Building Version 9.0 of Icon for MS-DOS

Ralph E. Griswold

Department of Computer Science, The University of Arizona

The implementation of Version 9.0 of Icon includes both an interpreter and an optimizing compiler. The interpreter and compiler are largely compatible and almost all Icon programs will run under either, although some features require the specification of options when used with the compiler [1]. The interpreter is appropriate for program development and many production applications. The compiler can be used to get faster execution speeds and a stand-alone executable file that can be distributed without the interpreter.

Note: This document is primarily concerned with building Icon for MS-DOS, but the source code also can be used to build Icon for OS/2; see [2].

1. Requirements

Processors

Version 9.0 of Icon runs on computers with 8086/88/x86-family processors. IBM hardware compatibility is not required. Either MS-DOS or PC-DOS, Version 3.0 or higher, is needed. The interpreter requires at least 500KB of conventional memory to run satisfactorily. The Icon compiler requires a 32-bit processor with at least 4MB of RAM. A fast 486 with considerably more RAM is recommended. The Icon compiler generates C code, which then must be compiled and linked using the C compiler under which the Icon compiler was built.

C Compilers

The implementation is written almost entirely in C and RTL [3], a superset of C for which a translator to C is supplied.

Building the interpreter is straightforward and can be done on a 16-bit platform with most of the popular C compilers that support the large memory model. Building the interpreter requires at least 510KB of free RAM. If there is not sufficient memory, MS-DOS may hang. If this happens, you may need to remove some memory-resident programs to build Icon. Building the compiler requires a 32-bit C compiler. The amount of RAM required depends on the compiler. 4 MB is typical.

To date, the Version 9.0 interpreter for Icon for MS-DOS has been successfully built using the following C compilers:

Borland C++ 3.0
Intel Code Builder 1.0
Microsoft C 6.0
Microsoft C 7.0
Turbo C 2.0
Watcom C/396 9.0
Zortech C++ 3.0

The Version 9.0 compiler for Icon has been successfully built using

Intel Code Builder 1.0
Watcom C/396 9.0
Zortech C++ 3.0

Version 9.0 also has been configured for Metaware High C 2.3, but this configuration has not been tested. The use of a C compiler not listed here involves additional work to provide appropriate conditional code.

Assembly Language

Icon's co-expressions require an assembly-language context switch. An appropriate assembler therefore is necessary to provide this feature when building Icon entirely from the source code. However, object files for context switches are provided, where available, for persons who do not have an assembler. Alternatively, co-expressions can be disabled without otherwise affecting the use of Icon.

Batch Scripts

Some of the batch scripts for building and testing Icon use the call command, which allows a batch script to be run from inside another batch script, with control then returning to the former one. If you are using an early version of MS-DOS that does not support call, you will have to decompose the batch scripts that use it into more elementary ones.

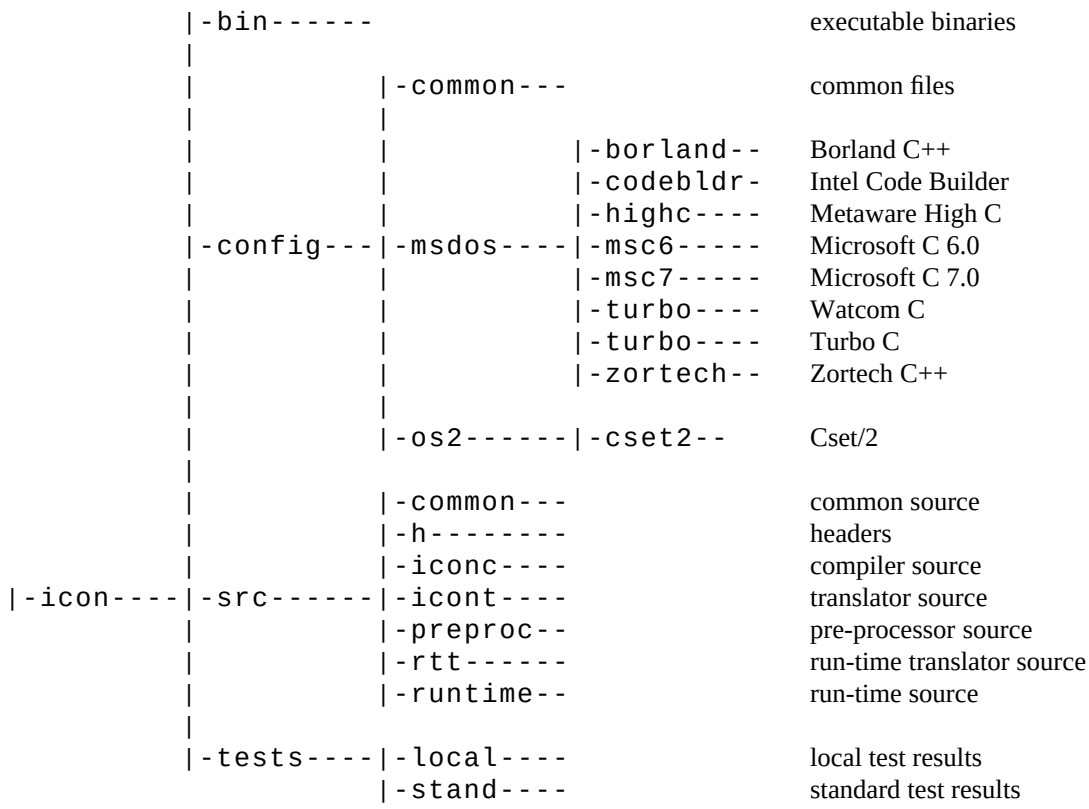
Disk Space

The material on the distribution diskettes occupies about 4.5MB when unloaded. This amount can be reduced by deleting files related to the interpreter and to C compilers that will not be used. About 1.5MB additional space is needed to completely build and test the Icon interpreter, although it is possible to get by with less by building incrementally. Building the Icon compiler requires additional disk space — 3 to 4MB depending on the C compiler used.

2. Organization of the Implementation

The source code for Icon is organized in a hierarchy. Files for various components of Icon are packaged using LHA. A copy of LHA is included in the distribution. Instructions for unloading the files are provided on the distribution diskettes.

If the Icon hierarchy is rooted in \icon, the directories after unloading are:



In the descriptions that follow, path specifications assume that the Icon hierarchy is unloaded in \icon\bin. If the location is different for your installation or requires a drive specification, interpret the path specifications that follow

accordingly.

The distribution diskettes also contain documentation and some tools that may be useful in building and testing Icon. See the README files on the distribution diskettes.

Binary Files

Since rtt.exe cannot be built under all the C compilers that otherwise support Icon, a copy is included in the distribution. Several other executable binaries used in various aspects of building Icon are also included. These executable binaries are installed in \icon\bin.

Source Files

The six source-code sub-directories under src contain the following components of Icon:

common	files common to different components of Icon.
h	header files used by files in the other directories.
iconc	source code for iconc.exe, the optimizing compiler.
icont	source code for icont.exe, the translator and linker that converts an Icon source-language program into a form suitable for the interpreter.
preproc	source code for an ANSI C pre-processor used by the run-time translator.
rtt	source code for rtt.exe, a program used in translating run-time code to C.
runtime	source code for the interpreter, iconx.exe, and run-time systems for the compiler and interpreter.

Configuration Directories

In order to simplify the process of compiling Icon under different C compilers, files that are compiler-specific, such as batch and linker files, are provided in subdirectories of the config directory. The MS-DOS configurations presently are:

borland	Borland C++
codebldr	Intel Code Builder
highc	Metaware High C
msc6	Microsoft C 6.0
msc7	Microsoft C 7.0
turbo	Turbo C
watcom	Watcom C
zortech	Zortech C++

The use of these configuration directories is described in next section. *Note:* Some files contain path information that may need to be changed for a particular configuration.

3. Building Icon for a Supported Configuration

Before starting to compile Icon, be sure your C compiler is properly installed and that any paths that it needs are properly set.

Setting up Files

The first step in the compilation process is to set up the files needed for compilation and linking. If you are using one of the C compilers mentioned above, there is a batch script in the top level of the Icon hierarchy (assumed to be \icon here) whose name corresponds to the C compiler. Running the batch script performs the configuration. For example, if you want to configure Version 9.0 of Icon to compile under Borland C++, just enter

```
borland
```

This batch script copies compiler-specific scripts, source files, and object files to appropriate places in the source-code hierarchy. *Note:* Some files may not be found during copying for some compilers. This is normal, since a common copying script is used and not all configurations have all the files that others do.

A file containing compiler-specific information, status, is also copied into the top level of the Icon hierarchy. Read this file before proceeding, since it may contain information about problems and limitations related to the C compiler.

Configuration Changes

Once Icon is set up for a specific C compiler, there may be a few changes you wish to make before building Icon.

Co-Expressions: Co-expressions are supported by default on most platforms. To disable them, add

```
#define NoCoexpr
```

to \icon\src\h\define.h.

MS-DOS Functions: There are a few functions specially designed for using Icon under MS-DOS that are not part of Icon's standard function repertoire. The functions are described in [4]. These functions normally are included if they are supposed for the C compiler used. If you wish to eliminate them (which decreases the size of iconx by a few thousand bytes), remove

```
#define DosFncs
```

from \icon\src\h\define.h.

Large Integers: Icon has facilities for large-integer arithmetic, but these facilities normally are disabled for 16-bit MS-DOS platforms because they increase the size of iconx substantially (20-30KB). If you have enough RAM and wish to enable large-integer arithmetic, remove the following line from \icon\src\h\define.h:

```
#define NoLargeInts
```

Directory Path: The icon compiler, iconc.exe, needs to know where certain files are located. For most C compilers used to build iconc.exe, this path can be patched after iconc.exe is built. In some cases, however, this is not possible because of the format of executable files. See the file status mentioned in the preceding section.

In cases where patching does not work, it is necessary to decide in advance where the files needed by iconc.exe will be located and to place a definition for RefPath in \icon\src\h\define.h. For example, if the location will be c:\dos, the definition would be

```
#RefPath "c:\\dos\\"
```

Note that the string must end in a backslash.

Use of the Make Utility

The instructions for building Icon in the next section specify the use of batch scripts. A public-domain UNIX-style make utility is included in this distribution. It is unloaded into \icon\bin as make.exe. To use it, move it to a place on your PATH or add \icon\bin to your PATH.

Makefiles are installed when files are set up for a specific C compiler. You may find it convenient to use make, especially if you modify the Icon source code. Be aware, however, that building Icon from Makefiles requires more RAM than building Icon from batch scripts. This may cause problems in some situations. Makefiles also are not available for some steps, either because some programs malfunction when run under make or because files are constructed on the fly and their names cannot be determined in advance.

Compilation

The steps in building Icon follow. Note that the first three steps refer to building the interpreter, while the last two refer to the compiler. If you only want to build the interpreter, stop after Step 3.

The directories mentioned in the following steps are relative to \icon\src.

1. First cd common and run the batch script build. This creates object files used in the various components of Icon.

2. Next cd ..\icont and run the batch script build. This builds icont.exe and installs it in \icon\bin.
3. Next cd ..\runtime and run the batch script build. This builds iconx.exe and installs it in \icon\bin. *Note:* Problems with insufficient RAM are most likely to occur at this step. If the build fails for lack of space or if MS-DOS hangs, reboot with fewer memory-resident programs.

Subsequent steps apply to building the Icon compiler and its support files. As mentioned above, a 32-bit C compiler is required for this.

4. First cd ..\iconc and run the batch script build. This builds iconc.exe and installs it in \icon\bin.
5. This step builds files needed by iconc.exe. A 386 or higher and at least 4MB of RAM are required. Two of the batch scripts assume the Icon interpreter is installed in \icon\bin as built by Steps 2 and 3 above. cd ..\runtime and run the following batch scripts:

header	(creates a header and an object file needed by iconc.exe)
icontrns	(creates C files and a database needed by iconc.exe*)
iconcomp	(creates object files**)
iconlibe	(creates a library needed by iconc.exe*)

The files needed by iconc.exe are copied to \icon\bin.

The scripts flagged with * take a long time. The script flagged with ** takes a *very* long time. The time depends not only on processor speed but also on the compiler used.

4. Installing Icon

After performing the steps specified in the preceding section, \icon\bin should contain the components of Icon indicated above.

To use Icon, the executable files need to be available via PATH. You can add \icon\bin to PATH or move the executable files to a place already on PATH.

iconc.exe needs to know the location of two files that are initially installed in \icon\bin. The files are:

rt.db	a database of information about Icon operations
rt.h	a header file specified in the code iconc.exe produces

In addition, the following files are needed for linking the object files that result from compiling the C files produced by iconc.exe:

rt.lib	an object library that contains run-time operations
dlrgint.obj	a stub for large integers (if large integers are not supported)

If these four files are moved, they should all be placed in the same directory.

iconc.exe is initially configured to expect its files in the directory from which it is executed. In order to be able to run iconc.exe from any location, a path in it must be patched. The program patchstr.exe, in \icon\bin, performs this task. It is used as follows:

patchstr iconc.exe *directory–path*\

where *directory-path* is the full path to the directory in which the four files mentioned above are located. Note that the path specification must be followed by a backslash.

For example, if the files that iconc.exe needs are not moved, the patch step would be

patchstr iconc.exe \icon\bin\

The patching process can be repeated as necessary if the directory is moved.

5. Running the Icon Compiler

Please note: Running the compiler requires significant resources and it may not be practical on slow processors or platforms with limited amounts of memory. Furthermore, it may be necessary to reconfigure your memory extender so that iconc.exe can use all available RAM.

The use of the compiler under MS-DOS differs somewhat from the description given in [1], since some of the operations iconc.exe performs under other operating systems are not practical under MS-DOS. Specifically, the C files produced by iconc.exe are not compiled automatically and subsequently deleted. Instead, these operations must be done after iconc.exe is run.

A batch script, icomp.bat, for using the Icon compiler, is installed in \icon\bin. A typical version is:

```
iconc %2 %3 %4 %5 %1
icc %1.c \icon\bin\rt.lib
del %1.c
del %1.h
```

The first argument is the name of the Icon program (without the .icn extension). The remaining arguments are provided for options to iconc. Note that the path for rt.lib needs to be changed if this file is moved as described in the preceding section.

For example, to compile hello.icn with no options, use

```
icomp hello
```

while to compile hello.icn with no optimizations, use

```
icomp hello –na
```

Note that options follow the program name. See [1] for other options.

Note: If you are using more than one C compiler, be careful that the C compiler used to compile the output of iconc.exe is the same as the one used to build rt.lib and the other files used by iconc.exe.

6. Testing Icon

The following files are required to test Icon:

compare.exe	comparison utility
icomp.bat	Icon compiler script (compiler tests only)
iconc.exe	Icon compiler (compiler tests only)
icont.exe	Icon translator
iconx.exe	Icon interpreter

These files are located in \icon\bin after building Icon. They need to be available via PATH for the procedures described below. Note that icomp.bat may need to be edited as described in the preceding section.

A suite of test programs is provided in \icon\tests. The expected output of the test programs is in \icon\tests\stand; \icon\tests\local is provided for local output. The ‘‘standard’’ output was produced by Intel Code Builder. *Note:* The tests provided are extensive, but they are not exhaustive.

The directory tests contains several batch scripts for testing Icon. The ones for testing the interpreter are:

intrcoex.bat	co-expressions (if supported)
intrlarg.bat	large-integer arithmetic (if supported)
intrmain.bat	other features

There are corresponding scripts for testing the compiler:

compcoex.bat	co-expressions (if supported)
complarg.bat	large-integer arithmetic (if supported)
compmain.bat	other features

These scripts report differences between ''standard'' and local output. Due to compiler or configuration differences, local output will differ in some cases from the output in stand. Test output from 16- and 32-bit versions of Icon also may differ because of different constants used in dynamic hashing and different sizes for storage regions. The string representation of floating-point numbers also various among different C libraries. Also, tests may fail because of inadequate RAM, in which cases the differences shown may be large.

In the case of differences that appear to be abnormal, look at the corresponding .icn file for a possible explanation.

Note: Running all the interpreter tests takes a long time. Running all the compiler tests takes a *very* long time. If you are building Icon using a compiler for which Icon has been built before, you may wish to limit your testing to a few programs. The scripts compsamp.bat and intrsamp.bat are provided for this purpose.

7. Configuring Icon for a Non-Supported C Compiler

If you want to build Icon under a C compiler for which there presently is no configuration, set up a configuration directory for it and copy files from a configuration that is similar to your C compiler. See [5] for information on modifying the new configuration to suit your C compiler.

If you are successful in building Icon with the new compiler, please send the configuration files and any changed source files to the Icon Project as described in Section 10 so that they can be incorporated in future releases.

8. The Implementation Book

If you are interested in the larger view of the implementation of Icon, or if you are interested in modifying or extending Icon, you may want to acquire the book on the implementation [6]. This book, which can be purchased from the Icon Project, concentrates on the run-time system and covers data structures, the virtual machine, the interpreter, the implementation of generators, and storage management.

The implementation book corresponds to Version 6 of the Icon source code. There have been several changes in the source code between Version 6 and the present version. Supplementary documentation describing these changes is available free of charge from the Icon Project. Ask for [7-9].

9. Trouble Reports and Feedback

If you run into problems, contact the Icon Project:

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)

We cannot guarantee to solve your problems, but we will try. We also may be able to place you in contact with other persons who are compiling Icon and who may have similar problems.

Please also let us know of any suggestions for improvements to the compilation process and its documentation.

Acknowledgements

Many persons have been involved in the implementation of Icon. Clint Jeffery and Gregg Townsend collaborated with the author on Version 9.0.

References

1. R. E. Griswold, *Version 9.0 of the Icon Compiler*, The Univ. of Arizona Icon Project Document IPD237, 1994.
2. R. E. Griswold, *Building Version 9.0 of Icon for OS/2*, The Univ. of Arizona Icon Project Document IPD260, 1994.
3. K. Walker, *The Run-Time Implementation Language for Icon*, The Univ. of Arizona Icon Project Document IPD261, 1994.
4. R. E. Griswold, *Version 9.0 of Icon for MS-DOS*, The Univ. of Arizona Icon Project Document IPD247, 1994.
5. R. E. Griswold, C. L. Jeffery and G. M. Townsend, *Configuring the Source Code for Version 9.0 of Icon*, The Univ. of Arizona Icon Project Document IPD238, 1994.
6. R. E. Griswold and M. T. Griswold, *The Implementation of the Icon Programming Language*, Princeton University Press, 1986.
7. R. E. Griswold, *Supplementary Information for the Implementation of Version 8 of Icon*, The Univ. of Arizona Icon Project Document IPD112, 1992.
8. R. E. Griswold, *Supplementary Information for the Implementation of Version 9.0 of Icon*, The Univ. of Arizona Icon Project Document IPD239, 1994.
9. K. Walker, *The Run-Time Implementation Language for Version 8.7 of Icon*, The Univ. of Arizona Tech. Rep. 92-18. July 1992.