User's Guide for Version 8.8 of Icon for MVS

Alan Beale
SAS Institute, Inc.

1. INTRODUCTION

Version 8.8 of Icon for MVS should run on the IBM 30xx and 43xx families of processors and on other 370- or 390-type processors that use the MVS (including MVS/XA and MVS/ESA) operating system. MVS Icon is intended primarily for use under the MVS Time Sharing Option (TSO); however, batch usage is also possible.

Version 8.8 of Icon for MVS is distributed on a tape which includes executable modules, test programs, data for the test programs and documentation files. Printed documentation is included with tapes distributed by the Icon Project at the University of Arizona.

This MVS 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.

Because Icon was developed for a wide range of environments, many of its features and commands resemble those of other operating environments such as UNIX[1] and MS-DOS, and may therefore seem strange to the MVS user. In particular, the MVS user should remember that although most TSO commands are not case sensitive, Icon source code as well as arguments and parameters passed to the Icon translator or executor are case sensitive.

2. DOCUMENTATION

The basic reference for Version 8 of Icon is the second edition of the book The Icon Programming Language <1>. 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.

---

[1] UNIX is a trademark of AT&T Bell Laboratories.

The new features of Version 8.8 of Icon are described in technical report IPD210, available from the Icon Project at the University of Arizona.

3. INSTALLING MVS ICON

Installation of Icon is described in the document "Version 8.8 of Icon for MVS - Installation" (INSTALL.DOC on the tape). Please see that document for installation information.

4. RUNNING MVS ICON - BASIC INFORMATION

This discussion assumes you are running Icon from TSO. Batch execution is discussed in a later section.

Files containing Icon programs must have ICN as their final qualifier and be prefixed by the current TSO prefix. Members of partitioned data sets are permitted. 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 data set MY.ICN(PROG) is translated by entering this at the READY prompt (or in SPF menu 6):

```
icont my(prog)
```

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

```
iconx my(prog)
```

When a filename (such as MY) is given to the Icon translator (ICONT), the translator automatically supplies the final qualifier of ICN. The executor (ICONX) automatically supplies the final qualifier of ICX. Thus, the final qualifier should not be entered.

Note that the discussion above assumes that Icon has been installed into a LINKLIST data set, or that the Icon load library is included in your session's STEPLIB. If neither of these are true, you must use the TSO CALL command to invoke Icon. For example,

```
call icon(icont) 'my(prog)'
call icon(iconx) 'my(prog)'
```

Because the CALL command translates arguments to upper case, you may find yourself unable to access certain Icon options via the CALL command. (This is a reason that Icon should be installed into LINKLIST.) Note that with some versions of the TSO CALL command you can use the ASIS keyword of CALL to inhibit upper-case translation.

There are two ways to run an Icon program for the first time: (1) as shown above, using the translation command (ICONT) and then the execution command (ICONX) separately; or (2) combining the two steps of translation and execution by using the -x option of ICONT, as:

```
icont my(prog) -x
```

The translator (ICONT) will run the executor (ICONX) only if translation was successful. The two-step procedure must be used if redirections, environment variables, or C runtime options are to be passed to the executor (see Sections 6.1, 6.5 and 6.6).

After an Icon program has been run in either of the two ways, the icode file (for example, MY.ICX(PROG)) is left undisturbed; this program may be executed subsequently with only the execution command (ICONX).

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. If you did not install Icon yourself, you must copy the Icon source and data files from the id of the installer to your own id before running the commands shown below.

hello.icn — This program prints the Icon version number, identifies the host computer, prints the date and time, and lists the implemented Icon features. Run this test as

```
icont examples(hello)
iconx examples(hello)
```

Note that this can be done in one step with

```
icont examples(hello) -x
```

cross.icn — This program prints all the ways that two words intersect in a common character. The file EXAMPLES.DATA(CROSS) contains typical data. Run this test as

```
icont examples(cross)
iconx examples(cross) <examples.data(cross)
```

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

```
icont examples(meander)
iconx examples(meander) <examples.data(meander)
```

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

```
icont examples(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 you should have 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(fred) >outfile.data
```

This command will run a translated program in PROG.ICX; it will use INFILE.DATA(FRED) as input; it will use OUTFILE.DATA for output.

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 examples(readrite) text.data readrite.out
```

runs the icode file EXAMPLES.ICX(READRITE), 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 EXAMPLES.ICN(READRITE) (on the distribution tape) begins as follows:

```
procedure main(a)
   if *a = 0 then a := $< "examples.icn(readrite)", "*" $>
   in := open(a$<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 MVS Icon.

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

```
icont examples(readrite) -x text.data readrite.out
```

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 my(prog1) my(prog2)
```

translates the members MY.ICN(PROG1) and MY.ICN(PROG2) and produces one icode file, MY.ICX(PROG1).

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 my(prog)
```

produces the icode file named PB.ICX rather than MY.ICX(PROG).

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

```
icont -c my(prog1)
```

leaves MY.U1(PROG1) and MY.U2(PROG1), instead of producing MY.ICX(PROG1). 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 my(prog2) my.u1(prog1)
```

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

```
icont prog2 my.u(prog1)
```

Ucode files also can be added to a program using the link declaration in an Icon source program as described in Section 6.4.

Icon source programs can also be read from standard input (normally the TSO terminal). The argument - signifies the use of standard input as a source file. (The end of terminal input is signalled by entering a line containing only the word EOF.) In this case, the ucode files are named STDIN.U1 and STDIN.U2 and the icode file is named STDIN.ICX.

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 -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.

6.4 THE LINK DECLARATION

Programs which call Icon procedures contained in other files may use the link declaration to declare the dependency. On most systems, the declaration "link subs" indicates that the ucode for required procedures is to be found in the files SUBS.U1 and SUBS.U2. This interpretation, though supported on MVS, is generally inadequate there, due to the use of partitioned data sets for Icon source and ucode files.

When a link declaration, such as "link subs", is processed by MVS Icon, the "search order" is as follows:

1. The files ICONLIB.U1 and ICONLIB.U2, if they exist, are searched for the member SUBS,

2. If this was unsuccessful, the files allocated to the DDnames U1LIB and U2LIB are searched for the member SUBS,

3. If this was also unsuccessful, an attempt is made to locate sequential files named SUBS.U1 and SUBS.U2.

For greater flexibility, the "environment variable" IPATH can be used. To use this facility, add the option "=IPATH=(dsl ds2 ...)" anywhere after the command name when invoking ICONT. For example:

```
icont my(prog) =IPATH=(mathlib setlib)
```

specifies that the ucode libraries MATHLIB.U1/U2 and SETLIB.U1/U2 are to be searched for members named in link declarations. If IPATH is specified, the first two standard search steps above are bypassed, that is, neither U1LIB/U2LIB nor ICONLIB.U1/U2 is searched.

Note that the ISPF interface to Icon provides a more convenient mechanism for specifying the IPATH information.

Also note that if your site licenses the SAS/C compiler, the SAS/C PUTENV command can be used in TSO to permanently assign environment variables.

6.5 ENVIRONMENT VARIABLES

When an icode file is executed, several environment variables are examined to determine execution parameters. The values assigned to these variables should be numbers. Since MVS does not support environment variables in the UNIX sense, they must be simulated by providing extra arguments to the ICONX command. To set the environment variable X to the value Y, you must specify =X=Y as an option on the command line. (If your site licenses the SAS/C compiler, you can also use the SAS/C PUTENV command to assign permanent values to environment variables.)

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. If additional string regions are needed, they may be smaller.

- BLKSIZE (65000). This variable determines the initial size, in bytes, of the region in which Icon allocates lists, tables, and other objects. If additional block regions are needed, they may be smaller.

- 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.

- QLSIZE (5000). This variable determines the size, in bytes, of the region used by the garbage collector for pointers to strings.

For example, to invoke the translated Icon program MY.ICN(PROG) with the environment variable TRACE set to -1, the following is entered:

```
ICONX MY(PROG) =TRACE=-1
```

Note that when you use the -x option of ICONT to invoke ICONX, any command-line environment variable specifications are passed only to ICONT, and will have no effect on execution.

Note that the ISPF interface to Icon provides a more convenient mechanism for specifying environment variables.

6.6 C RUNTIME OPTIONS

MVS 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.

A particularly useful option is:

- =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.

6.7 RUNNING ICON FROM ISPF

Icon Version 8.8 includes an ISPF panel for invoking the Icon translator and/or executor which may be more convenient than the standard procedures, especially if it is necessary to use IPATH or other environment variables. The method of access to this panel will be chosen at the time Icon is installed. Often, it will be accessible as the I option of the U (User Application) panel. See your installer for more information.

The Icon ISPF panel allows you to specify up to 4 environment variable settings, up to 4 IPATH libraries, and also to specify whether U1/U2LIB should be searched for link files. These settings are retained in your ISPF profile.

The Icon ISPF panel is invoked using the ISPF option NEWPOOL(ICON). This means that ISPF variable pools created by Icon programs are kept separate from those for other ISPF applications. See Section 9 for further information on use of ISPF services from Icon.

7. FEATURES OF MVS ICON

MVS Icon supports all the features of Version 8.8 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 MVS Icon is that of EBCDIC, the native MVS 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 MVS 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 is normally interpreted as a TSO file name, that is, the user's current prefix is added automatically. Filename prefixes may be used to request other styles of file name. The file name "dsn:sysl.maclib(dcb)" specifies the data set whose full name is SYSl.MACLIB(DCB); the file name "ddn:sysutl" specifies the file referenced by the DDname SYSUTl. Use of these prefixes may be necessary when Icon programs are run in batch: see section 9.

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

  ```
  dsn:sysout=a  for output to a SYSOUT data set
  dsn:&name     for output to a temporary data set
  *             for input to or output from the TSO
                terminal
  ```

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

- MVS 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 has print attributes (RECFM A), 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 format 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 MVS I/O, translated mode is usually preferred because of the inflexibility of record-structured mode for files with fixed-length records.

o 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 sequential file with RECFM F, FS or FBS 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>.

o 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 RECFM=VB,LRECL=259,BLKSIZE=6160. For untranslated mode, the attributes are RECFM=FBS,LRECL=1,BLKSIZE=4080. 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.

o 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").

o 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.

o MVS 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 MVS Icon, the call open("my.output", "c", "recfm=f,reclen=80,blksize=3200, alcunit=blk,space=400") can be used to create a new file named MY.OUTPUT, with fixed-length 80-byte records, and enough space for 400 3200-byte blocks. If there is no third argument, default attributes are assumed.

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

recfm=f/v/u	The file's record format. Only f, v or u may be specified (e.g., don't try fb).
reclen=nnn	The file's maximum record size. This does not include 4 bytes for V-format control data
blksize=nnn	The file's block size.
print=yes	To cause the file to be generated in print format (RECFM A)
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
alcunit=blk/trk/cyl	Specifies the space allocation unit, blocks, tracks or cylinders
space=nnn	Specifies the amount of space to allocate for a new file
extend=nnn	Specifies the amount of space to request when extending the file
dir=nnn	Specifies the number of directory blocks for a new PDS
rlse=yes/no	Specifies whether to release unused space when the file is closed

- The system() function can be used to execute a TSO command or CLIST from Icon. For instance, system("listds my.data") invokes the LISTDS command to display attributes of the data set MY.DATA.

8. THE ICON INTERFACE TO ISPF

Version 8.8 of MVS Icon includes an interface to ISPF which can be used to display panels, access and modify ISPF variables, etc., from an Icon program. This functionality is provided via the procedures in EXAMPLES.ICN(ISPF). These procedures in turn use the Icon callout() procedure to call C functions that interface to ISPF.

The procedures provided in EXAMPLES.ICN(ISPF) are as follows:

- ISPQry - to determine whether ISPF services are available
- ISPExec - to pass a request for an ISPF service to ISPF
- ISPVcopy - to obtain the value of a variable in an ISPF variable pool
- ISPVrepl - to modify the value of a variable in an ISPF variable pool

For more detailed information, including return code and error handling, see the comments in the source of EXAMPLES.ICN(ISPF).

9. RUNNING ICON IN MVS BATCH

The MVS implementation of Icon was designed for interactive use. However, it can be used in MVS batch if certain guidelines are followed.

One way to run Icon in batch is to run it under the batch terminal monitor program (IKJEFT01). When this is done, the only difference from normal TSO operation is that &input, &output and &errout are allocated to the DDnames SYSIN, SYSPRINT and SYSTERM respectively, rather than to the terminal.

When ICONT or ICONX is run as a normal batch program, without the use of IKJEFT01, certain restrictions and differences apply, as follows:

- o File names specified to Icon in batch will be prefixed by your userid, whch may differ from your usual TSO prefix. Note that if your site does not run RACF or some similar security product, no userid is available. In this situation, each file to be translated by ICONT must be prefixed with dsn:, and must be fully qualified. For example the EXEC statement

```
// EXEC PGM=ICONT,PARM='dsn:mytsoid.hello.icn'
```

  compiles the program contained in the data set MYTSOID.HELLO.ICN, and stores the icode in MYTSOID.HELLO.ICX.

- o The -x option of ICONT is not available in batch.

- o The files &input, &output and &errout are allocated to the DDnames SYSIN, SYSPRINT and SYSTERM respectively.

- o If your site does not run a security product such as RACF, each file name used by an Icon program, either as an argument to open(), or as a redirection, must have a dsn: or ddn: prefix.

- o The system() function may not be used in batch.

- o Be careful to specify lower case in options where necessary, as the options may not be recognized if upper case is used.

10. ICON LIBRARY PROGRAMS

Several programs and procedures of particular interest on MVS have been added to the public library for MVS Icon. These are distributed on the MVS Icon distribution tape. These include:

- o ICONLIB.ICN(EBCDIC) - 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 many ASCII-dependent programs can be easily made portable by use of these procedures.

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

See the source code for these members 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 MVS Icon.

ICONT depends on the ability to allocate a work file using the unit name VIO. If this unit name is not defined at your site, translation will fail. See the Installation document for information on correcting this problem. Alternately, you can allocate the SYSTMP01 DDname to a temporary data set when you run ICONT to bypass the problem.

12. REPORTING PROBLEMS

Problems with MVS Icon should be noted on a trouble report form (ICON.FORM(TROUBLE) 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) 621-8448 (voice)
(602) 621-4246 (fax)
icon-project@cs.arizona.edu (Internet)
... uunet!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 8.8 of Icon for MVS 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 8.8 of Icon for MVS from a source other than the Icon Project should fill out a registration form

(ICON.FORM(REGIS) 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 grants from the the National Science Foundation.

Clint Jeffery, Gregg Townsend, and Ken Walker collaborated with Dr. Ralph Griswold in the development of Version 8.8.

This implementation of Icon for MVS was created by Alan Beale of SAS Institute, Inc.

REFERENCES

1. R. E. Griswold, The Icon Programming Language, Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1990.

2. R. E. Griswold, C. L. Jeffery, G. M. Townshend and K. Walker, Version 8.8 of the Icon Programming Language, The Univ. of Arizona Icon Project Document IPD210, 1992.

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.