Installing Version 9 of Icon on UNIX Platforms

Gregg M. Townsend, Ralph E. Griswold, and Clinton L. Jeffery

Department of Computer Science
The University of Arizona
Tucson, Arizona

IPD243c
November 29, 1996
http://www.cs.arizona.edu/icon/docs/ipd243.html

---

1. Introduction

Version 9 [1] is the current version of Icon, superseding Version 8. Version 9 contains new features and major changes to the implementation. This report provides the information necessary to install Version 9 of Icon on computers running UNIX.

The implementation of Icon is designed so that it can be installed, largely automatically, on a variety of UNIX platforms. This is accomplished by configuration information that tailors the installation to specific platforms.

The distribution contains configuration information for many UNIX platforms. These are listed in the appendix. Some of these originated under earlier versions of Icon. The platforms marked with an asterisk in the appendix have been tested under Version 9. Installation on a tested platform should be routine, although minor configuration adjustments may be necessary for local conditions.

If there is configuration information for your platform, you may be able to install Icon without modification, but if problems show up, you may have to modify configuration files [2]. In some cases, there may be partial configuration information. If the configuration information for your platform is partial or lacking altogether, you still may be able to install Version 9 of Icon by providing the information yourself, using other configurations as guides.

If your platform is not listed in the appendix, it may have been added since this report was written. See Section 2 for information on how to check for a configuration for a specific platform.

2. The Installation Process

There are only a few steps needed to install Icon proper. In addition to the Icon translator and interpreter, there are three optional components that you can install: a compiler [3], a variant translator system [4], and a program library [5]. You may want to review the technical reports describing these optional components before beginning the installation. In any event, the installation of optional components can be done separately after Icon itself is installed.

There are `Makefile` entries for most steps. Those steps are marked by asterisks. Steps that are optional are enclosed in brackets:

```
1.    Decide where to unload Icon.
```

```
2.    Unload the Icon hierarchy at the selected place.
3*    Check the status of the configuration for your system.
4*    Configure the source code for your system.
5*    Compile Icon.
6*    Run simple tests.
[7*]  Run extensive tests.
[8*]  Run benchmarks.
[9.]  Install Icon at the desired place.
```

Step 1: Deciding Where to Unload Icon

You can build Icon at any place you wish. The executable binaries can be moved to another place later.

In the balance of this report, relative paths and the location of files are given with respect to the location at which the Icon hierarchy is unloaded. For example, a reference to `make` is with respect to the `Makefile` at the top level of this hierarchy.

Step 2: Unloading the Files

The distribution consists of a hierarchy, which is rooted in `"."`. Icon is distributed in a variety of formats. It requires about 20 MB of disk space when unloaded. The amount of space it takes to build Icon depends on the platform, what components are built, and whether intermediate files are deleted between building components.

If the root of the Icon hierarchy is icon, the resulting hierarchy should look like this after the distribution files are unloaded:

```
          |-bin------              executable binaries and support files
          |
          |-config---|-unix------  UNIX configuration directories
          |
          |-docs-----              documents
          |
          |-ipl------              Icon program library
          |
          |          |-common----  common source
          |          |-h---------  header files
          |          |-iconc-----  Icon compiler source
|-icon----|-src------|-icont-----  Icon translator source
          |          |-preproc---  preprocessor source
          |          |-rtt-------  run-time translator source
          |          |-runtime---  run-time source
          |          |-vtran-----  variant translator source
          |          |-xpm-------  XPM support
          |
          |          |-bench-----  benchmarks
          |          |-calling---  calling C functions
          |          |-general---  general tests
          |-tests----|-graphics--  graphics tests
                     |-samples---  sample programs
                     |-vtran-----  variant translator tests
```

There are additional subdirectories that are not shown above.

Step 3: Checking the Status of the Configuration for Your Platform

Check the status of the configuration for your platform before attempting an installation; it may contain essential information. This can be done by

```
make Status name=name
```

where name is one of those given in the table in the appendix at the end of this report. For example,

```
make Status name=sun4 solaris
```

lists the status of the configuration for a Sun 4 workstation running Solaris 2.x.

In many cases, the status information was provided by the person who first installed Icon on the platform in question. The information may be obsolete and possibly inaccurate; use it as a guide only.

There are some configurations for which not all features of Icon are implemented. If the status information shows this for your platform, proceed with the installation, but you may wish to implement the missing features later. See Reference 2 for this.

Step 4: Configuring Icon for Your Platform

Configuring Icon creates several files for general use. Before starting the configuration, be sure your umask is set so that these files will be accessible.

There are two configuration possibilities: with or without graphics facilities.

To configure Icon without graphics facilities, do

```
make Configure name=name
```

where `name` is the name of your platform as described above. For example,

```
make Configure name=sun4_solaris
```

configures Version 9 of Icon for a Sun 4 Workstation, but without graphics facilities.

To configure Icon with the X Window System graphics facilities, use `X-Configure` instead of `Configure`, as in

```
make X-Configure name=sun4_solaris
```

Note: On some platforms, error exit codes from installation processes may be intercepted by `make` and result in warning messages. These messages can be safely ignored.

If you first configure without graphics facilities and later decide to add them, you will need to re-install Icon starting with this step.

If errors occur because the X include files or libraries are not found where they are expected, modify the appropriate files in the subdirectory of `config/unix` (see Reference 2) and restart from the make `X-Configure` step.

Step 5: Building the Icon Interpreter

Next, compile the Icon interpreter by

```
make Icon
```

There may be warning messages on some platforms, but there should be no fatal errors.

Step 6: Performing Simple Tests

If Icon compiles without apparent difficulty, a few simple tests usually are sufficient to confirm that Icon is running properly. The following does the job:

```
make Samples
```

This test compares local program output with the expected output. There should be no differences. If there are no differences, you presumably have a running installation of Icon.

Step 7: Extensive Testing

If you want to run more extensive tests, do

```
make Test
```

Some differences are to be expected, since tests include date, time, local host information, and platform-specific formats for floating-point numbers. In addition to `Test` there are some individual tests of optional features. See the main `Makefile` for more information about the tests.

To test Icon's graphic facilities, use `gpxtest.icn` in `test/graphics`. It should build and run without error, producing a window similar to the GIF image `gpxtest.gif` in the same area.

Step 8: Benchmarking

Programs are provided for benchmarking Version 9 of Icon. To perform the benchmarks, do

```
make Benchmark
```

See also the other material in the subdirectory `tests/bench`. It contains a form that you can use to record your benchmarks with the Icon Project (see Section 9).

Step 9: Installing Icon

The files needed to run Icon are placed in `bin` in the Icon hierarchy as the result of building the Icon interpreter:

```
icont Icon translator
iconx Icon interpreter
```

Some other files related to installing Icon and the optional components mentioned earlier also are placed in `bin`. The executable files needed to run Icon -- `icont` and `iconx` -- can be copied or moved to any desired place, and they need not be in the same directory.

Since `icont` must know the location of `iconx`, it is necessary to patch `icont` if `iconx` is moved. The program `patchstr`, also installed in `bin`, is provided for this purpose. It is used as follows:

```
patchstr icont-location iconx-location
```

For example, if `icont` is moved to `/usr/local/icont` and `iconx` is moved to `/usr/local/icon/iconx`, the patching step is

```
patchstr /usr/local/icont /usr/local/icon/iconx
```

Patching can be repeated if necessary. The patch value can be checked by using `patchstr` without a second argument, as in

```
patchstr /usr/local/icont
```

which prints the path to `iconx` in `/usr/local/icont`.

3. Installing the Compiler

In addition to the interpreter, whose installation is described above, there is a compiler. The interpreter gets a program into execution quickly and is recommended for program development, debugging, and most production situations. The compiler produces code that executes somewhat faster than interpreted code (a factor of 2 or 3 is typical), but the compiler requires a large amount of resources and is very slow in producing executable code. The compiler is recommended only for small programs where execution speed is the paramount concern.

The interpreter and compiler are independent of each other and can be built or rebuilt separately. You can skip this section if you do not need the compiler.

Installing the compiler is very similar to installing the interpreter. Steps 1 through 4 in Section 2 apply to both the interpreter and compiler and need be done only once.

For subsequent steps, there are `Makefile` entries that are the same as for the combined installation, but with the suffix `-iconc` to distinguish the compiler. The steps to build the compiler are:

```
make Icon-iconc
make Samples-iconc
make Test-iconc
make Benchmark-iconc
```

Note: When testing the Icon compiler in conjunction with some C compilers, it may be necessary to remove the options `-p -w` for suppressing warning messages that appear in `icon/tests/general/Makefile`. The file `iconc` needed to run the Icon compiler is placed in `bin` in the Icon hierarchy as the result of building Icon. Files needed by `iconc` also are placed in `bin`:

```
dlrgint.o   stubs for large integer arithmetic
libXpm.a    XPM library if configured for graphics
rt.a        compiler library
rt.db       compiler database
rt.h        include file
```

The executable file `iconc` can be moved to any place. Similarly, the files needed by `iconc` can be moved to another directory. There is a `Makefile` entry for doing this:

```
make CopyLib Target=directory
```

where directory is the directory in which the files needed by `iconc` are to be placed.

Since `iconc` must know the location of the files it uses, it is necessary to patch `iconc` if the files it needs are moved:

```
patchstr iconc-location directory/
```

where `iconc-location` is where `iconc` is located and `directory` is where the files that `iconc` needs are located. For example, if `iconc` is moved to `/usr/local/iconc` and the files needed by `iconc` are placed in the directory `/usr/local/icon/iconc.lib`, the patching step is

```
patchstr /usr/local/iconc /usr/local/icon/iconc.lib/
```

Note that a full path should be used for the directory that contains the files `iconc` needs and that this path must be followed by a terminating slash. The patching of `iconc` can be repeated if necessary.

The path used by `iconc` can be checked by using `patchstr` without a second argument, as in

```
patchstr /usr/local/iconc
```

4. Variant Translators

The variant translator system facilitates the construction of preprocessors for variants of the Icon programming language.

The variant translator system requires a version of *yacc(1)* with large regions. You may have to tailor your version of *yacc(1)* for this. If there is a problem, it will show up during testing.

A script, `icon_vt`, for creating variant translators, is placed in `bin` during the configuration step described earlier. There is no separate step for building the variant translator system.

For testing, do

```
make Test-vtran
```

There may be warning messages during compilation, but there should be no fatal errors.

5. Icon Program Library

The Icon program library contains a variety of programs and procedures. This library not only is useful in its own right, but it provides numerous examples of programming techniques that may be helpful to novice Icon programmers. While this library is not strictly necessary for using Icon, most sites install it, and it is required for all but the most trivial graphics programs.

In addition to the library proper, the directory `ipl/idol` contains an object-oriented version of Icon written in Icon. Go to that directory for more information.

The Icon program library can be used with both the interpreter and the compiler. However, its use under the compiler requires command-line options in some programs to enable features that are not enabled by default when using the compiler. Because of this problem, the installation of the the Icon program library is not supported for `iconc`.

To build the Icon program library, do

```
make Ipl
```

This puts compiled programs in `ipl/icode` and translated procedures in `ipl/ucode`.

To test the library, do

```
make Test-ipl
```

No differences should show.

You can copy the executable programs in `ipl/icode` and the translated procedures in `ipl/ucode` to other places to make them more accessible, although they can be used from any location that is readable by the user.

6. Installing Documentation

The directory docs contains manual pages:

```
icon.1     Icon compiler and interpreter
icon_vt.1  Icon variant translator
```

You may wish to copy these manual pages to a standard location for such documentation. If you are replacing an earlier version of Icon, you should delete the obsolete manual pages, `icont.1`, `iconc.1`, and `icon_pi.1`.

The `docs` directory also contains PostScript files for technical reports related to Version 9 of Icon, including those listed under **References**.

7. Cleaning Up

You can remove object files and test results by

```
make Clean
```

If you copied components of Icon to other places, you can delete the copies left in the Icon hierarchy.

You also can remove source files, but think twice about this, since source files maybe useful to persons studying or modifying the implementation. In addition, you can remove files related to the option components of the Icon system that you do not need. If you are tight on space, you may wish to remove documents as well.

8. Communicating with the Icon Project

If you run into problems with the installation of Version 9 of Icon, contact the Icon Project:

Icon Project
Department of Computer Science
The University of Arizona
P.O. Box 210077
Tucson, AZ 85721-0077
U.S.A.

(520) 621-6613 (voice)
(520) 621-4246 (fax)

icon-project@cs.arizona.edu

Please also let us know if you have any suggestions for improvements to the installation process or corrections or refinements to configuration information.

Acknowledgement

Cliff Hathaway assisted in the testing and distribution of Version 9 of Icon for UNIX platforms.

References

1. R. E. Griswold, C. L. Jeffery and G. M. Townsend, *Version 9.3 of the Icon Programming Language* , The Univ. of Arizona Icon Project Document IPD278, 1996.

2. G. M. Townsend, R. E. Griswold and C. L. Jeffery, *Configuring the Source Code for Version 9 of Icon* , The Univ. of Arizona Icon Project Document IPD238, 1995.

3. R. E. Griswold,*Version 9 of the Icon Compiler* , The Univ. of Arizona Icon Project Document IPD237, 1995.

4. R. E. Griswold, *Variant Translators for Version 9 of Icon* , The Univ. of Arizona Icon Project Document IPD245, 1994.

5. R. E. Griswold and G. M. Townsend, *The Icon Program Library; Version 9.3* , The Univ. of Arizona Icon Project Document IPD279, 1996.

Appendix -- UNIX Icon Configurations

Configuration information for the platforms listed below is provided in Version 9 of Icon. Asterisks identify configurations that have been tested under Version 9, although some have documented problems.

computer	UNIX system	name
Amdahl	UTS	amdahl_uts
Apollo Workstation	BSD	domain_bsd
Astronautics ZS-1	UNIX	zs1
AT&T 3B1 (UNIX PC)	System III	unixpc
AT&T 3B2	System V	att3b_2
AT&T 3B5	System V	att3b_5
AT&T 3B15	System V	att3b_15
AT&T 3B20	System V	att3b_20
AT&T 3B4000	System V	att3b_4000
AT&T 6386	System V	att6386
CDC Cyber	NOS/VE	cdc_vxve
Celerity	4.2BSD	celerity_bsd
Codata 3400	Unisis	codata
Convergent MegaFrame	CTIX	mega
Convex C240	BSD	convex
Cray-2	UNICOS	cray2
*DEC Alpha	OSF/1 Version 3.x	dec_osf
DEC MIPS	Ultrix	decstation
DG AViiON	System V	aviion
DIAB	D-NIX	diab_dnix
Elxsi-6400	BSD	elxsi_bsd
Encore	UMAX	multimax_bsd
Gould Powernode	UTX	gould_pn
HP 9000/330	HP-UX	hp9000_s300
HP 9000/500	HP-UX	hp9000_s500
*HP RISC	HP-UX	hp_risc
IBM 370	AIX	ibm370_aix
IBM PS/2	AIX	ps2_aix
IBM RS6000 Workstation	AIX	rs6000_aix
IBM RT Workstation	ACIS	rtpc_acis
IBM RT Workstation	AIX	rtpc_aix
Intel 286	XENIX 286	i286_xenix
Intel 386	BSD/OS 2.0	i386_bsdos
Intel 386	FreeBSD	i386_freebsd
Intel 386	Linux	i386_linux
Intel 386	Linux	ix86_linux_elf
Intel 386	Solaris	i386_solaris
Intel 386	System V	i386_sysv
Intel 386	System V/GNU C	i386_sysv_gcc
Intel 386	System V, Release 4	i386_svr4
Intel 386	XENIX 386	i386_xenix
Intel 386	XENIX 386/GNU C	i386_xenix_gcc
Intel 486	FreeBSD	i486_freebsd_gcc
Intergraph Clipper	System V	clix
Macintosh	AU/X	mac_aux
Masscomp 5500	System V	masscomp
Microport V/AT	System V	microport
MIPS/r3000	System V	mips

Motorola 8000/400	System V	mot_8000
Multiflow Trace	UNIX	trace
NeXT	Mach	next
Plexus P60	System V	plexus
Pyramid 90x	4.2BSD	pyramid_bsd
Ridge 32	ROS	ridge
Sequent Balance 8000	Dynix	balance_dynix2
Sequent Symmetry	Dynix	symmetry
Siemens MX500	SINIX	mx_sinix
*SGI 4D	Irix	iris4d
Stride 460	UniStride	stride
Sun 2 Workstation	SunOS	sun2
Sun 3 Workstation	SunOS	sun3
Sun 3 with 68881	SunOS	sun3_68881
Sun 386i	SunOS	sun386i
*Sun 4 Workstation	SunOS 4.1	sun4
Sun 4 Workstation	SunOS 4.1/GNU C	sun4_gcc
Sun 4 Workstation	SunOS 4.1/Open Windows	sun4_openwin
Sun 4 Workstation	SunOS 4.1/Code Center	sun4_saberc
*Sun 4 Workstation	Solaris 2.x/SunPro C	sun4_solaris
Sun 4 Workstation	Solaris 2.x/Centerline C	sun4_solar_clc
Sun 4 Workstation	Solaris 2.x/GNU C	sun4_solar_gcc
Unisys 7000/40	4.3BSD	tahoe_bsd
VAX-11	4.1BSD	vax_41_bsd
VAX-11	4.2BSD and 4.3BSD	vax_bsd
VAX-11	System V	vax_sysv
VAX-11	Ultrix	vax_ultrix
VAX-11	9th Edition	vax_v9

Icon home page