Monarch Data Systems, Inc.

MAKEBOOT

Create Self-Booting Diskettes and Tapes
From ABC Compiled BASIC Programs

Reference Manual

Create Self-Booting Cassettes and Disks
From ABC Compiled BASIC Programs

Version 1.03

Monarch Data Systems, Inc.
P.O. Box 207
Cochituate, Massachusetts 01778
(617) 877-3457

CONTENTS

Section 1
Introduction

Makeboot is a utility that lets you create self-booting disk or cassette versions of your compiled software. It greatly reduces the loading time of your programs, and saves memory and disk space by eliminating the need for a disk operating system (DOS). The Makeboot disk also comes with two other interpreters allowing you to compile your programs so that they will be loaded in memory at an address as low as $700 (hexadecimal notation).

1.1 Restrictions

Makeboot can be used with virtually any BASIC program that has been successfully compiled by the ABC compiler. The only exceptions are those programs which require the presence of a disk operating system such as Atari DOS 2.0S. Also, you can not use relocatable ABC compiled programs with Makeboot.

When writing a BASIC program that may be used with Makeboot, you should avoid the following:

1. OPENing a disk based file ("Dn:") for any type of INPUT, PRINT, PUT, or GET command;

2. The NOTE and POINT commands;

3. XIO commands which access a disk device; e.g. XIO 33,#1,0,0,"D:FILENAME" to delete a file (other XIO commands for screen fill, etc., may still be used).

Remember that the ABC compiler will still accept these BASIC commands as being perfectly valid, because it has no way of knowing whether or not you intend to use Makeboot after compilation. It's up to you to make sure your compiled program doesn't need DOS.

Section 2
Running Makeboot

a. Compile your BASIC source program according to the instructions in Section 3 of the ABC Reference Manual. Any interpreter address is fine.

b. Run your program to verify that it is working properly in compiled form.

c. Make sure that you have a supply of formatted disks on hand if you will be creating disk booting software. If you need to, use ATARI DOS option 'I' to format disks. Cassettes need no special formatting.

d. Boot the Makeboot diskette as follows: Remove all cartridges, place the Makeboot disk in your powered on drive number one, then turn on the computer.

e. Makeboot will first ask you for an input object filename. Respond with the name of your compiled program. Include device numbers and filename extensions as necessary. The default device is "D1:", and the default extension is ".CMP" (the default extension used by the compiler).

f. The specified file will be read into memory and modified to include all necessary boot code. Makeboot will alert you if the file cannot be opened, or is not in the proper format. Relocatable compiled programs cannot be used with Makeboot.

g. You will then be asked if you want to add other modules to the boot program. Respond with 'N' (for no) at this time. For details on using this optional facility see section 5 in this manual.

h. Next, you will see the prompt "Write to Cassette or Disk?" Remove your source disk from the drive, and respond with the first letter of the desired output device, 'C' for cassette, and 'D' for disk.

   If you type 'D' for disk, Makeboot will prompt you for a drive number. Type 1, 2, 3 or 4 and RETURN, or just RETURN for the default of drive number one. You will now be instructed to insert a formatted disk into the selected

drive. Make sure this disk has nothing important on it, because Makeboot is about to write all over it! When everything is ready, hit RETURN and Makeboot will create a boot-loading disk version of your compiled program. Note that a disk created by Makeboot cannot be duplicated using option 'J' of Atari DOS 2.0S.

If you type 'C' for cassette, Makeboot will instruct you to insert a blank tape into your cassette drive. The double "squawk" means that it is ready to write the cassette. Cue the tape up to the beginning (a leaderless, computer-certified brand is recommended). Press the PLAY and RECORD keys on the cassette drive, hit RETURN and Makeboot will create a self-booting image of your compiled program on tape.

i. When the new disk or tape is ready, Makeboot will ask you to press OPTION to reboot the system, SELECT to return to DOS, or START to write another tape or disk with the same program. If you are using Atari DOS, you must place a disk with DUP.SYS into drive one before choosing a return to DOS. If you want to boot the self-booting program you just made, make sure that there are no cartridges in the system (they should already have been removed to run Makeboot).

Section 3
The New Interpreters

3.1 Purpose

One reason for creating a self-booting program is to reduce the memory requirements for that program. Because self-booting programs are loaded without DOS, programs may be situated in memory starting with page 7 (hex $700). If you normally use the default ABC interpreter ($2600) you would gain 7,936 bytes of space! Even if you used the lowest interpreter normally supplied with ABC ($1F00) you would gain 6,144 bytes. Two interpreters are provided with Makeboot to allow you to compile your programs for the self-booting environment. One is at address $700 (hex) and the other at $E00 (hex). The $700 interpreter is intended for most programs, while the $E00 interpreter is used for programs using the RS-232 driver.

3.2 Instructions For Use

To make use of the new interpreters, you must follow instructions similar to those found in section 5.1 of the ABC Manual which are restated here.

a. When Booting ABC, keep the OPTION console button pressed until the title screen is displayed and the prompt for a new interpreter name appears. Now place the Makeboot disk in drive number one and then respond with either "INTERP.X07" or "INTERP.X0E". Another drive may be used to load the interpreters by simply adding the appropriate disk designation to the name; e.g. "D2:INTERP.X07".

b. When the prompt for the source filename appears, you may remove the Makeboot disk and proceed as though you were compiling normally (see page 7, Section 3 of the ABC Manual).

Remember, Makeboot will make self-booting programs for any interpreter address. However, you will only be able to test your compiled programs under DOS if the address is high enough ($1F00 or more). We suggest that you first try to make a self-booting program that is compiled at the default address and had been run under DOS. If that works ok, you should have no problem with a lower addressed interpreter. Do watch out for the size of string arrays, though. In the self-booting

environment, you may find that you have more than 32K of memory available. However, if you dimension a string array with the size dependent on the amount of free memory, you may exceed the limitation that a string should not be dimensioned larger that 32,767 bytes.

3.3 Compatibility With Older Compilers

The interpreters provided with Makeboot may be of a later revision level than those supplied with your compiler. The interpreters with Makeboot may therefore be slightly longer and different. This means that you cannot create a relocatable program using MKRELO by mixing a program compiled with one of the interpreters on the Makeboot disk with a program compiled with an interpreter on the compiler disk. Other than that, you should have no problem with the newer interpreters.

3.4 Problems Using the RS-232 Driver

The 850 interface module responds and boots like a disk drive if no disk drives are present or powered on during system boot time. If you are making a self-booting cassette, and the load address of your program is $E00 hexadecimal (3584 decimal) or greater, the interface module will be 'booted' for you automatically and load 'beneath' your program. However, if you are creating a self-booting disk (at any address), your program will have to load the RS-232 driver from the 850 by itself. The information to load the RS-232 driver from the 850 is unknown to us at this time, and therefore cannot be printed in this manual. If you are aware of the procedure and care to let us know, we will keep it on file for other interested customers.

Section 4
Running Boot-Load Programs

4.1 Booting Disks

A disk created by Makeboot is loaded just like any other self-booting disk in your library:

a. Turn off your computer and remove all cartridges.

b. Turn on Drive number one and wait for its red BUSY light to go out.

c. Insert the disk created by Makeboot into drive number one and turn on your computer. The program will load and run automatically.

4.2 Booting Tapes

The cassette procedure is also standard for boot-loading tapes:

a. Turn off your computer and remove all cartridges.

b. Rewind the tape created by Makeboot to the beginning. If you have removed the tape, be sure the correct side is facing upward when you put it back in.

c. Press and hold the START key as you turn on the computer. You should hear a single "squawk" from the console. Press the PLAY key on the cassette drive, then hit RETURN. The compiled program will load and run automatically.

4.3 Re-Running Boot-Load Programs

You can automatically RE-RUN a disk or tape program created with Makeboot by pressing the SYSTEM RESET key.

Section 5
Adding Other Modules

Some programs tend to load additional data (such as new character sets) from the disk. Without DOS, this is harder to do because you cannot use the simple OPEN and GET commands. Instead, you would have to use "raw" disk I/O which requires that you place your data on specific sectors on the disk.

Makeboot has another solution to this problem. It allows you to include other files as part of your self-booting program so that they automatically appear in memory when the disk or cassette is booted. The main job is to locate the data at an appropriate address and have the program know where the data will be. Sometimes this requires a two-step process, once to find out the first available address, and the second time to actually create the boot program. The following instructions should clear up this point.

5.1 Procedure to Add Files

a. When Makeboot asks if you want to load additional modules, respond with 'Y' (for yes).

b. Makeboot then requests the name of the file to be included. A default "D1:" is supplied if necessary, but no default extension is added.

c. Makeboot then shows you the first address unused by the compiled program on three different memory boundaries; the very next byte (1), the next 256-byte boundary (256), and the next 1K boundary (1024). It also shows how much memory is left within the Makeboot program buffer for each of those conditions.

d. Respond to the prompt for a load address. You may use any of the addresses displayed, or any arbitrary address that is both above the next available byte and below the limitation imposed by the end of the internal Makeboot buffer. An incorrectly entered number (an empty line or bad digits) will cause Makeboot to print out an error message, time out, then start over with step 'a' above. This error handling allows you to break out of the "load additional modules" loop.

e. Makeboot will proceed to read in the entire file specified and make it part of the self-booting program.

f. You may continue to load as many files as you like. Respond with 'N' when you have no more files to include and proceed normally from step 'h' in section 2 of this manual.

5.2 Choosing a Load Address for an Included File

As can be seen from the above instructions, you may not know the first available address until you actually go through the process of attempting to create the self-booting program. At that point, you may have to recompile you program so that it knows the address that a module will appear, then go through the Makeboot process again.

Although an arbitrary address can be used to load an included file, the self-booting image that is created will contain all the bytes from the lowest address to the highest address loaded. There are no gaps even if an included file is loaded many thousands of bytes above the first available byte shown. This will tend to slow down the boot process but should otherwise not affect normal operation.

Any included files actually become part of the compiled program, and do not have to be accounted for in any way (e.g. you do not have to adjust pointers in the operating system). The memory that is used for the compiled BASIC INPUT buffer, the GOSUB stack, etc., is allocated beyond any file that is included. Do not expect to load a file and consider that to be the last location used by the compiled code. PEEK(106), and PEEK(14)+256*PEEK(15) still gets you pointers to the end of memory and end of compiled basic, respectively.

Do not include standard assembly language files using this method without stripping them of any header and internal segment boundaries. Makeboot includes the added files as strictly raw, uninterpreted data. If your file cannot be treated as a simple block of data, it probably will not be successfully used in this environment.

Warranty Information

Monarch Data Systems, Inc., warrants to the original purchaser that this Monarch Data Systems, Inc. program diskette (not including the computer programs) shall be free from any defects in materials or workmanship for a period of 90 days from the original date of purchase. If a defect is discovered during this 90-day warranty period, Monarch Data Systems, Inc. will repair or replace the diskette at Monarch Data Systems, Inc.'s option, providing that the diskette and proof of purchase are delivered or mailed, postage prepaid, to Monarch Data Systems, Inc.

This warranty shall not apply if the diskette:

-- Has been misused, or shows signs of excessive wear;
-- Has been damaged by the playback equipment, or;
-- If the purchaser causes or permits the diskette to be serviced or modified by anyone other than Monarch Data Systems, Inc.

Any applicable implied warranties, including warranties of merchantability or fitness, are hereby limited to 90 days from the original date of purchase. Consequential or incidental damages resulting from a breach of any applicable express or implied warranties are hereby excluded.

Notice

All Monarch Data Systems, Inc., computer programs are distributed on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of such programs lies with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

Monarch Data Systems, Inc., shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss or damage caused or alleged to have been caused directly or indirectly by computer programs sold through Monarch Data Systems, Inc. This includes but is not limited to any interruption of service, loss of business or anticipatory profits, or consequential damages resulting from the use or operation of such computer programs.

The provisions of the forgoing warranty are subject to the laws of the state in which the diskette is purchased. Such laws may broaden the warranty protection available to the purchaser of the diskette.

NOTES

Monarch Data Systems, Inc.

P.O. Box 207, Cochituate, Massachusetts 01778