BD Software C Compiler
Addenda Information for v1.50a

Leor Zolman
BD Software, Inc.
P.O. Box 9
Brighton, Ma. 02135

This document contains supplementary documentation for BDS C v1.50a. The first section explains some potential incompatibilities between the software (as configured for distribution) and certain computer systems. The second section lists new features of the compiler and libraries. The final section lists bugs found in the v1.50 release that have been corrected in this version.

Potential System Incompatibilities

The following points regarding compiler configuration/operation are not made entirely clear in the v1.5 User's Guide:

- The v1.50a run-time package file-I/O mechanism for BDS C presumes that the target programs are being run on a standard CP/M system where the "user area" numbers may range from user 0 to user 31. Since only 5-bits are allocated for user-number memory within the internal file descriptor routines, standard BDS C generated COM files may not run on certain "CP/M-like" operating systems which support user numbers larger than 31. In order to fix this problem, the symbol USAREA has been added to the configuration area at the start of the run-time package source file (CCC.ASM). If you are experiencing problems opening files on your non-standard CP/M system, try changing this symbol to FALSE, re-assembling CCC.ASM to yield a new C.CCC, and re-linking your C program with this new version of C.CCC. Note that while this fix will allow your programs to run, you will no longer be able to make use of the user-number prefix feature when specifying filenames at run-time. All files will be expected to reside in the currently-logged user area. Drive prefixes may still be specified, of course.

- A dangerous conflict was present in v1.50 involving the CDB debugger and the C.CCC run-time package: when BDS C programs are intended to be run under the debugger, they are compiled with the CC option -k. This causes the

generated code to contain "RST 6" instructions (by default) which are inserted at various points throughout the program for debugging purposes. The first thing CDB does, when invoked, is to write some instructions into restart vector area 6 (at location 30h in the base page of CP/M) in order to trap the RST 6 instructions compiled into the object program about to be debugged. If the "RST 6" vector is already in use, say for a terminal I/O interrupt handler, then CDB overwrites the interrupt handler and crashes the system. Also, the BDS C run-time package (C.CCC, source in CCC.ASM) for v1.50 had itself written into the "RST 6" location in order to allow programs compiled with the **-k** option to run stand-alone (without CDB in memory). This caused **any program** compiled under v1.50 to crash on systems where the "RST 6" vector was being used for important system I/O, e.g. the DEC Rainbow 100.
In v1.50a, the run-time package contains some new options to select whether or not a restart location is to be initialized for stand-alone **-k** programs, and if so, to select **which** restart location shall be used. By default, the package comes configured **not** to write into **any** restart locations, to prevent any programs from coming up destroying vital interrupt handlers and crashing the system.
If you know that the "RST 6" vector is safe to use on your system, simply change the equated "USERST" symbol in the file CCC.ASM to TRUE, and reassemble CCC.ASM to yield a new C.CCC which supports the stand-alone execution of programs compiled with **-k**.
If "RST 6" is already in use on your system, and you know of an alternative restart vector that is free for use by the CDB mechanism, then edit both of the symbols "USERST" and "RSTNUM" in CCC.ASM, create a new C.CCC, and then go create a new CDB (as described in the CDB documentation) to use the new restart location.
If you have no unused restart locations on your system, you cannot use CDB.

- The bios and bdos library functions make certain assumptions about which registers CP/M's BDOS uses to return values from system calls. These assumptions are valid under all CP/M systems, but do not necessarily hold true under certain CP/M "look-alike" systems (such as SDOS or CDOS). If you are trying to use one of these system call interface functions and things don't seem to be working correctly, check to make sure your operating system returns BDOS values in the HL register and BIOS values in the A register...if this is not the case, you must rewrite the bdos and/or bios functions to obtain operating system return values from the proper registers.

- The bios library function as supplied with BDS C makes the assumption that location 0000h of your system (the first instruction of the CP/M base page) contains a direct jump to the second entry (**wboot**) in the BIOS jump vector table. If this is not the case, such as on the Xerox 820, the bios function will not work correctly on your system and you must rewrite it to compute the address of the the BIOS vector table in whichever manner is appropriate for your particular system.

New Functions and Features

This section lists new BDS C functions and features that are either new for v1.50a, or were mistakenly omitted from the User's Guide:

1. The following library function have been added to the standard library:

   - int fappend(name, iobuf)
     char *name;
     FILE *iobuf;

     Just like the fcreat function, except that if the file already exists then the next output written to it is appended onto the end of the file. This function should only be used on **text** files.

   - lprintf(format, arg1, arg2, ...)
     char *format;

     Like printf, except the output is directed to the CP/M "LIST" device instead of to the console.

   - int index(str, substr)
     char *str, *substr;

     Returns the position of string substr within string str, or -1 if not found.

   - int memcmp(source, dest, length)
     char *source, *dest;
     unsigned length;

     Does a quick memory comparison of two blocks of memory length bytes long. Returns TRUE (1) if identical, or FALSE (0) if not identical. This function is coded in assembly language and runs very fast.

2. The CLOAD.C utility, written by Will Colley, has been added to simplify the CASM assembly-to-CRL conversion procedure. CASM.SUB has been modified to use CLOAD.COM, and it is no longer necessary to give a "SAVE" command to save a newly-created CRL file to disk. CLOAD performs the same task as the standard LOAD.COM CP/M utility, except it allows out-of-sequence data that would draw an "INVERTED LOAD ADDRESS" message from LOAD.COM.

3. The CP.C file copying utility, supplied as a sample source program, has been given a new "verify" option. See CP.C for detailed usage.

4. A new sample program called DI.C has been provided. This is a simple file comparator utility for quick verification of the equality of (or minor disparity between) two versions of a file.

5. The TELED.C telecommunications program has been improved. CRC ability has now been added, there are some new modes to allow easier user-to-user communications, and the file transfer mechanisms have been cleaned up.

6. The WILDEXP.C wild-card expansion utility has been expanded to allow disk drive and user area specifiers on wild card designations. A wild-card user area specifiers searches through all user areas between 0 and 15 (you can make it search 0-31 by modifying the source), but this takes a little while to do.

Corrected Bugs

The following is a list of bugs detected in version 1.50. **All have been corrected for version 1.50a:**

1. Only the **first** use of an undeclared identifier had been drawing an appropriate message. Now, all uses of an undeclared identifier are diagnosed.

2. The constant-expression evaluation mechanism did not correctly evaluate expressons involving negative constants.

3. The default drive mechanism, for both the CC and CLINK commands, had some serious problems that caused the wrong logical drive to be accessed under certain compilation and linkage conditions. For example, take the situation where you have configured CC.COM to look in drive C for default files (as described in the User's Guide, section 1.9.2.1), you are currently logged into drive A, and you are compiling a program on drive B with a command such as:

   ```
   A>cc b:test.c
   ```

   Under these circumstances, any **#include** directives within the file test.c which reference the default library area (by delimiting the filename in angle brackets) caused drive B: to be searched for the file, not the default drive C as directed by the CC.COM configuration block. There were other problems of this nature even more confusing to describe, but it seems that everything works predictably under version 1.50a.

4. Some very early v1.50's did not allow the re-use of structure identifiers. For example, the sequence:

```
struct foo {
        int a;
        int b;
};
struct foo foostruct;
```

would have drawn an "illegal identifier" error at the attempted declaration of foostruct.

5. The CLIB program didn't allow filenames of more than six characters if a disk designator was used.

6. A symbolic definition given a null value caused the compiler to crash. For example:

```
#define BLANK       /* this is a null definition */
```

7. One particular arrangement of whitespace in a function definition confused the declaration mechanism. Here is an example of a declaration that was not processed properly:

```
char *              /* Normally these two lines would    */
func(n)             /* be together. This way, el gronko  */
int n;
{
    ...
}
```

8. The alloc and free library functions, as presented in the Kernighan and Ritchie book, have a misfeature which presented itself in the BDS C library versions of these functions by running out of memory, unexpectedly, following certain tricky sequences of allocation and de-allocation. In particular, the continuous allocation and subsequent freeing of large blocks, with very tiny allocations in between, had caused this problem. The bug has been exterminated by improving the allocation algorithm to re-allocate freed blocks from their heads instead of their tails (thanks to William C. Colley III for this fix).

9. The functions fprintf and sprintf, in some versions, did not handle newlines correctly, due to a bug in the undocumented low level library utility function "_fputc". The symptom was that newlines were converted into single 'r' (ASCII 0Dh) bytes, with no linefeed following.

10. Repeated use of the the execl library function under MP/M or TURBODOS had caused the system to run out of file slots, because execl did not bother to close the file before transferring control to new program just loaded. This has been corrected, at the expense of seven bytes worth of command line text area. Note that this affects the exec and execv functions also, since they both use execl themselves.

11. Format strings passed to printf (and all related functions) caused problems if they happened to end with a single % character.

12. Line numbers given in preprocessor error messages while processing **#include** files were completely wrong.

13. When using CLINK to link overlay segments (via the **-v** option), the error messages:

    Warning! Externals extend into BDOS!
    Warning! Externals overlap code!

    sometimes appeared even though the described condition may not have been true.

14. The longjmp library function would not work when placed in ROM due to an "impure" direct store of temporary data within the body of the function. This was a pretty dumb move on my part; longjmp has been corrected to instead use a temporary scratch word in the run-time package data area.

15. When the CPM symbol in CCC.ASM was set to FALSE and CCC.ASM assembled, an undefined entry point called "clrex" was flagged. This has been corrected by removing the "clrex" jump vector from within the conditional assembly block dependent on the CPM symbol.