head 1.1; access; symbols; locks; strict; comment @# @; 1.1 date 99.12.17.01.39.13; author guy; state Exp; branches; next ; desc @@ 1.1 log @Start a new version of the Accu-Basic developer's manual - this time in HTML format. Change the CURSOR definition to be part of the DIM statement of the form: DIM cursor-name AS CURSOR file-name This makes it more BASIC-like based on my Visual Basic manual. Rationalize the use of the line continuation using the trailing backslash to apply to all lines. It was too hard to describe the exceptions in the manual. A few other compiler fixes like not allowing nesting of procedures. @ text @

# Accu-Basic Developer's Reference Manual

Revised for version 2.0 by Guy Lancaster on December 16, 1999




## Contents



## Overview

Accu-Basic is a programming language designed to generate reports on the Accu-Vote line of voting machines. It is loosely based on the BASIC programming language with forth generation language extensions to provide access to the election data. Accu-Basic programs are strictly read-only with respect to the election data and so cannot be used to manipulate the election data in any way.

The person developing the report (the developer) uses a text editor to edit a file containing the Accu-Basic source (.abs). When ready to test the report, the developer runs the Accu-Basic compiler to generate an object file (.abo). If the compiler reports errors, the developer will need to make corrections to the source and recompile until the compiler finds no errors. Should the compiler still report warnings, the resulting object file may or may not be valid and normally these should be dealt with before continuing.

The object file should be tested before being used in elections. This may be done by downloading it to an Accu-Vote for each of the election configurations that it is designed to handle. All options should be exercised for each election configuration before releasing the file for general use. In order to download a new object file using GEMS, one can either substitute it for a currently supported object file or add an entry in the ABasic.ini file in the main GEMS directory.

It is strongly recommended that the Accu-Vote reports be run as part of the election testing for each election. Should there be a problem, the program will need to be fixed and all affected memory cards will need to be reprogrammed with the new program.



# Compiling Your Accu-Basic Program

## Program Structure Overview

An Accu-Basic source file consists of a number of "lines" where each line may is either a statement or a statement continuation. Logically the program consists of a set of definitions where each definition defines a symbol as either a variable (DIM), a cursor (CURSOR), or a procedure (PROC). Procedure definitions include a set of command statements, its "statement list", that forms the body of the procedure.

Statements are usually indented using tab or space characters to indicate that the statement is part of the statement list for a procedure or command. It is recommended that the developer use either tabs or spaces for indentation and not mix the two since different editors and printers use different sizes for tab indents.

It is very useful both to the original developer and to later readers and maintainers of the program to include remarks in the program source. It is common practice to include a summary of the program's purpose and unique features at the top of the source file and then to include remarks that explain any statements or blocks of code that are not obvious to the reader. These are done using REM statements and the double slash markers. Blank lines are useful to separate sections of code so that they are easier to recognize and read.

The Accu-Basic compiler is case insensative meaning that it doesn't matter whether words are in upper, lower, or mixed case or whether or not they are consistently capitalized in different parts of the code. In our examples, reserved keywords will be in upper case and programmer defined words will be in lower or mixed case as a matter of convention.



## Statements

A statement may be blank or begin with either a command keyword or a variable name that is being assigned the value of an expression. Lines with nothing in them or with only space or tab characters are considered "blank lines" or "null statements" are are ignored by the compiler. A statement is terminated by either the end of a line or by a double slash "//" mark. The end of line may be "escaped" by finishing the line with the backslash character "\" so that the next line is treated as a continuation of the current one. This may be repeated as many times as necessary until reaching a double slash mark or an un-escaped end of line.

Example of continuation:

```
DIM %theTotal, \
    %theAverage, \
    %theLargest
```

The double slash mark "//" may be used to add a remark to the end of any statement including null statements. This is considered a statement terminator and all further text up to the end of the line is ignored by the compiler. This means that there is no way to insert a remark in the text of a statement and that there can not be more than one statement per line of code.

Example of the double slash statement termination mark:

```
DIM $myword   // This comment will be \
                 ignored by the compiler
```

### REM Statements

The REM statement begins with the REM keyword and is simply ignored by the compiler.

Example of a REM statement:

```
REM This is a comment \
    that will be ignored \
```

```
by the compiler.
```

## DIM Statements

The DIM statement begins with the DIM keyword and is followed by one or more variable definitions separated by commas. The DIM statement is a definition statement and thus can appear outside the body of a procedure. It's effect is to define the type of and reserve space for the variables being defined. It is most commonly used outside of the procedure definitions to define global variables that are shared between the procedures.

See the section "Variables and Expressions" for details on legal variable definitions.

Example:

```
DIM $myword, %theTotal, race1 AS CURSOR RACE
```

## Procedure Statements

```
Syntax: PROC procedure-name
          statement-list
        ENDPROC
```

Procedure statements begin with either PROC or ENDPROC and a procedure is defined as the statements between a PROC and and ENDPROC statement. The PROC statement must include a procedure name that is unique within the program file. This procedure name is used by the CALL statement to invoke the procedure. When the procedure completes, it returns and the program continues with the statement following the CALL statement. The RETURN statement can be used to return from the procedure before it reaches the end.

Example:

```
PROC myproc
  DISPLAY "HELLO WORLD"
ENDPROC

...
CALL myproc
```

Accu-Basic does not support nesting of procedures and does not currently support forward declarations

of procedures, parameters, or return values.

There are a set of required procedures and the compiler will warn you if they are not defined. These are the entry points that the Accu-Vote systems use to invoke your program. These include:

- ZERO_TOTAL_REPORT - to print the optional zero totals report on download,
- ELECTION_ZERO_REPORT - to print to official zero totals report prior to opening the polls,
- ELECTION_RESULTS_REPORT - to print the defaults official results report after close of polls,
- TEST_ZERO_REPORT - to print the optional zero totals report prior to counting in pre-election test mode,
- TEST_RESULTS_REPORT - to print the optional results report upon completion of a pre-election count test, and
- LABEL_REPORT - to print a memory card label.



## BREAK Statements

```
Syntax: BREAK
```

A Break statement is composed merely of the keyword BREAK and is used to break out of a loop. It can be used in either a FOREACH or a WHILE loop. It causes the program to continue as if the loop completed normally.

Example:

```
FOREACH PRECINCT pct1
  IF pct1.label = "JOKER" THEN
    BREAK
  ENDIF
  PRINT pct1.label
ENDFOREACH
```

## CALL Statements

```
Syntax: CALL procedure-name
```

The CALL statement is used to execute the specified procedure. When the procedure returns, the program continues with the statement following the CALL statement.

Example:

```
%reportTitle = "ZERO TOTALS REPORT"
CALL print_results
...

PROC print_results
  PRINT $reportTitle
  ...
ENDPROC
```

## CONTINUE Statements

Syntax: CONTINUE

The CONTINUE statement skips any remaining statements in the current loop and starts back at the top of the loop. It can be used in either a FOREACH or a WHILE loop. It's useful to handle error conditions or to selectively apply some contents of the loop.

```
FOREACH CANDIDATE cand1
  REM Skip the cross-endorsements for each candidate
  IF cand1.vgroup.index < 0 THEN
    CONTINUE
  ENDIF
ENDFOREACH
```



## DISPLAY Statements

## IF Statements

## FOREACH Statements

## PRINT Statements

## RETURN Statements

## WHILE Statements

# Variables and Expressions

@