User's Guide for Version 8 of Icon for CMS

Alan Beale
SAS Institute, Inc.

1. INTRODUCTION

Version 8 of Icon for CMS should run on the IBM 30xx and 43xx families of processors and on other 370-type processors that use CMS under Release 4, 5 or 6 of VM/SP or VM/HPO. This version of Icon will also run under all releases of VM/XA SP, in either a 370-mode or XA-mode virtual machine.

Version 8 of Icon for CMS is distributed on a tape which includes executable modules, test programs, data for the test programs, documentation files, and the C source code. The tape also includes the Icon program library. Printed documentation is included with tapes distributed by the Icon Project at the University of Arizona.

This CMS implementation of Icon is in the public domain and may be copied and used without any restriction. The Icon Project makes no warranties of any kind about the correctness of this material or its suitability for any application. The responsibility for the use of Icon lies entirely with the user.

Version 8 of Icon for CMS is an implementation of Icon that was developed for the UNIX* operating system. Some of its features and commands retain the flavor of UNIX, and thus they are sometimes unlike typical CMS features and commands.

The CMS user should remember that although most CMS commands are not case sensitive, Icon source code as well as arguments and parameters passed to the Icon translator or executor are case sensitive.

---

* UNIX is a trademark of AT&T Bell Laboratories.

2. DOCUMENTATION

The basic reference for the Icon programming language is the book

> The Icon Programming Language, second edition, Ralph E. Griswold and Madge T. Griswold, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990. 365 pages. ISBN 0-13-447889-4. $29.95.

This book is available from the Icon Project at the University of Arizona. It also can be ordered through any bookstore that handles special orders or by telephone directly from Prentice-Hall: (201) 767-9520.

Note that the first edition of this book, published in 1983, describes an older version of Icon and does not contain information about many of the features of Version 8.

A brief overview of Icon is contained in technical report TR 90-6 <1>. Features that have been added to Icon since the book was written are described in the report TR 90-1 <2>. These technical reports, together with this document (ICON LISTING on the tape), provide enough information to write and run simple Icon programs, but anyone who intends to use Icon extensively will need the book.

3. INSTALLING CMS ICON

Installation of Icon is described in the document "Version 8 of Icon for CMS - Installation and Recompilation" (INSTALL LISTING on the tape). Please see that document for installation information.

4. RUNNING CMS ICON - BASIC INFORMATION

Files containing Icon programs must have ICN as their filetype. Such files should not have line numbers or other extraneous information. They may have any record format. Lines can be of any practical length (they need not be limited to eighty characters).

The ICONT translator produces an "icode" file that can be used by the ICONX executor. For example, an Icon program in the file PROG ICN A1 is translated by entering this at the Ready; prompt

```
icont prog
```

The result is an icode file with the name PROG ICX A1. This file can then be executed by entering the following:

```
iconx prog
```

When a filename (such as PROG) is given to the Icon translator (ICONT), the translator automatically supplies the filetype of ICN. The executor (ICONX) automatically supplies the filetype of ICX and filemode A1. Thus, the filetype and filemode should not be entered. ICONT and ICONX can be run by entering their names next to a file name on the FILELIST screen followed by /N.

There are two ways to run an Icon program for the first time: (1) as shown above, the translation command (ICONT) and then the execution command (ICONX) can be entered; or (2) the two steps of running the translator and the executor can be combined into one step by using the -x option of ICONT, as:

```
icont prog -x
```

The translator (ICONT) will run the executor (ICONX) only if translation was successful. The two-step procedure must be used if redirections or C runtime options are to be passed to the executor (see Sections 6.1 and 6.5).

After an Icon program has been run in either of the two ways, the icode file (for example, PROG ICX A1) is left on your A disk; this program may be executed subsequently with only the execution command (ICONX). Note that the icode file is put on the A disk regardless of which disk contains the Icon source file.

5. TESTING THE INSTALLATION

There are a few Icon programs on the distribution tape that can be used for testing the installation and getting a feel for running Icon.

hello icn — This program prints the Icon version number, identifies the host computer, prints the date and time, and gives a message to the world. Run this test as

```
icont hello
iconx hello
```

Note that this can be done in one step with

```
icont hello -x
```

kross icn — This program prints all the ways that two words intersect in a common character. The file KROSS DAT contains typical data. Run this test as

```
icont kross
iconx kross <kross.dat
```

meander icn — This program prints the "meandering strings" that contain all subsequences of a specified length from a given set of characters. The file MENADER DAT contains test data. Run this test as

```
icont meander
iconx meander <meander.dat
```

roman icn — This program converts Arabic numerals to Roman numerals. Run this test as

```
icont roman -x
```

and provide some Arabic numbers from the terminal. Enter the word EOF to stop the program.

If these tests work, the installation is probably correct and it should be a running version of Icon.

6. MORE ON RUNNING ICON

For simple applications, the instructions for running Icon given in Section 4 may be adequate. The ICONT translator supports a variety of options that may be useful in special situations. There also are several aspects of execution that can be controlled with environment variables. These are listed here. Users who are new to Icon may wish to skip this section on the first reading but come back to it if they find the need for more control over the translation and execution of Icon programs.

6.1 COMMAND-LINE PROCESSING

Standard input and standard output are received from and sent to the terminal. They can be redirected using greater-than or less- than signs:

```
iconx prog <infile.data >outfile.data
```

This command will run a translated program in PROG ICX A1; it will use INFILE DATA A1 as input; it will use OUTFILE DATA A1 for output. Note that in a redirection, a period must be placed between the CMS filename and filetype. For an input redirection, the file may exist on any accessed disk; for output, a filemode of A1 is assumed. You can also specify an explicit filemode letter, for example, >outf.data.c.

If you run an Icon program from an "old-style" EXEC, you must specify the standard file redirections differently. Surround the fileids with parentheses and omit the period between the filename and filetype, and between the filetype and filemode. For example,

```
ICONX PROG <(INFILE DATA) >(OUTFILE DATA)
```

6.2 ARGUMENTS

Arguments can be passed to the Icon executor by appending them to the command line. Such arguments are passed to the main procedure as a list of strings. For example,

```
iconx readrite text.data readrite.out
```

runs the icode file READRITE ICX A1, passing its main procedure a list of two strings, "text.data" and "readrite.out". These arguments might be the names of files that the program reads from and writes to. For example, the main procedure of READRITE ICN (on the distribution tape) begins as follows:

```
procedure main(a)
   if *a = 0 then a := $< "readrite.icn", "*" $>
   in := open(a$<1$>) | stop("cannot open input file")
   out := open(a$<2$>,"w") | stop("cannot open output file")
                .
                .
                .
```

Note that the sequences $< and $> may replace the left and right brackets in CMS Icon.

When you use the -x option to execute a program after translation, you can specify program arguments after -x, for example:

```
icont readrite -x text.data readrite.out
```

Keep in mind that the "old-style" EXEC language converts all program arguments to uppercase characters and truncates them to a maximum of eight characters. Therefore, if the previous example is invoked from an old-style EXEC, the program arguments received by ICONT will be

```
READRITE
-X
TEXT.DAT
READRITE
```

If you plan to execute an Icon program from an "old-style" EXEC, you should code your program to accept arguments in this style.

6.3 THE TRANSLATOR

The ICONT translator can accept several Icon source files at one time. When several files are given, they are translated and combined into a single icode file whose name is derived from the name of the first file. For example,

```
icont prog1 prog2
```

translates the files PROG1 ICN A1 and PROG2 ICN A1 and produces one icode file, PROG1 ICX A1.

A name other than the default one for the icode file produced by the translator can be specified by using the -o option, followed by the desired name. For example,

```
icont -o pb.icx PROG
```

produces the icode file named PB ICX A1 rather than PROG ICX A1.

If the -c option is given to ICONT, the translator stops before producing an icode file, leaving intermediate "ucode" files with the filetypes U1 and U2 for future use (normally they are deleted). For example,

```
icont -c prog1
```

leaves PROG1 U1 A1 and PROG1 U2 A1, instead of producing PROG1 ICX A1. These ucode files can be used in a subsequent

ICONT command by using the U1 name. This saves translation time when the program is used again. For example,

```
icont prog2 prog1.u1
```

translates PROG2 ICN A1 and combines the result with the ucode files from a previous translation of PROG1 ICN A1. Note that only the U1 name is given. The final qualifier can be abbreviated to .u, as in

```
icont prog2 prog1.u
```

Ucode files also can be added to a program using the link declaration in an Icon source program as described in <2>.

The informative messages from the translator can be suppressed by using the -s option. Normally, both informative messages and error messages are sent to standard error output (the terminal).

The -t option causes &trace to have an initial value of -1 when the icode file is executed. Normally, &trace has an initial value of 0.

The option -u causes warning messages to be issued for undeclared identifiers in the program.

Icon has several tables related to the translation of programs. These tables are large enough for most programs, but translation is terminated with an error message, indicating the offending table, if there is not enough room. If this happens, larger table sizes can be specified by using the -S option. This option has the form -SXn, where the letter X is replaced by one of the letters below to specify the table and n is the number of storage units to allocate for the table.

letters		default values
c	constant table	100
f	field table	100
g	global symbol table	200
i	identifier table	500
l	local symbol table	100
n	line number table	1000
r	record table	100
s	string space	20000
t	tree space	15000
C	code buffer	15000
F	file names	10
L	labels	500

The units depend on the table involved, but the default values can be used as guides for appropriate settings of -S options without knowing the units. For example,

```
icont -Sc200 -Sg600 prog
```

translates PROG ICN with twice the constant table space and three times the global symbol table space that ordinarily would be provided.

6.4 ENVIRONMENT VARIABLES

When an icode file is executed, several environment variables are examined to determine execution parameters. Except for MEMMON, the values assigned to these variables should be numbers. In CMS, environment variables are implemented as GLOBALV variables, in the group CENV.

Environment variables are particularly useful in adjusting Icon's storage requirements. This may be necessary if there is insufficient memory to run programs that require an unusually large amount of data. Particular care should be taken when changing default sizes: unreasonable values may cause Icon to malfunction.

The following environment variables can affect Icon's execution parameters. The default values are listed in parentheses after the environment variable name:

- TRACE (undefined). This variable initializes the value of &trace. If this variable has a value, it overrides the translation-time -t option.

- STRSIZE (65000). This variable determines the initial size, in bytes, of the region in which strings are stored.

- HEAPSIZE (65000). This variable determines the initial size, in bytes, of the region in which Icon allocates lists, tables, and other objects.

- STATSIZE (20480). This variable determines the initial size, in bytes, of the static region in which co-expressions are allocated.

- STATINCR (STATSIZE/4). This variable determines the amount by which the static region is expanded if necessary.

- COEXPSIZE (2000). This variable determines the size, in 32-bit words, of each co-expression block.

- MSTKSIZE (10000). This variable determines the size, in words, of the main interpreter stack.

- MEMMON (undefined). This variable specifies a file name to which memory-monitoring data should be written. (This data can be interpreted by the MEMSUM program in the Icon program library.)

For example, to set the value of the environment variable TRACE to -1, and then invoke the translated Icon program PROG ICN A1, the following is entered:

```
GLOBALV SELECT CENV SET TRACE -1
ICONX PROG
```

6.5 C RUNTIME OPTIONS

CMS Icon is implemented using the SAS/C (r) compiler. The SAS/C runtime environment supports several runtime options which may be useful in some circumstances. These options all begin with the = character, and may appear anywhere on the command line.

Useful options include:

- =/nnnK - This option establishes the total amount of space managed by Icon for all its regions. Because CMS Icon is an "expandable regions" implementation of Icon, each region can expand beyond its initial size; however, the total allocated space is limited by this option. The default is =/500K.

- =warning - This option enables warning messages from the SAS/C environment. These messages may be helpful if an Icon program misbehaves, especially in performing I/O. By default, Icon suppresses all these messages.

C runtime options must be passed directly to ICONX. That is, if they are used with ICONT, they apply only to the execution of ICONT, even if the -x option is used to invoke ICONX after translation is complete.

7. FEATURES OF CMS ICON

CMS Icon supports all the features of Version 8 of Icon, with the following exceptions and additions.

- Because most IBM 3270 terminals and emulations cannot directly enter brackets, most users will require substitutions in Icon source programs. The two-character sequence $< can be substituted for the left bracket, and $> can be substituted for the right bracket. For example, the following line of Icon code for MVS truncates the string TEXT to sixty characters:

```
text := text$<1:61$>
```

  Brackets (EBCDIC numbers X'AD' and X'BD') can, of course, be used if they can be entered.

  Similarly, the sequences $(and $) can be used in place of the left and right braces.

  Note that either the EBCDIC unbroken bar (X'4F') or the broken bar (X'6A') may be used as the Icon or operator.

- The collating sequence (used for the sort() function and for lexical comparisons) of CMS Icon is that of EBCDIC, the native CMS character set. Similarly, hexadecimal and octal escape sequences are given EBCDIC rather than ASCII interpretations, for instance, the string "\x40" prints as a blank, not as the @ symbol. Note that this may cause Icon programs developed under another system to produce different results using CMS Icon.

  Note that in EBCDIC \n and \l are considered to be different characters (X'15' and X'25' respectively), even though they are identical in ASCII.

- The Icon control character notation \¬x produces the EBCDIC equivalent of the ASCII character control-x, for x any "normal" character. If x is an EBCDIC character without an ASCII equivalent, the value of \¬x is completely meaningless.

- The keyword &ascii produces the set of characters which are EBCDIC equivalents to characters in the standard 128-character ASCII set, according to one popular interpretation. If &ascii is converted to a string, its characters are in their EBCDIC order, not the ASCII order.

  The other cset keywords (&lcase, &ucase, &digits, &cset, &letters) are as defined by The Icon Programming Language.

+ Input and output can be redirected (see Section 6.1).

+ In an Icon program, a file name may be specified with spaces or periods between the filename, filetype, and filemode; for example, the following open expressions are equivalent:

```
intext := open("dickens chapter1 a1")

intext := open("dickens.chapter1.a1");
```

  Any messages generated by the translator or executor about a file name (or about the Icon program itself) will have a period (rather than a space) between the filename and the filetype.

+ The following special names can be used in redirection commands (see Section 6.1) or as an argument to open() in Icon programs.

reader (rdr)	for input from the virtual reader
printer (prt)	for output to the virtual printer
punch (pun)	for output to the virtual punch
terminal (term, *)	for input to or output from the terminal
ddn:anyname	for input or output to the file defined by DDname anyname.

  Also a null file name ("") can be used to process an empty (DUMMY) file.

  Pipes are not supported. A file cannot be opened with the "p" option.

+ CMS Icon supportes three different I/O modes, selected by options in the second argument of open. These modes are translated ("t"), untranslated ("u") and record-structured ("s"). (The translated and untranslated modes are sometimes called "text" and "binary".) The default mode is translated.

  When a file is processed in translated mode, it is treated by Icon as a stream of characters, with each record or line break in the file represented as a new line character ("\n"). (If a translated file is the virtual printer, or has filetype LISTING, the form feed ("\f") and carriage return ("\r") characters can also be used to effect page formatting.) When a file is processed in untranslated mode, it is treated by Icon as a stream of characters, with record breaks ignored. When a file is processed in structured mode, each record is treated by Icon as a line, but no character represents the line break. Note that when a fixed for-

mat file is processed in translated mode, trailing blanks are ignored on input, or added as necessary on output. No similar processing is performed in record-structured mode; in untranslated mode, the last record of a file will be padded with null characters (X'00') if necessary.

The exact operation of the Icon I/O functions in each mode is as follows:

- read(), write() and the ! operator process a file a "line" at a time. In translated or untranslated mode, read() and ! read to the next new line character, and write() writes a new line character after the rest of its output. In translated mode, read() and write() therefore actually do read or write a line of the file, unless (1) on input, the file contains a physical new-line character, or (2) on output, a line is too large for the file format, in which case it will be split. Observe that, in untranslated mode, read() and write() have nothing to do with the way the file is divided into records.

- In record-structured mode, read(), write() and ! process a file a record at a time. If the length of a record generated by write() is incompatible with the file format, the write call will fail. In record-structured mode, the new-line character is just another EBCDIC character.

- reads() and writes() process a file a character at a time. In untranslated or record-structured mode, record breaks are ignored. In translated mode, a record break is read as a new-line character, and if a new-line character is output, a record break is forced.

Translated mode is usually to be preferred, except when processing a file that might contain control characters. In this case, untranslated mode is to be preferred unless the record structure of the file is significant, in which case record-structured mode should be used. Even though record-structured mode is the I/O mode closest to standard CMS I/O, translated mode is usually preferred because of the inflexibility of record-structured mode for files with fixed-length records.

+ Whether the seek() and where() functions can be used for a file, and the meaning of the results, depend on the file's attributes, and on whether it was opened in translated or untranslated mode. Three important cases are

- seek() and where() can be used with most files opened in translated mode, but the file position does not represent a count of characters. Seeking to a negative file position is not supported.

- For a file with fixed record format opened in untranslated mode, seek() and where() are fully supported, and have the same meaning as in UNIX.

- seek() and where() cannot be used with most other files opened in untranslated or record-structured mode, except for seeks to positions 1 and 0.

The supported functionality is the same as that of the C library functions fseek and ftell. For further information on the behavior of these functions, see the SAS/C Library Reference manual <3>.

+ The default attributes of a file created by the Icon open() function depend on whether it was opened in translated or untranslated mode. For translated mode, the attributes are record format V with maximum record length 65535. For untranslated mode, the attributes are record format F with record length 1. Note that the latter attributes allow full use of the seek() and where() functions, and inhibit padding with nulls at the end of the file.

+ When an Icon program reads from the standard input file, a prompt of "iconx:" appears. No prompt appears for any other terminal input file. The terminal may not be opened as a bidirectional file (open mode "b").

+ End of file can be signalled from the terminal by entering EOF. This string must be in capital letters, and may not have any trailing spaces.

+ CMS Icon supports an optional third argument to the open() function which can be used to specify file attributes. The third argument is an "atttribute string", which specifies system-dependent information about the file. For example, using CMS Icon, the call open("my output a1", "c", "recfm=f,reclen=80") can be used to create a new file named MY OUTPUT A1, with fixed-length 80-byte records. If there is no third argument, default attributes are assumed.

+ The attribute string should contain one or more keywords, separated by commas. The keywords you may find useful include:

recfm=f/v	The file's record format. Only f or v may be specified (e.g., don't try fb).

reclen=nnn	The file's maximum record size. This does not include the carriage control byte for LISTING format files.
print=yes	To cause the file to be generated in LISTING format
page=nn	Specifies the number of lines per page for a print file
eof=xxx	Specifies the end of file string for a terminal file
prompt=xxx	Specifies a prompt to appear on each read from a terminal file

- The system() function can be used to execute a CP or CMS command or EXEC from Icon. For instance, system("query rdr") invokes the CP QUERY command to display the contents of the virtual reader. Because Icon runs in the CMS user area, you cannot use the system() function to call any CMS command or program that also runs in the user area.

  The system() function can also be used to pass commands to an active subcommand environment. For instance, if XEDIT is active, system("xedit: stack") can be used to transfer the current XEDIT line to the CMS stack.

8. THE ICON INTERFACE TO EXEC2/REXX

Version 8 of CMS Icon includes an interface to the EXEC2 and Rexx command languages which can be used to fetch and set EXEC variables from an Icon program. This functionality is provided via the procedures in REXX ICN. These procedures in turn use the Icon callout() procedure to call C functions that interface to EXEC2 and Rexx. See the technical report TR 90-8 <4> for more information on callout().

The procedures provided in REXX ICN are as follows:

- RexxActive - to determine whether an EXEC is active
- RexxVar - to return the value of an EXEC2 or Rexx variable
- RexxSet - to assign a value to an EXEC2 or Rexx variable
- RexxDrop - to drop an EXEC2 or Rexx variable
- RexxAll - to return an Icon table containing the names and values of all defined EXEC2 or Rexx variables

For more detailed information, see the comments in the source of REXX ICN.

10. THE ICON PUBLIC LIBRARY

The Icon public library is a collection of Icon source code and data files contributed by Icon users. It is described in technical report TR 90-7 <5>. The organization of the files is somewhat different from the organization described in that document, due to the absence of "directories" on CMS. Also, the Idol preprocessor for "object-oriented Icon" is not present on the tape, as it is not yet functional on CMS.

On CMS, the files described in TR 90-7 as part of the "procs" directory are assigned filemode 1 on the Icon installation minidisk. These files contain useful procedures, many of which are referenced by other public library members. These files should be compiled using the -c option, so that the resulting .u1 and .u2 members can be referenced later.

Similarly, the files described as part of the "progs" directory in TR 90-7 have been assigned filemode 2. These files contain the source for various Icon applications. Many of the programs include link declarations referencing these files. Data files for these programs (the "data" directory) are also assigned filemode 2.

Most of the programs and procedures in the Icon public library were developed on UNIX and other operating systems dissimilar to CMS. Some of them also have dependencies on the ASCII character set. For these reasons, it is possible that some of these programs will prove either useless or non-functional when run on CMS.

Several programs and procedures of particular interest on CMS have been added to the public library for CMS Icon. These are not yet described in TR90-7. They include:

- EBCDIC ICN - A collection of procedures to assist in running Icon programs dependent on either the ASCII or EBCDIC character set on other systems. It is possible that some ASCII-dependent programs in the program library can be easily made portable by use of these procedures.

- ICVT ICN - A program to convert an Icon program to an equivalent program more easily edited on the 370, replacing brackets and braces with the corresponcing digraphs, or vice versa.

See the source code for these files for further details on their use.

11. KNOWN BUGS AND LIMITATIONS

A list of known bugs in Icon itself is given in <2>. At this time, there are no known bugs specific to CMS Icon.

12. REPORTING PROBLEMS

Problems with CMS Icon should be noted on a trouble report form (TROUBLE ICONFORM) on the distribution tape) and sent to

Icon Project
Department of Computer Science
Gould-Simpson Building
The University of Arizona
Tucson, AZ 85721
U.S.A.
(602) 621-4049
icon-project@cs.arizona.edu (Internet)
... {uunet, allegra, noao}!arizona!icon-project (uucp)

If a program is involved, a copy of the program will be appreciated. The program may be necessary to provide help.

13. REGISTERING COPIES OF ICON

Those who received a copy of Version 8 of Icon for CMS directly from the Icon Project are registered users and will receive the Icon Newsletter without charge. This Newsletter contains information about new implementations, updates, programming techniques, and information of general interest about Icon.

Those who received a copy of Version 8 of Icon for CMS from a source other than the Icon Project should fill out a registration form (REGIS ICONFORM on the distribution tape) and send it to the Icon Project at the address listed above. This will entitle them to a free subscription to the Icon Newsletter and assure that they receive information about updates.

14. ACKNOWLEDGEMENTS

The design and development of the Icon programming language was supported, in part, by the National Science Foundation under grants MCS75-01397, MCS79-03890, MCS81-01916, DCR-8320138, DCR- 8401831, and DCR-8502015.

Many individuals contributed to the design and implementation of Icon. The principal ones are Cary Coutant, Dave Gudeman, Dave Hanson, Tim Korb, Bill Mitchell, Kelvin Nilsen, Janalee O'Bagy, Gregg Townsend, Ken Walker, and Steve Wampler.

This implementation of Icon for CMS was created by Alan Beale and Tim Hunter of SAS Institute, Inc.

REFERENCES

1. R. E. Griswold, An Overview of Version 8 of the Icon Programming Language, The Univ. of Arizona Tech. Rep. 90-6, 1990.

2. R. E. Griswold, Version 8 of Icon, The Univ. of Arizona Tech. Rep. 90-1, 1990.

3. SAS/C Library Reference Manual, second edition, volumes 1 and 2.

4. R. E. Griswold, Icon-C Interfaces, The Univ. of Arizona Tech. Rep. 90-8, 1990.

5. R. E. Griswold, The Icon Program Library The Univ. of Arizona Tech. Rep. 90-7, 1990.