These routines implement natural logarithmic and exponential functions:

hp_exp - raises the base of the natural logarithms **e** (approximately 2.71828...) to the *x*th power, placing the result in *dest* and returning a pointer to *dest*.

hp_log - computes the natural logarithm (log base **e**) of *x*, placing the result in *dest* and returning a pointer to *dest*.

DIAGNOSTICS

Overflow and underflow are not detected; in these conditions unexpected results will be returned.

BUGS

In the current implementation *hp_log* does not recognize the sign of *x* (ie. $\ln -30$ is equivalent to $\ln 30$). Additionally, $\ln 0$ evaluates to zero.

NAME

`s_to_hp`, `hp_to_s`, `l_to_hp`, `hp_to_l`, `a_to_hp`, `hp_to_a` - input and output conversions for the high precision math package.

SYNOPSIS

```
#include <hp.h>

hpf s_to_hp( s, dest )
    short s;
    hp dest;

hpf l_to_hp( l, dest )
    long l;
    hp dest;

hpf a_to_hp( str, dest )
    char *str;
    hp dest;

int hp_to_s( h )
    hp h;

long hp_to_l( h )
    hp h;

char *hp_to_a( h, str )
    hp h;
    char *str;
```

DESCRIPTION

These functions implement the input and output conversions necessary to integrate the high precision math modules into application software. The routines to convert numbers to the high precision representation are `s_to_hp`, `l_to_hp`, and `a_to_hp`. The routines to convert high precision numbers back to normal machine representation are `hp_to_s`, `hp_to_l`, and `hp_to_a`.

`s_to_hp` converts the short integer argument `s` to a high precision number, leaves the result in `dest`, and returns a pointer to `dest`.

`l_to_hp` converts the long integer argument `l` to a high precision number, leaves the result in `dest`, and returns a pointer to `dest`.

`a_to_hp` converts the string pointed to by the argument `str` to a high precision number, leaves the result in `dest`, and returns a pointer to `dest`.

hp_to_s converts the high precision argument *h* to a short integer and returns that integer. Any fractional portion of the argument is ignored (i.e. truncated).

hp_to_l converts the high precision argument *h* to a long integer and returns that integer. Any fractional portion of the argument is ignored.

hp_to_a converts the high precision argument *h* to a character string, leaves the result at the argument *str*, and returns the argument *str*. The string buffer must be at least of length 30.

DIAGNOSTICS

Overflow and underflow are not detected; in these conditions unexpected results will be returned.

CAVEATS

In the current implementation, *hp_to_a* only returns the first eight (8) fractional decimal places and requires string buffer of at least 30 characters.

NAME

`hp_add`, `hp_sub`, `hp_mul`, `hp_div`, `hp_mod`, `hp_pwr` - high precision numerical manipulation (`hp_onto` `hp_yielding` `hp`).

SYNOPSIS

```
#include <hp.h>

hpf hp_add( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;

hpf hp_sub( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;

hpf hp_mul( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;

hpf hp_div( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;

hpf hp_mod( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;

hpf hp_pwr( arg1, arg2, dest )
    hp arg1, arg2;
    hp dest;
```

DESCRIPTION

These functions implement the high precision mathematical manipulations needed for accounting and related applications. All perform the specified operation *arg2* onto *arg1*, placing the result in *dest* and returning a pointer to *dest*, with the exception of *hp_mod* which returns a pointer to the modulo *arg2* of *arg1* and places the result of *arg1* divided by *arg2* in *dest*.

DIAGNOSTICS

Overflow and underflow are not detected.

BUGS

In the current implementation *hp_pwr* does not recognize the sign of any value raised to a non-integral power (i.e. -2 raised to the .5 will be equivalent to +2 raised to the .5) and yields only non-negative roots.

NAME

`hp_adds`, `hp_subs`, `hp_muls`, `hp_divs`, `hp_mods`, `hp_pwrs` - high precision numerical manipulation (short onto `hp` yielding `hp`).

SYNOPSIS

```
#include <hp.h>

hpf hp_adds( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;

hpf hp_subs( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;

hpf hp_muls( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;

hpf hp_divs( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;

short hp_mods( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;

hpf hp_pwrs( arg1, arg2, dest )
    hp arg1;
    short arg2;
    hp dest;
```

DESCRIPTION

These functions implement the high precision mathematical manipulations needed for accounting and related applications. All perform the specified operation *arg2* onto *arg1*, placing the result in *dest* and returning a pointer to *dest*, with the exception of *hp_mod* which returns the modulo *arg2* of *arg1* and places the result of *arg1* divided by *arg2* in *dest*.

DIAGNOSTICS

Overflow and underflow are not detected.

NAME

`hp_round`, `hp_copy`, `hp_negate`, `hp_cmp`, `hp_isneg`, `hp_issshort`, `hp_islong`, `hp_hasfract`, `hp_int` - utilities available for working with high precision (hp) numbers.

SYNOPSIS

```
#include <hp.h>

hpf hp_round( arg1, dplace )
    hp arg1;
    int dplace;

hpf hp_copy( arg1, dest )
    hp arg1;
    hp dest;

hpf hp_negate( arg1, dest )
    hp arg1;
    hp dest;

hp_cmp( arg1, arg2 )
    hp arg1;
    hp arg2;

hp_isneg( arg1 )
    hp arg1;

hp_issshort( arg1 )
    hp arg1;

hp_islong( arg1 )
    hp arg1;

hp_hasfract( arg1 )
    hp arg1;

hp_int( arg1 )
    hp arg1;
```

DESCRIPTION

These routines perform miscellaneous operations or comparisons on the hp's.

hp_round - rounds *arg1* to *dplace* decimal places (18 to -9) and returns a pointer to *arg1*.

hp_copy - copies *arg1* into *dest* and returns a pointer to *dest*.

hp_negate - negates *arg1* and places result into *dest*, returns a pointer to *dest*.

hp_cmp - arithmetic comparison between *arg1* and *arg2*, returns (-1) if *arg1* is less than *arg2*, (0) if *arg1* equals *arg2*, and (1) if *arg1* is greater than *arg2*. If *arg2* is a null pointer, then *arg1* is compared to zero.

hp_isneg - returns (1) if *arg1* is negative, (0) otherwise.

hp_isshort - returns (1) if the integral portion of *arg1* will fit into a short, (0) if it will not.

hp_islong - returns (1) if the integral portion of *arg1* will fit into a long, (0) if it will not.

hp_hasfract - returns (1) if *arg1* has any fractional digits, (0) if it does not.

hp_int truncates any fractional digits.

SEE ALSO

hpmath(3), *hpmaths(3)*, *hpexp(3)*, *hputil(3)*.

7.8 STRINGS - STRINGS UTILITIES

NAME

dechar, derep, detab, scat, sncat, scmp, strcmp, strncmp, scpy, sncpy, slen, schr, srchr, sbrk, strtok, strtok - string operations.

SYNOPSIS

```
char *dechar( str, c )
      char *str;
      char c;

char *derep( str, c, r )
      char *str;
      char c;
      char r;

char *detab( s1 )
      reg char *s1

char *scat( s1, s2 )
      char *s1, *s2;

char *sncat( s1, s2, n )
      char *s1, *s2;
      int n;

int scmp( s1, s2 )
      char *s1, *s2;

int strcmp( s1, s2 )
      char *s1, *s2;

int strncmp( s1, s2, n )
      char *s1, *s2;
      int n;

char *scopy( s1, s2 )
      char *s1, *s2;

char *sncpy( s1, s2, n )
      char *s1, *s2;
      int n;

int slen( s )
      char *s;

char *schr( s, c )
      char *s, c;
```

```
char *srchr( s, c )
           char *s, c;

char *spbrk( s1, s2 )
           char *s1, *s2;

char *stok( s1, s2 )
           char *s1, *s2;

char *strmtok( s1, s2 )
           char *s1, *s2;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

dechar creates a new string that is the same as *str* with the exception that all characters in *str* matching *c* are absent. *dechar* returns a pointer to a character buffer containing the new string. *dechar* does not change *str* at all.

derep creates a new string that is the same as *str* with the exception that all characters in *str* matching *c* are replaced with *r*. *derep* returns a pointer to a character buffer containing the new string. *derep* does not change *str* at all.

detab replaces tabs in a string with up to eight spaces and returns a pointer to a detabbed buffer.

scat appends a copy of string *s2* to the end of string *s1* and returns a pointer to the null-terminated result.

scmp compares its arguments and returns an integer greater than, equal to, or less than zero, according as *s1* is lexicographically greater than, equal to, or less than *s2*. *sncmp* makes the same comparison but looks at most *n* characters. *strccmp* makes the same comparison as *scmp* except that uppercase letters are mapped to lowercase.

scopy copies string *s2* to *s1*, stopping after the null character has been moved. *sncpy* copies exactly *n* characters, if necessary truncating *s2* or adding null characters to *s1*. The target will always be null-terminated. Both return *s1*.

slen returns the number of non-null characters in *s*.

schr returns a pointer to the first (last) occurrence of character *c* in string *s* or NULL if *c* does not occur in the string.

spbrk returns a pointer to the first occurrence in string *s1* of any character from string *s2* or NULL if no character from *s2* exists in *s1*.

stok considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token and will have written a **NULL** character into *s1* immediately following the returned token. Subsequent calls with zero for the first argument will work through the string *s1* in this way until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, **NULL** is returned.

strmtok is similar except it does not collapse white space and thus may return null (zero length) tokens.

BUGS

All string movement is performed character by character starting at the left. Thus, overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.

7.9 OTHER UTILITIES

NAME

Chain_Away - chain to new program preserving the calling convention

SYNOPSIS

```
#include <appgen.h>  
  
Chain_Away( Prog, Def, Msg )  
    char *Prog;  
    char *Def;  
    char *Msg;
```

DESCRIPTION

Chain_Away performs an *execl* to transform the current process into the new process represented by *Prog*. The *Def* argument to *Chain_Away* is a pointer to a character string containing the name of an APPGEN PDEF. The Appgen PDEF file will be opened and read if the program named by *Prog* is an Appgen program. The *Msg* argument is a pointer to a character string containing a message that will be printed if *Chain_Away* fails.

Chain_Away performs the following steps:

- 1) close all files
- 2) for each path defined in the shell environmental variable, AGPATH, attempt an *execl*
- 3) if unsuccessful, for each path defined in the shell environment, PATH, attempt an *execl*
- 4) if still unsuccessful, print *Msg* concatenated with "Chain Away failed" and exit through *perorr*.

SEE ALSO

execl in UNIX/XENIX manual.

NAME

formdata - format a data field for display or printing

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>

char *formdata( format, data, width )
    char *data, *format;
    int   width;
```

DESCRIPTION

formdata formats *data* as directed by the other parameters, returning a pointer to a static buffer containing the external format string. Numeric formats are right justified in a field of *width* spaces. All other formats are left justified.

SEE ALSO

Appgen Reference Manual for applicable formats

NAME

getvar - look for the name of an APPGEN environmental variable

SYNOPSIS

```
#include <appgen.h>
#include <ctype.h>

char *getvar( vname, dflt )
           char *vname;
           char *dflt;
```

DESCRIPTION

getvar looks in the UNIX shell environment for a variable named AGINIT. If found, it is searched for a variable named *vname*. If *vname* is found, then the return value of *getvar* is the value assigned to *vname*. Otherwise, it looks for *vname* in the UNIX shell environment. If still not found, the *dflt* is returned.

NAME

`mmove` - move memory to memory

SYNOPSIS

```
#include <appgen.h>  
#include <ctype.h>
```

```
mmove( f, t, n )  
char *f, *t;  
int n;
```

DESCRIPTION

mmove moves *n* number of bytes from *f* to *t*. *mmove* will make a faithful copy even if *f* and *t* overlap.

NAME

`pformat` - perform printf type formatting

SYNOPSIS

```
char *pformat( format, a1, a2, a3, a4, a5, a6, a7, a8 )  
char *format;  
int a1, a2, a3, a4, a5, a6, a7, a8;
```

DESCRIPTION

pformat takes a *printf* format string and an arbitrary number of arguments and returns a pointer to the resulting formatted string.

SEE ALSO

`printf(3)` in the UNIX/XENIX manual.

NAME

trap - ignore certain signals

SYNOPSIS

```
#include <signal.h>
```

```
trap()
```

DESCRIPTION

trap causes the following signals to be 'trapped' and ignored: SIGINT, SIGQUIT, and SIGTERM.

CHAPTER 8: IMPORTING FOREIGN DATA STRUCTURES

This chapter is provided as a guideline for moving data from another storage format to the APPGEN format. It supplies the necessary information to construct and successfully compile a 'C' language program that creates and updates an APPGEN data base file.

Two examples of importing are provided as a guideline only and are not designed to serve for actual accessing of foreign data structures. The examples provided in this chapter are both named import.c. The examples are referred to as Example I and Example II. Example I creates an APPGEN file from a foreign file consisting of ASCII textual data. The ASCII data described in Example I is listed in Appendix C. Example II creates an APPGEN file from a foreign file that contains data in a structured non-ASCII format.

In order for you to import data from a foreign file structure to the APPGEN file structure, you should use the techniques described here to write and execute a 'C' language program which follows the general outline of the sample programs listed. The techniques described in chapters one through six must be used to compile the program.

The four major sections of this chapter are:

- | | |
|-------------|--|
| Section 8.1 | Section 8.1 describes the uses of programs that import data. The basic steps executed in the example programs and the sample data used in the example programs are also described. |
| Section 8.2 | Section 8.2 describes the principal declarations of data constants and variables for Examples I and II. |
| Section 8.3 | Section 8.3 explains the execution of the importing program. |
| Section 8.4 | Section 8.4 describes the methods for reviewing program results. |

PREREQUISITE KNOWLEDGE

In order to create and compile an import program as explained in this chapter, the reader should understand:

- The 'C' programming language.
- The procedures described in chapter six of this manual for 'making' the program.
- The APPGEN data base structure and library routines.
- The structure of the data being moved.

8.1 OVERVIEW OF IMPORTING

PURPOSE OF IMPORTING

The program listings in Appendices D and E are examples of 'C' programs that transfer data from foreign file formats to the APPGEN file format. A foreign file is a file having a format that differs from the APPGEN format. Two applications of importing programs are:

1. In changing from another system to an APPGEN application such as Accounts Receivable, the data from existing computer files can be converted into the APPGEN format.
2. Data from foreign files may be used to regularly update APPGEN files.

EXAMPLE 1 DATA STRUCTURES

The foreign file in Example I contains Accounts Receivable customer transactions in an ASCII textual format. The Example I data is transferred to an APPGEN file as shown by the arrows below:

FOREIGN FILE	—>	APPGEN DATA BASE CUSTOMER FILE
Field 1 Customer Number	—>	Attribute 0 Part 1 APPGEN File Key
Field 2 Customer Name	—>	Attribute 1 Value 0 Customer Name
Field 3 Document Number	—>	Attribute 2 Multi-Value Document Number
Field 4 Sale Amount	—>	Attribute 3 Multi-Value Sale Amount
Field 5 Transaction Date	—>	Attribute 0 Part 2 APPGEN File Key

Example I reads multiple records from the foreign file and condenses these records into multi-values in the APPGEN file. Example I also illustrates the procedures for handling the following types of data fields in an APPGEN file:

1. Numeric fields. Field4 in Example I contains a decimal point that is removed before being copied to the APPGEN file. The decimal point is not stored in APPGEN files because the APPGEN formatter routines are able to convert fields from implied decimal formats to decimal formats upon output.
2. Date fields. Field5 is a transaction date in the format (mm/dd/yy) that must be converted to the APPGEN internal date format. The internal date format consists of a number that is converted by the APPGEN Julian Functions upon output. The number itself is calculated as the number of days since December 31, 1967 with 12/31/67 being day one. For example, 6508 is the internal date for 10/25/85.

3. Multi-part key fields. The key to the APPGEN file in Example I is multi-part consisting of the customer number and the internal date separated by a '*' character delimiter. For instance, the first foreign file record in Appendix C would have an APPGEN record key of 3145*6508 because the customer number is 3145 and the internal date of 10/25/85 is 6508. The multi-part key separator character in all applications created by APPGEN is the '*' character.
4. Multi-valued fields. Field3 and Field4 are single-value fields in the foreign file records that are copied to the APPGEN file as multi-valued fields. Multi-values are created if there are multiple input records with the same customer number and transaction date combination.

The fields in the foreign file listed in Appendix C are terminated by a pipe symbol character '|'. Each data record is terminated by a newline character. To create a foreign file as used in Example I, the fields can be of any length but must be terminated by the '|' separator character. Since APPGEN records, attributes, and multi-valued fields can have variable lengths, the length of the fields need not be adjusted before being copied to the APPGEN file.

EXAMPLE II DATA STRUCTURE

The foreign file in Example II again contains Accounts Receivable customer data, but the fields do not all contain ASCII textual data. The data records in Example II are a series of binary coded bytes that do not all translate to human readable characters. The foreign file format in Example II would probably have been created by a program.

Example II illustrates the procedures for handling the following types of data fields in an APPGEN file.

1. Non-ASCII format. The data type of the zip code field in Example II is a short numeric format.
2. Structured definition. Example II illustrates the use of a structure definition for the foreign file. The structure definition permits the foreign fields to be treated as a unit instead of as separate fields.
3. Conversion of data. In Example II the state code is a number that is converted to a state abbreviation.

The Example II data is transferred to an APPGEN file as shown by the arrows below:

FOREIGN FILE	—>	APPGEN DATA BASE CUSTOMER FILE
Customer Number (10 bytes)	—>	Attribute 0 Customer File Key
Customer Name (60 bytes)	—>	Attribute 2 Customer Name
Customer Address (30 bytes)	—>	Attribute 5 Customer Address
Zip Code (2 bytes)	—>	Attribute 3 Zip Code
State Code (1 byte)	—>	Attribute 1 State Code
Alignment (1 byte)	—>	not copied

The one byte alignment variable is used to change the size of the structure since most computers require that the structure be a multiple of the word size. The alignment variable is not copied to the APPGEN file.

OUTLINE OF PROGRAM

The execution of the importing program takes two arguments, the name of the foreign file and the name of the APPGEN file. An example of the execution of the importing program:

```
import foreign agfile
```

where import is the name of the importing program, foreign is the name of the foreign file, and agfile is the name of the APPGEN file. The procedures performed by both Example I and Example II can be simplified into the following steps:

1. Define the variables and constants.
2. Open file one, the foreign file.
3. Create the APPGEN file.
4. Read a foreign file record group.
5. Create an APPGEN data record using the first field of the record group as the key.
6. Copy fields two through five of the record group into the APPGEN record.
7. Execute a write to the APPGEN file.
8. Repeat steps four through seven until all record groups have been read.
9. Close the files.
10. End.

8.2 PROGRAM ORGANIZATION

The topics below describe the APPGEN include files and other declarations that are used throughout the example programs. The program listings in Appendices D and E contain additional comments and descriptions of the example programs.

INCLUDE FILES

The header files shown below must be included in the importing program in the order listed. The header files declare variables that are used by the APPGEN library routines. The header files are located in the appgen/include directory; their purposes are described below.

- appgen.h Contains declarations used by all APPGEN programs. The UNIX system include file, `stdio.h`, is included in `appgen.h` and should not be listed again in the include section of the program.
- vm.h Contains declarations used by the APPGEN Virtual Machine software to standardize certain system calls among various computers.
- db.h Contains declarations used by APPGEN data base routines.

DECLARATIONS - EXAMPLE 1

The variables and functions discussed below are used throughout Example 1 and must be initialized before being used. The variable declarations will depend on the format of the foreign file and the particular file handling routines being used. In Example 1 the foreign file is formatted as ASCII text, the foreign file is declared as a type `FILE`, and the file handling routines are `fopen()` and `fgets()` from the standard library, `libc`. Additional information on the `fopen()` and `fgets()` functions can be found in the UNIX Reference Manual.

char **argvec;

A global variable required by the APPGEN library software. The declaration of this variable occurs outside of any function. Inside `main()` the global variable `argvec` is set by the statement `argvec = argv` to point to the import argument vector.

char line[256];

The `line` variable is an array of 256 characters into which lines of foreign file data are read and parsed into data fields. This array should be at least as large as the largest foreign file record.

extern char *fgets();

This statement declares a standard library function that returns a pointer to a character string. The fgets() function call in Example 1 is:

```
fgets (line, sizeof(line), in_file);
```

The fgets() function fetches the next line of input from in_file, up to and including newline into line, and adds a terminating null character '\0'. At most, sizeof(line)-1 characters are copied. The function fgets() returns a value of 256. fgets() reads only 256 characters and terminates the string with a null character '\0'.

The fgets() function returns NULL at the end of the file. It provides no warning if the input line is too long for the size of the line buffer. The maximum number of characters to be read by fgets() should not be more than the size of the line buffer. Also the line buffer should be long enough for the largest record in the foreign file. In Example 1 the largest foreign record is not more than 255 characters including a newline character at the end of the record.

extern FILE *fopen();

The standard library function fopen() returns a pointer to a FILE type variable. The fopen() function call in Example 1 is:

```
in_file = fopen ( argv[1], "r");
```

The fopen() function returns an internal name to be used in subsequent operations on the file. The internal name is actually a pointer to a structure that contains information about the file such as the location of a buffer, the current character position in the buffer, and whether the file is being read or written.

The name of the file opened is in argv[], the first argument given on the command line. The FILE pointer returned by fopen() is assigned to the variable in_file. The 'r' option used in the above statement specifies that the file is to be opened for reading only.

char *word, *nxt_word;

The variables word and nxt_word are declared as pointers to character strings. The pointers are used within the two functions first() and next() that are described below. The nxt_word pointer is incremented by next() until the end of a field is found. The word variable marks the beginning of a field of data and a null character marks the end of a field of data.

int first(), next();

The two functions `first()` and `next()` are used to set the two pointers `word` and `nxt_word` to point to the beginning of a field of data. The function `first()` is called after each read by `fgets()` and sets the pointer `nxt_word` to the start of the line buffer. When `next()` is called, `nxt_word` is an index to the first character of either the line buffer or a data field. The `nxt_word` index is saved in `word` and `nxt_word` is incremented until one of the three characters below is found:

1. A null character.
2. A Pipe symbol, '|'.
3. A newline character.

In Example I the sample data file contains pipe symbol characters that mark the ends of data fields and newline characters that mark the ends of data records. If one of the foreign file data fields is null, the delimiting character '|' should nevertheless be included in the foreign file to mark the end of the null field. A sample of the ASCII data file described in Example I is listed in Appendix C.

When `nxt_word` points to one of the three characters above, the character is replaced by a null character, thereby terminating the field of data. The final step performed by `next()` is to increment `nxt_word` to the beginning of the next field of data.

extern char *date_in();

`date_in()` is an APPGEN library function that returns a pointer to character string. The argument to the `date_in()` function is a pointer to a null terminated string of characters having the format of `mmddy`. The character string created by `date_in()` is a series of numbers representing an internal date in the APPGEN file.

char *field1, *field2, *field3, *field4, *field5;

These pointers are used to mark the beginning of data fields in the foreign file record. It is necessary in our example to combine the data from `field1` and `field5` and put these in the key variable to create the record key for the APPGEN file. After the APPGEN record is created, `field2`, `field3`, and `field4` from the foreign file record are copied to the new record. `field3` and `field4` are single-value fields in the foreign file that are copied to the APPGEN record as multivalued by the statements shown below.

```
replace (out_db, 2, -1, field3);  
replace (out_db, 3, -1, field4);
```

The APPGEN records in Example I are read more than once because some of the foreign file records have the same key field values, i.e. customer number and transaction date. The multivalued records are created when an existing record is read more than once and the above statements are executed for the record. In the first statement above the third data field from the foreign file record is added to the end of the list of multivalued records in attribute two of the APPGEN record. The use of -1 as the attribute value causes the foreign file data field to be placed at the end of the multivalued list. Similarly, in the second statement above, the fourth data field from the foreign file record is added at the end of the list of multivalued records in attribute three of the APPGEN record. Additional information on the `replace()` function is available in Chapter 7 of this manual.

char key[100];

Although not all APPGEN file keys are multipart, the record key in Example I consists of two parts, the customer number and the transaction date. The key variable is an array of 100 characters in which the two parts of the APPGEN key record are combined. The key array should be at least as long as the longest record key. The statement to combine the two fields is:

```
sprintf(key, "%s*%s", field1, date_in(field5));
```

`field1` and `field5` are defined in Section 8.2. The resulting character string in the key variable consists of the customer number followed by a '*' character, followed by the internally formatted transaction date. The '*' character is a separator character that is used in multipart keys within all applications. In the next statement the key variable is used to create or read an APPGEN record.

```
if (db_newrec (out_db, key, 0L ) < 0 )
```

The `db_newrec()` function returns -1 if an error condition occurs and 0 if the record was created without problems. If the record already exists `db_newrec()` attempts to read the record with a lock and returns 1 if the record was read. In Example I, an error message is printed only if `db_newrec()` returns a -1. The sample data in Appendix C contains several foreign file records having the same APPGEN file key (customer number and transaction date in this case). If the APPGEN record is read more than once, the single-valued fields in the foreign file records are added to the APPGEN fields as multivalued.

FILE *in_file;

The variable `in_file` is a pointer to a standard library FILE structure. `in_file` is a FILE pointer declaration that is required by the `fopen()` and `fgets()` standard library routines.

DB *out_db;

The variable `out_db` is declared a pointer to an APPGEN library DB structure. `out_db` is a DB pointer declaration that is required by the APPGEN library routines. In the statement below the APPGEN library routine `db_create()` returns a DB pointer that is assigned to the variable `out_db`.

```
out_db = db_create ( argv[2], 357L );
```

int rec_cnt=0;

The `rec_cnt` variable is incremented each time an APPGEN record is created. The value in this counter is printed at the end of the program to show the number of records copied.

DECLARATIONS - EXAMPLE II

The variables and functions discussed below are based on a foreign file having a structure of data fields that are not all character types. In Example II the foreign file is formatted as a sequence of machine readable structures and the file handling routines are the standard system calls `open()` and `read()`.

char **argvec;

The `argvec` vector should be declared as described above for Example I.

struct foreign;

This is a collection of one or more variables grouped together for convenient handling. In Example II the fields related to a single customer record are grouped together. The statement:

```
struct foreign fbuf
```

defines and allocates storage for a collection of variables named by `fbuf`. By placing the declaration of `foreign` and `fbuf` before `main()` they become available to all functions that reside in the source file.

Not all foreign files will be able to use a structure to define the fields of data. For example, the foreign file fields and records may be variable lengths and separated by delimiter characters. Instead of reading a structure at a time the program may read only a field at a time up to a delimiter character. If a structure is used, the structure variables would be defined to conform to the fields in an existing file rather than building a file according to an arbitrary structure as in Example II.

int in_file;

The variable `in_file` is a file descriptor that identifies the foreign file. This file descriptor is initialized by the statement:

```
if ((in_file = open (argv[1], 0)) < 0)
```

The name of the foreign file is the first argument given on the command line when the program is executed. If the foreign file cannot be opened, an error message is printed and processing is ended.

The `open()` function is like the `fopen()` function in Example I except that instead of returning a file pointer it returns a file descriptor which is an `int`. As with `fopen()` in Example I, the file name argument is a character string containing the foreign file name and the access mode is read only.

DB *out_db;

The out_db declaration and the creation of the APPGEN file in Example II should be performed as described for Example I above.

char xbuf[80];

xbuf is a general purpose character array of 80 characters into which the foreign fields are read. In order for the field to be copied correctly to the APPGEN file each field must be terminated with a null character '\0'. The null character is added to the variables in the xbuf buffer before they are copied to APPGEN file. The array must be long enough to contain the largest data string copied into it.

#define

The define statements of Example II allow substitution of a symbol such as NAMATTR, ADDATTR, ZIPATTR, STAATTR, or FBUFSIZ for an actual value. Through the use of the #define statement data fields of the APPGEN file are assigned meaningful symbolic names that can be used instead of attribute numbers when copying data to the APPGEN file.

LIBRARIES

The APPGEN functions that are required to create an APPGEN file are described in Section 7.2 of this manual. These APPGEN functions provide the necessary tools for:

- Creation, deletion, opening, closing, and locking of files
- Creation, deletion, reading, writing, and locking of records
- Extraction, deletion, insertion, and replacement of attributes and their multi-values

8.3 EXECUTING THE PROGRAM

Before executing import the appgen/bin directory must be specified in your UNIX PATH variable definition. If the C shell is being used and the importing program did not exist prior to logging in, import must be added to the list of available commands by typing at the UNIX shell prompt:

```
rehash
```

To invoke the import program type:

```
import foreign AGFILE
```

where foreign is the foreign file and AGFILE is the name of the APPGEN file to be created.

8.4 REVIEWING PROGRAM RESULTS

The APPGEN data file created by the import program can be viewed through the use of the APPGEN Development System or the APPGEN utilities dbed and dbdump. The procedures for using these methods are described in detail in the APPGEN Development System Manual.

The APPGEN Query Language (AQL) Reference Manual also explains procedures that may be used to display the APPGEN data file. AQL requires one of the following to be used as a data dictionary:

1. A data dictionary named after the data file in the current directory; e.g., D.AR-CUSMAS.1 for AR-CUSMAS.1.
2. A Master Dictionary, D.MASTER, in the current directory.

An example of an AQL statement that uses the Master Dictionary is as follows:

```
SHOW AR-CUSMAS.1 Key A1 A2 A3 A4 A5
```

The statement above lists attributes 0, 1, 2, 3, 4, and 5 of the file named AR-CUSMAS.1. To obtain a listing of the items in the Master Directory located in the current directory type the command below at the UNIX shell prompt.

```
SHOW D.MASTER BY Key Key Desc
```

For further information on the use of AQL, consult the AQL Reference Manual.

APPENDIX A: SUB-ROUTINE LISTING

```

/*
 * U("arma3",E,E(1))
 *   argv[1] is the entry
 *   argv[2] is the customer number
 *
 * A(2,27) controls open item list
 *   E(1):"":A(2,27,V) read file 3
 *   display as in basic version.
 *
 * rows 6-15
 * col 35 - doc number A(2,27,V)      A10
 * col 46 - doc type A(3,4,1) A 1
 * col 48 - date A(3,3,1)      D
 * col 57 - sum file 3, attr 5 - 8 + 12 - 15, val 1 ( original amt )N7.211
 * col 69 - A(3,1,1) balance type D > 0; type C < 0;      N7.211
 */

# include <appgen.h>
# include <vm.h>
# include <db.h>
# include <hp.h>
# include <ctype.h>

extern DB *db_[];
extern char *schr();
extern char *scopy();
extern char *scmp();

/* ARGSUSED */
arma3(argc, argv, dest)
    int argc;
    char *argv[];
    char dest[];
{
    int vnum;
    int lnum;
    hp sumbuf;
    char temp[80];
    hp hptemp;
    char type;

    if ((argv[1][1] == '\0') && argv[1][0] && schr("CcDd", argv[1][0])) {
        for (lnum = 6; lnum <= 15; lnum++) {
            clrflid(lnum, 35, 45);
        }
        lnum = 6;
        for (vnum = 1; extract(db_[1], 27, vnum, dest, 79) > 0; vnum++) {
            if (lnum > 15) {
                putfld(MESGLINE, 0, 80,
                    "there is more - return to continue, E to end ");
                getfld(-1, -1, 3, temp);
                clrflid(MESGLINE, 0, 80);
                if (tolower(temp[0]) == 'e') {
                    break;
                }
            }
            for (lnum = 6; lnum <= 15; lnum++) {
                clrflid(lnum, 35, 45);
            }
            lnum = 6;
        }
        sprintf(temp, "%s%s", argv[2], dest);
        if (db_read(db_[2], temp, 0) != 0) {
            putfld(MESGLINE, 0, 80, temp);
            putfld(-1, -1, 80, ": document not on AR-OPEN");
            getfld(-1, -1, 0, "");
            continue;
        }
    }
}

```

```

}
extract(db_2], 1, 1, temp, 10);
if (tolower(argv[1][0]) == 'd' ? atol(temp) > 0 : atol(temp) < 0) {
    putfld(lnum, 35, 10, dest);
    dispdata(lnum, 69, "N7.211", temp, 11);
    extract(db_2], 4, 1, temp, 3);
    type = temp[0];
    putfld(lnum, 46, 1, temp);
    extract(db_2], 3, 1, temp, 10);
    dispdata(lnum, 48, "D", temp, 8);
    l_to_hp(0L, sumbuf);
    if (type == 'P') {
        extract(db_2], 5, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
    } else {
        extract(db_2], 5, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 6, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 7, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 8, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 12, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 13, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 14, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
        extract(db_2], 15, 1, temp, 10);
        hp_add(a_to_hp(temp, hptemp), sumbuf, sumbuf);
    }
    hp_to_a(sumbuf, temp);
    dispdata(lnum, 57, "N7.211", temp, 11);
    lnum++;
}
}
scopy(dest, "N");
} else {
    if (!argv[1][0]) { /* null response */
        for (lnum = 6; lnum <= 15; lnum++) {
            clrflld(lnum, 35, 45);
        }
        dest[0] = '\0';
        return (0);
    }
    sprintf(dest, "%s*%s", argv[2], argv[1]);
    if (db_read(db_2], dest, 0) == 0) {
        extract(db_2], 1, 1, temp, 10);
        if (atol(temp)) {
            dest[0] = '\0';
            return (0);
        } else {
            putfld(MESGLINE, 0, 80,
                "Item has a zero balance - return to continue ");
        }
    } else {
        putfld(MESGLINE, 0, 80, "Item not on file - return to continue ");
    }
    getfld(-1, -1, 0, "");
    clrflld(MESGLINE, 0, 80);
    scopy(dest, "N");
}
return (0);
}

```

APPENDIX B: STAND ALONE PROGRAM LISTING

```

/* file_copy.c */

/*
 * This is general purpose file copy routine. It expects the argument vector
 * to contain the standard information AND the menu and select files must
 * contain the appropriate information ( ss.tty? se.tty? ). It traps such
 * errors as having the source file name defined the same as the destination
 * file. If file names are different but both are defined at the same inode
 * ( linked ), processing will continue as if the record to copy already exists
 * in the destination file. When a destination file currently contains the
 * current record to be copied, a warning message is displayed along with a
 * prompt ( End ). If the users selects to End then processing is terminated
 * otherwise it goes about its business. The record is not overlaid.
 * Selected but missing records along with bad writes, invalid file
 * parameters, and missing files, all will display the appropriate error
 * message and force user response.
 */

# include <appgen.h>
# include <vm.h>
# include <db.h>

# define STRSIZE 30
# define BUFSIZE 80
# define DEBUG

DB *pdef; /* pdef file */
DB *fr_file; /* file copying from */
DB *to_file; /* file copying to */
FILE *s_se; /* select file */
FILE *m_se; /* menu selections */
char buffer[BUFSIZE];
char endprog[STRSIZE];
char endpdef[STRSIZE];
char **argvec; /* naming conventions */
extern int errno;

int slen(), atoi(), scmp();

char *scopy(), *scat();

main(argc, argv)
int argc;
char *argv[];
{
char p_name[STRSIZE]; /* pdef file name */
char s_name[STRSIZE]; /* select file name */
char m_name[STRSIZE]; /* menu selection file */
char fr_name[STRSIZE]; /* from file name */
char to_name[STRSIZE]; /* to file name */
char fr_key[STRSIZE]; /* from key */

int atr, max_att, val;

argvec = argv;
errno = 0;
if (argc <= 6) {
printf(buffer, "Invalid number of arguments passed to [%s]", PROGRAMNAME);
error();
}
/* open pdef file */

```

```

sprintf(p_name, "PDEF.%s", PDEFNAME);
_db_size_check = FALSE;
if ((pdef = db_open(p_name)) == (DB *) NULL) {
    _db_size_check = TRUE;
    sprintf(buffer, "Open failed on [%s] in [%s]", p_name, PROGRAM);
    error();
}
_db_size_check = TRUE;

/* get ending stuff */

if (db_read(pdef, "PROG.NAME", FALSE) != 0) {
    sprintf(buffer, "PROG.NAME not constructed for [%s]", p_name);
    error();
}
extract(pdef, 8, 1, endprog, STRSIZE);
extract(pdef, 9, 1, endpdef, STRSIZE);
extract(pdef, 14, 1, buffer, STRSIZE);
if (atoi(buffer) <= 0) {
    max_att = 13;
} else {
    max_att = atoi(buffer);
}
extract(pdef, 10, 1, buffer, STRSIZE);
if (buffer[0] != '\0')
    scpy(endprog, buffer);
extract(pdef, 13, 1, buffer, STRSIZE);
if (buffer[0] != '\0')
    scpy(endpdef, buffer);

/* get from file for processing */

extract(pdef, 2, 1, fr_name, STRSIZE);      /* from file      */

/* get destination company from menu prompt */
sprintf(m_name, "se.%s", TTYNAME); /* menu selc */
if ((m_se = fopen(m_name, "r")) == (FILE *) NULL) {
    sprintf(buffer, "Open failed on menu selection [%s]", m_name);
    error();
}
if (fgets(buffer, STRSIZE, m_se) == NULL) {
    sprintf(buffer, "No destination company defined for [%s]", PROGRAM);
    error();
}
buffer[slen(buffer) - 1] = '\0'; /* replace nl      */

/* form destination file name */
scopy(to_name, fr_name);
if (atoi(COMPANYNUM) != 0) {
    scat(fr_name, ".");
    scat(fr_name, COMPANYNUM);
}
if ((buffer[0] != '0') && (buffer[0] != '\0')) {
    scat(to_name, ".");
    scat(to_name, buffer);
}
if (scmp(fr_name, to_name) == 0) {
    scpy(buffer, "You CANNOT copy to the SAME file");
    warn();
    quit();
}
/* now open processing files */

if ((fr_file = db_open(fr_name)) == (DB *) NULL) {
    sprintf(buffer, "Opened failed on source file [%s]", fr_name);
    error();
}
if ((to_file = db_open(to_name)) == (DB *) NULL) {
    sprintf(buffer, "Opened failed on destination file [%s]", to_name);
    error();
}

```

```

}
sprintf(s_name, "ss.%s", TTYNAME); /* menu prompts */
if ((s_se = fopen(s_name, "r")) == (FILE *) NULL)
    quit();

/* all necessary files are now open - start processing */
putfld(MESGLINE, 17, 12, "Now copying ");
S_flush();
while (fgets(fr_key, BUFSIZE, s_se) != NULL) {
    fr_key[slen(fr_key) - 1] = '\0';
    if (db_read(fr_file, fr_key, FALSE) == 0) {
        if (db_newrec(to_file, fr_key, (long) BUFSIZE)) {
            sprintf(buffer,
                "**** WARNING *** - File [%s] currently contains record [%s]",
                to_name, fr_key);
            warn();
            if ((scmp(buffer, "E") == 0) || (scmp(buffer, "e") == 0))
                quit();
        } else {
            putfld(MESGLINE, 30, STRSIZE, fr_key);
            S_flush();

            for (atr = 1; atr <= max_att; atr++) {
                for (val = 1;
                    extract(fr_file, atr, val, buffer, BUFSIZE) > 0;
                    val++) {
                    replace(to_file, atr, val, buffer);
                }
            }
            if (db_write(to_file) == -1) {
                sprintf(buffer, "Write failed on file [%s], key is [%s]",
                    to_name, fr_key);
                error();
            }
        }
    } else {
        sprintf(buffer, "Selected record missing [%s]", fr_key);
        error();
    }
}
quit();
}

static
quit()
{
    closes();
    S_exit();
    Chain_Away(endprog, endpdef);
    scpy(endprog, "Menu");
    scpy(endpdef, SYSINITIALS);
    scat(endpdef, "000000");
    sprintf(buffer, "Chain Away failed on [%s]", PROGNAME);
    error();
}

error()
{
    static char scratch[BUFSIZE];

    sprintf(scratch, "%s ERRNO is [%3d]", buffer, errno);
    scpy(buffer, scratch);
    warn();
    S_exit();
    closes();
    Chain_Away(endprog, endpdef);
    exit(-1);
}

closes()

```

```
{
  if (pdef != (DB *) NULL)
    db_close(pdef);

  if (fr_file != (DB *) NULL)
    db_close(fr_file);

  if (to_file != (DB *) NULL)
    db_close(to_file);

  if (s_se != (FILE *) NULL)
    fclose(s_se);

  if (m_se != (FILE *) NULL)
    fclose(m_se);
}

warn()
{
  scat(buffer, " ");
  putfld(MESGLINE - 1, 0, 79, buffer);
  S_flush();
  scpy(buffer, "E");
  getfld(-1, -1, 1, buffer);
  if (buffer[0] == '\0')
    scpy(buffer, "E");
}
```

APPENDIX C: EXAMPLE 1 DATA

3145 AMERICAN DIODE|112|2700.00|10/25/85
3120 AMERICAN GRAPHICS|200|4820.00|01/15/85
3170 AMERICAN GROUP|93|21500.00|12/07/84
3010 BEST CHEMICALS|6014|7280.00|06/13/85
3055 BESTWAYS COMPANY|897|12400.00|09/23/85
3085 BINDER PRINTING|750|6200.00|10/06/85
3175 COASTAL FLYER|903|77250.00|03/20/85
3145 AMERICAN DIODE|113|5608.10|10/25/85
3010 BEST CHEMICALS|6015|3280.00|06/13/85
3170 AMERICAN GROUP|94|1400.00|12/07/84
3120 AMERICAN GRAPHICS|201|330.00|01/15/85

APPENDIX D: IMPORT.C LISTING - EXAMPLE 1

```

/*
 * A sample import program for the APPGEN DB file system.
 *
 * processes records from an ASCII text file, one record per line with the
 * or-bar (|) separating the fields, creating an APPGEN database file. The
 * data fields are:
 *     1) a numeric customer number is the first part of a
 *        multi-part key,
 *     2) text (name) attribute one,
 *     3) text (address) attribute two,
 *     4) number w/decimal attribute three,
 *        (convert to implied decimal form)
 *     5) date w/slashes (D2/) is second part of multi-part key,
 *        (convert to APPGEN date form )
 *
 * This program is intended to be a 'go-by', it should be modified to reflect
 * the actual data in your situation.
 *
 * To compile the code, enter it in import.c in the 'appgen/src' directory and
 * make appropriate entries in the makefile.
 *
 * Note that the program as presented is not very tolerant of input errors.
 */

# include <appgen.h>
# include <vm.h>
# include <db.h>

char **argvec; /* global copy of argument vector required by
 * some library functions */

/*
 * definitions for text record handling
 */
char line[256]; /* big enough for largest record */
extern char *fgets(); /* type must be declared for externals */
extern FILE *fopen();
char *word, *nxt_word; /* point into line */
int first(), next(); /* decompose line (defined below) */
extern char *date_in(); /* from APPGEN library */
char key[100];
char *field1, *field2, *field3, *field4, *field5; /* field pointers */

main(argc, argv) /* standard C program start-off */
int argc;
char *argv[];
{
    FILE *in_file; /* stdio file descriptor for text file */
    DB *out_db; /* APPGEN database file */
    int rec_cnt = 0; /* count the records */
    int i; /* general index */

    argvec = argv; /* initialize global argvec */

    if (argc != 3) { /* need exactly 2 arguments */
        /* error message */
        printf("usage: %s infile outfile\n", argv[0]);
        exit(1); /* non-zero convention for error exit */
    }

    /*
     * Try to open input file first to simplify the error handling if it

```

```
* cannot be opened (it's easier to back out of than a dbopen).
*/

if (!(in_file = fopen(argv[1], "r"))) {
    perror(argv[1]);      /* print the usual cryptic message */
    exit(1);
}

/*
 * Now either create ( or open ) the output file.
 *
 * The second parameter to dbcreate is a long, it is the hashtable size to
 * be used in the new file.
 *
 * Choose the hashtable size to be a prime number approximately equal to
 * the number of records that will eventually inhabit the file
 */

if (!(out_db = db_create(argv[2], 357L))) {
    /* if the pointer is null ... couldn't create data base - error */
    /* if the file exists db_create will attempt to open and lock it */
    perror(argv[2]);
    fclose(in_file);
    exit(1);
}

/*
 * Now read all the data in the input file constructing and writing a
 * record for each.
 */

while (fgets(line, sizeof(line), in_file)) {      /* read a line ... */
    first();      /* isolate first word */
    field1 = word;
    next();      /* isolate next word */
    field2 = word;
    next();
    field3 = word;
    next();
    field4 = word;
    next();
    field5 = word;

    /* record key is multi-part: customer number * dat4e */

    /*
     * field five must be converted to an Internal date. two steps are
     * required: eliminate punctuation by moving each of the month and
     * year digits left one char; and perform conversion. The string
     * starts in the format mm/dd/yy
     */
    field5[2] = field5[3]; /* mddd/yy */
    field5[3] = field5[4];
    field5[4] = field5[6]; /* mddyyy */
    field5[5] = field5[7];
    field5[6] = '\0'; /* mddy */

    /* date_in converts from format mddy to internal julian date */

    sprintf(key, "%s*s%s", field1, date_in(field5));
    /* create a record and handle duplicate case */
    /* if record exists db_newrec attempts to read and lock it */
    if (db_newrec(out_db, key, 0L) < 0) {
        perror(word); /* probably locked record */
        continue; /* read next line */
    }
    /* increment record count */
    rec_cnt++;
}
```

```
/* attributes one is simple text */
replace(out_db, 1, 0, field2);

/*
 * attribute three must be converted to implied decimal form. we are
 * assuming that there will be exactly two decimal places.
 */

i = strlen(field4);
/* move decimal digits left one char overwriting decimal point */
field4[i - 3] = field4[i - 2];
field4[i - 2] = field4[i - 1];
field4[i - 1] = '\0';
/* attributes two and three are multi-valued fields */
replace(out_db, 2, -1, field3);
replace(out_db, 3, -1, field4);

/* db_write makes the replaces permanent */
db_write(out_db);

/* continue while loop until in_file exhausted */
}

/* Close the files */
fclose(in_file);
db_close(out_db);

printf("done!! %d records imported\n", rec_cnt);
exit(0);
}

/*
 * first() and next() — string breakdown functions
 */
first()
{
    nxt_word = line;
    next();
}

next()
{
    word = nxt_word;
    for (nxt_word = word; *nxt_word && *nxt_word != '|' &&
        *nxt_word != '\n'; nxt_word++);
    *nxt_word++ = '\0';
}
}
```


APPENDIX E: IMPORT.C LISTING - EXAMPLE 2

```

/*
 * A sample import program for the APPGEN DB file system.
 *
 * reads a fixed format structure described by 'struct foreign' sequentially
 * from argv[1] (the first command-line argument) and adds them to the DB
 * file named by the argv[2].
 *
 * This is intended to be a 'go-by', it should be modified to reflect both the
 * foreign and native structure that you actually have.
 */

# include <appgen.h>
# include <vm.h>
# include <db.h>

char **argvec;          /* global copy of argument vector */

/*
 * define and allocate storage for the foreign structure
 *
 * most machines will insist, perhaps silently, that the structure have a
 * size that is a multiple of the word size.  this may take some juggling, or
 * the FBUFSIZ macro may be CAREFULLY redefined.
 */

struct foreign {
    char    f_key[10];          /* ten byte key string, null padded */
    char    f_fullnam[60];     /* full name, null padded */
    char    f_address[30];     /* street address */
    short   f_zip;             /* 16 bit zip code */
    char    f_scode;           /* 8 bit state code */
    char    f_align;           /* padding for alignment */
};

# define FBUFSIZ             (sizeof (struct foreign))

struct foreign fbuf;        /* should only need one of these */

/*
 * define the attribute numbers for the data strings
 */

# define  NAMATTR            2
# define  ADDATTR            5
# define  ZIPATTR            3
# define  STAATTR            1

/*
 * Map the state code to the standard abbreviation for its name
 */
char *sname[51] =
{
    "XX",          /* zero means unassigned */
    "TX",          /* 1 = texas */
    "CA",          /* 2 = calif. */
    /* ..... (I'm not really going to type this out */
    /* you get the idea */
    "AZ"          /* 50 = ariz. */
};

main(argc, argv)          /* standard C program start-off */
    int  argc;

```

```

char    *argv[];
{
int     in_file;          /* file descriptor for foreign file */
DB      *out_db;         /* pointer for database file */
int     rec_cnt = 0;     /* count the records, if you want */
char    xbuf[80];       /* a utility string buffer */

argvec = argv;          /* initialize global argvec */

if (argc != 3) {        /* need exactly 2 arguments */
    /* error message */
    printf("usage: %s infile outfile\n", argv[0]);
    exit(1);            /* non-zero convention for error exit */
}

/*
 * Try to open input file first to simplify the error condition. If it
 * cannot be opened (it's easier to back out of than a dbopen)
 */

if ((in_file = open(argv[1], 0)) < 0) {
    perror(argv[1]);    /* print the usual cryptic message */
    exit(1);
}

/*
 * Now either create ( or open ) the output file.
 *
 * The second parameter to dbcreate is a long, it is the hashtable size to
 * be used in the new file.
 *
 * Choose the hashtable size to be a prime number approximately equal to
 * the number of records that will eventually inhabit the file
 */

if (!(out_db = db_create(argv[2], 357L))) {
    /* if the pointer is null ... couldnt create data base - error */
    perror(argv[2]);
    close(in_file);
    exit(1);
}

/*
 * Now read all the data in the input file constructing and writing a
 * record for each.
 */

while (read(in_file, (char *) &fbuf, FBUFSIZ) == FBUFSIZ) {
    /* While a structure is successfully read ... */

    /* If a string may not have a null after it, take care of that */

    strncpy(xbuf, fbuf.f_key, 10);    /* max of 10 chars */
    xbuf[10] = '\0';                /* add a null, just in case */

    /* Create a new record for the item */
    /* (DB *) file, (char *) key, (long) initial length */
    /* Use the an initial record length of 0 (zero) unless */
    /* you know each record will be exactly the same size in which */
    /* case use that size as the initial length. */

    if (db_newrec(out_db, xbuf, (long) 0) != 0) {
        perror(xbuf);
        continue;            /* try the next one */
    }
    /* Increment the count of successful records */

    rec_cnt++;
}

```

```
/* Handle name and address like the key ... */
sncpy(xbuf, fbuf.f_fullnam, 60);
xbuf[60] = '\0';

/* except that it gets 'replaced' in the new record */
/* (DB *) file */

replace(out_db, NAMATTR, 0, xbuf);

sncpy(xbuf, fbuf.f_address, 30);
xbuf[30] = '\0';
replace(out_db, ADDATTR, 0, xbuf);

/* The zip code needs to be made a string */
sprintf(xbuf, "%05d", (int) fbuf.f_zip);
replace(out_db, ZIPATTR, 0, xbuf);

/*
 * Let's map the numeric state code to a two letter abbreviation,
 * using the sname array.
 */

if ((fbuf.f_scode >= 0) && (fbuf.f_scode < 50))
/* If it is in the array ... */
{
    replace(out_db, STAATTR, 0, sname[fbuf.f_scode]);
} else {
    /* Error code */
    replace(out_db, STAATTR, 0, "XX");
}

/*
 * The record must be written to make the foregoing replaces
 * effective.
 */

db_write(out_db);
}

/* Close the files */
close(in_file);
db_close(out_db);

/* Be cute */
printf("done!! %d records imported\n", rec_cnt);
exit(0);
}
```

