DCom©

Basic09 Decompiler

Copyright © 1991, Wayne Campbell

Turns ICode into Source Code

AniMajik
Productions

DCom User Manual Contents

DCom - Basic09 I-Code De-Compiler

Version 03.10.00

Copyright (c) 1991,1992,1993
by Wayne Campbell

All Rights Reserved

DISCLAIMER NOTICE

DCom and its related modules are copyrighted and are not to be distributed as freeware or shareware in any manner or in any form. The contents of this disk and of this document are also protected, and may not be distributed in any fashion without prior written consent of the author, and then ONLY to the person(s) named in the written authorization.

The author accepts NO liability for ANY damage to hardware or software, the cause of which is the MISUSE of this program. DCom is sold on an AS-IS basis with NO allowance for refund or exchange. DCom is believed to be free of programming faults, except as outlined in this document, and therefore, no responsibility is assumed by the author for defects. Every attempt will be made by the author to rectify any problems with the use of DCom relating to the inability of the software to run on a given system, to the extent that it can be determined whether or not the system is the cause of the problem.

NOTICE TO PROGRAMMERS

DCom has been written in such a way as to dis-allow the de-compilation of itself. DCom will only work on Basic09 I-Code files. It will NOT work on 6809 OBJECT files, Pascal P-Code files, or C or Pascal derived OBJECT files.

The 'protect' feature allows authors of Basic09 programs for the Color Computer 6809 OS9 Level 1 & 2 to protect their commercial programs from de-compilation by DCom. Programmers may inquire about this by contacting the retailer from whom they have purchased DCom.

Purchasers will receive a registration card with their purchase of DCom. This registration card, when filled out and returned to the retailer from whom the purchaser purchased DCom, will entitle the purchaser to receive upgrades to DCom up to and including version 3.10.00 at no extra cost.

My design goals were to make DCom as USER FRIENDLY as possible, and to make DCom as COMPLETE as possible. I believe I have accomplished these goals.

Before deciding to try and market DCom, I was writing it soley for the purpose of recovering my own lost source code. After I had written the first revision, I decided that others may be in need of an I-Code de-compiler. This led to the decision to market it. As a result, I am hereby obligated to state that the purpose of this program is as a programming tool, designed to allow Basic09 programmers the ability to recover lost source code. It is NOT intended to allow general users to de-compile copyrighted software. Any such use of this program constitutes piracy, as well as copyright infringement, and is in NO WAY condoned or supported by the author.

Feedback may be sent to:

Wayne Campbell
c/o AniMajik Productions
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

(818) 761-4135

DISK CONTENTS

This Disk Includes:

- DCom - Packed Basic09 I-Code Module(s) (The Program)
- DCP1 - Packed Basic09 I-Code Module(s) (The Program)
- DCP2 - Packed Basic09 I-Code Module(s) (The Program)
- DCP3 - Packed Basic09 I-Code Module(s) (The Program)
- DCP4 - Packed Basic09 I-Code Module(s) (The Program)
- DCom.DOC - This File
- Split - Packed Basic09 I-Code Module (Sample Program)
- Split.SRC - Source Code To Split
- Split.B09 - DCom Output File To Split (Comparison File)

NOTE ABOUT SPLIT:

I've included the Split program as a sample. Compare the B09 source with the SRC file to see the differences between the original source and DCom's output. Also, you may use the I-Code module to test-run DCom to be sure it will run on your system. Split is a program that allows you to split a text file into smaller files, and to specify the number of lines per file. Output files default to 'x.**', where ** = aa thru ?? (aa-az, ba-bz, etc.), but an alternate name may be assigned, in which case, if you assign a name of 'temp', the output files will be named 'temp.aa' - 'temp.??'.

USING DCOM

DCom first checks to see if your page pause is currently set as ON. If it is, DCom turns the pause off while processing.

DCom will de-compile single module files, as well as merged files. With merged modules, each module is written to a separate .B09 file (in normal decompile).

DCom must perform multiple passes on each module, for the purpose of Line Number Reference and Variable Identification, as well as Complex Variable, Multiple - Dimension Array Variable, String Variable, and Paramater Variable Identification and Definition. On larger modules, this can become a lengthy process.

DCom is TOTALLY disk-intensive. I suggest that floppy users use a ram-disk, as this will speed the de-compile up by at least 30%. I use one myself, and it works flawlessly. My ram-disk is setup to be 140K. I usually put the module to be de-compiled in the ram-drive, and CD to the ram for de-compile. This way, DCom is reading the source-file and building/using its work files all on the ram, allowing faster de-compiles.

DCom generates working files during execution,and automatically cleans up after itself when done.

DCOM - USAGE

Required Modules:

- RunB - RunTime Module
- SysCall - System Call Sub-Routine
- Tmode - OS9 Module
- Load - OS9 Module
- Unlink - OS9 Module

Syntax For Use:

Std. Shell: dcom ("pathname"|,"opt"|)
Shell+ : dcom pathname |opt|

Where:

pathname = filename or /path/filename to I-Code to be De-Compiled

And opt (Optional):

-v = Build Output File, Dupe Output to StdOut
-o = Suppress Output File, Output to StdOut only

DCom|ENTER| - Print built-in help
DCom -|ENTER| (or DCom ("-")|ENTER|) - Print built-in help
DCom ?|ENTER| (or DCom ("?")|ENTER|) - Print built-in help
DCom -?|ENTER| (or DCom ("-?")|ENTER|) - Print built-in help

DCom pathname|ENTER| (or DCom ("pathname")|ENTER|) = Build Output File Only

DCom uses the current data directory for its work files, and for the output file(s). The '-O' option may be used to redirect the output to another directory as follows:

dcom /dd/cmds/junkfile -o >/d1/JunkFile.B09

will redirect the output of DCom to a file named JunkFile.B09 on /D1.

NOTE: Using this method will cause merged modules to be decompiled to a single file instead of separate files for each module. This output INCLUDES the printed display of the function line: 'Building Work Files: .....' as described later in this documentation. (In other words, you'll see nothing on your screen except a waiting cursor until DCom completes decompilation of the module(s) specified.)

FUNCTIONS

Building Work Files: .....
Building <modulename>.B09

While writing DCom, I found that waiting for up to an hour to begin seeing printed results was unacceptable (BORING is the word!). I decided to include the above statement because I couldn't really tell where the program was in its execution. It is output to StdOut so the use may tell about how far through the execution DCom is.

For merged files, the above two (2) lines will be repeated for each module found in the I-Code file. The '.B09' file will reflect the name of the current procedure being de-compiled.

The line for building the '.B09' file will be suppressed if the '-o' option is specified. Also, Version 3 de-compiles much faster than Version 1 did, but the above output lines are still included, as it can still take some time to begin to see output

REMARK LINES

The output code generated by DCom is begun with a series of remark lines displaying the numbers corresponding to the total of the item(s) identified, or a function being executed. I chose to use this method of output due to the fact that longer modules can take time to de-compile.

This display is as follows:

```
PROCEDURE sokoban
(* ---------- DCOM I-CODE DE-COMPILER ---------- *)
(* Total Identified Variables      :      140 *)
(* Total Mirror Variables          : -     42 *)
(* Total Un-identified Variables   : +      0 *)
(* Total Complex Type Variables    : +      3 *)
(* Total Sub-Routines              : +      3 *)
(* .                                       --*)
(* Total Program Variables         : =    104 *)
(* ---------------------------------------- *)
(* Total Complex Variables         :       45 *)
(* Total Line Numbers              :       20 *)
(* ---------------------------------------- *)
```

It appears that Basic09 doesn't concern itself with unused variables, beyond allocating the storage for them. Because of this, DCom will NEVER be able to correctly identify unused variables in a Complex Type, although, DCom is now able to correctly identify the NUMBER of variables a TYPE statement was created with, as well as the SIZE (in BYTEs) of the TYPE. Immediately following the variable remark lines, this information is printed as follows:

```
(* TYPE: TP1   Total Variables: x   Type Size: x
(* TYPE: TP2   Total Variables: x   Type Size: x
(* TYPE: TP3   Total Variables: x   Type Size: x
etc.
```

Immediately following the above remark lines, DCom informs you that Statement Construction begins with TYPE DIM and PARAM Statements:

```
(* ------------------------------------------------------*)
(* TYPE DIM and PARAM Statement Construction *)
(* ------------------------------------------------------ *)
```

Following the TYPE DIM and PARAM Statements, DCom informs you that Instruction Statement construction begins:

```
(* ------------------------------------- *)
(* Instruction Statement Construction *)
(* ------------------------------------- *)
```

After the Instruction Statements have been constructed, DCom reports that the current module has been completed:

```
(* ---------- END soloban ---------- *)
```

If there are more modules in the specified file, DCom begins again with the next module.

LINE NUMBERS

Line Numbers are identified from GOTO/GOSUB statements, ON variable GOTO/GOSUB statements, ON ERROR GOTO statements, RESTORE line statements, and IF / THEN line statements. What this means is, if you have included line numbers in your program that are only used as a convenient way to get to some portion of the program quickly, with NO other references, DCom will NOT identify them, as Basic09 uses an integer value that is the offset location relative to the execution offset of the program for branching, and there is no way to know where branches occur, except by branch statements.

This also means that, if you have set some line numbers for subroutines that you haven't yet included the GOSUB references to, they will NOT get identified. You will have subroutines (ending with RETURN) with no line number at the beginning of the routine.

DCom may still, on occasion, identify an invalid reference, but it also handles this in the instruction statement routine, so that the invalid references are skipped. As a result, if DCom identifies a false line number reference at, say, line #20, the printed source will show a jump from line #10 to line #30. The false line number is skipped, and output is unaffected.

VARIABLES

There will be times that DCom will identify a variable that is false. This poses no real problem, as the only place this variable will show up is in the DIM statements. Often, this will not be noticable at first, as the variable will be of the type BYTE, INTEGER or REAL. Sometimes a false BOOLEAN will occur. The noticable false variables will be STRINGs, usually showing a dimensioned size greater than 20,000 bytes in length. Again, no problem. It will only show up in the DIM statements, and can be easily removed by your favorite word processor (or deleting the reference from within Basic09).

(----) VARIABLE NAMES (----)

The most simple way I found of naming variables was by numbering them. In order to establish a difference between variables in a complex type, paramaters, and other variables was to use a letter before the number (the letter also comes before the number because

Basic09 won't allow variable names to begin with a number). The referencing is established as follows:

TYPE Statement Variable: TPx

ATOMIC TYPE:	BYTE	INTEGER	REAL	BOOLEAN	STRING	COMPLEX
*TYPE Variables:	bxxxx	ixxxx	rxxxx	lxxxx	sxxxx	txxxx
PARAM & DIM Variables:	Bxxxx	Ixxxx	Rxxxx	Lxxxx	Sxxxx	Txxxx

*In the TYPE statements, the variables are shown as ixxxx,lxxxx but in the instruction statements, the same variables are shown as .ixxxx in conjunction with the Complex variable. (ie: T0024.i0001)

It is up to the user to rename the variables in his/her program to those suitable to them.

NOTE: Any REAL or STRING variables used that were defined by use rather than by DIMension statement will be DIMed in the de-compile. Therefore, any STRINGS defined in your program by using the form var$ will show up as DIM var:STRING, and any REALS defined by the form var:- (or var-) will show up as DIM var:REAL. Also, I followed convention with STRING length, in that, just as Basic09 assigns a default length of 32 to those strings DIMed as var:STRING, so also does DCom. When it finds a string with a defined length of 32 characters, it builds the DIM statement as var:STRING instead of var:STRING[32].

TYPE STATEMENTS

When DCom encounters a TYPE statement with fewer variables identified than were originally included in the TYPE statement, it detects the places within the TYPE where the unidentified variables were located. As it is impossible to determine the ATOMIC TYPE or SIZE of any individual variable within this gap, DCom uses a 1-dimensional array as a means to 'fill-in' the gap.

EXAMPLE:

Original TYPE:

```
TYPE FRT=orange,apple:BYTE; banana,pear:INTEGER
DIM fruit:FRT
```

Let's say that apple and pear were used, but orange and banana were not.

DCom TYPE:

```
TYPE TP1=g000(1):BYTE; b0000:BYTE; g001(2):BYTE; i0001:INTEGER
DIM T0002:TP1
```

All 'gap' identifiers use a 3-digit number instead of a 4-digit number as used on identified variables. While the incrementing of these variables is done the same as the other variables, they are incremented separately. As a result, you may see variable numbering such as:

```
i0025,i0026:INTEGER; g003(12):BYTE; r0027:REAL
```

DCom also detects the occurance of a variable DIMed to a TYPE for which there was no TYPE statement constructed. In this case, it creates a TYPE statement that defines the entire TYPE as a 1-dimensional BYTE array, as follows:

```
TYPE TP4=g007(221):BYTE
DIM T0028:TP4
```

NOTE: DCom still has major problems with complex types. The construction of TYPE statements, and identification of the variables DIMed or PARAMed to them, is mostly a hit-and-miss operation at best. Continued effort to resolve this problem is currently being addressed, and it is hoped that a solution will be found. However, due to the complexity of (and the lack of information found in) the I-Code, this problem may NEVER be solved.

INSTRUCTION STATEMENTS

Line numbers are inserted in the Instruction Statements in increments of 10. This means that the 1st line number will be '10', the 10th line number will be '100', etc.

There is one area that will be affected in the de-compile. This is the area of multiple instructions in a single instruction statement, using the '\' to separate instructions.

The use of the '\' is a carryover from the RSDOS Basic instruction ':' which allows the 'stacking' of instructions. Most of those who use Basic09 that I know, refrain from using this as much as possible, and when they do, only stack a maximum of 2 or 3 instructions in one line.

In Version 1 I followed this convention, in that if there are more than two instructions stacked in a line, DCom would start a new line with the 2nd '\' and begin counting '\'s again. (ie: 4 '\'s in one line would result in 3 lines of code in the de-compile, 2 lines with 2 instructions, and 1 line with 1 instruction) In Version 2 I have done away with this for ONE main reason. If you have a line of code such as:

```
x=15 \ (* This is a remark
```

Basic09 will remove the remark line, but NOT the '\'. If the next line is the beginning of a FOR loop or some other construct, the printout in de-compile may be confusing. As a result, ALL '\' characters are treated as terminators. So, if you have:

```
PRINT \ PRINT
```

it will show up as:

```
PRINT
PRINT
```

Also, the character '=' has many uses in Basic09, including assignment of variables. Basic09 also allows the use of ':=' in assignment of variables. Therefore, I have chosen to use the ':=' as the assignment operator in DCom. Whenever DCom encounters an '=' in an assignment role it automatically uses the ':=' instead, reserving '=' for conditionals.

REAL NUMBER GENERATOR

DReal was written by Paul Fitch and Rodney Hamilton for the purpose of generating real number values from the 5-byte storage code used by Basic09. I include the copyright notice here, and the same restrictions that apply to DCom apply to DReal as well.

DReal - DCom Real Number Generator

Copyright (c) 1992
by Paul Fitch and Rodney Hamilton

All Rights Reserved

The Real Number Generator does NOT round the values determined. Test loads of source output show that Basic09 won't round them either. Basic09 assumes that the value indicated in the source file was specified by the programmer. For this reason, you may encounter places in the source where you will have to perform manual rounding.

EXAMPLE: 65000. may show up as 65000.0001 in the output.

ACKNOWELEDGEMENTS

Version 03.00.00 of DCom no longer requires (or uses) the external DRPN program, as DRPN is now internal. However, to preserve continuity with the previous versions, I have included the copyright notice for DRPN here.

DRPN - DCom Output Parser

Version 01.07.00
Copyright (c) 1992
by Paul Fitch

Version 02.00.00
Copyright (c) 1992, 1993
by Paul Fitch and Wayne Campbell

All Rights Reserved

DRPN was written by Paul Fitch for use with DCom as a means to create useable source code that can be loaded into Basic09. It is copyrighted, and the same restrictions that apply to DCom apply to DRPN as well.

Thank you for purchasing this product. It REALLY is APPRECIATED!

Wayne Campbell - Author

ACKNOWLEGEMENTS:

My heartfelt thanks go, in no particular order, to:
Melissa Dolash - for her patience and understanding, and her concern when things get rough
Paul Fitch and Rodney Hamilton - for their help with the real number generator and the precedence ordering operations

Paul Pollock and
Alan Sheltra - for their patience and endurance with the answers to a million questions

AniMajik
Productions

AniMajik
Productions

Available from **FAT CAT** Publications®...

Special reprints of the **The OSKer** Magazine (Now out of print). The entire 6 issue set is now available and comb-cound (so pages lay flat). This is over 160 pages of OS9 and OSK enjoymen for only **$15.00** (S&H included in price)

Back Issue of **The "International" OS9 Underground®** Magazine are available at the cover price.
Please write or call for prices and issue info.

Subscriptions to The Underground are still $18.00/yr. (12 issues, US) ($22. Canada, $27. Overseas)

Call (818) 761-4135 for more info or write to:

Fat Cat Publications*
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

Fat Cat Publications is a subsidiary of AniMajik Productions