Installation Guide for Version 6 of Icon on UNIX Systems*

Ralph E. Griswold

TR 86-11e

May 7, 1986; Last Revised February 9, 1987

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grant DCR-8502015.

Installation Guide for Version 6 of Icon on UNIX Systems

1. Introduction

This report provides the information necessary to install Version 6 of Icon [1] on computers running UNIX. For other operating systems, see [2].

The implementation of Version 6 of Icon is designed so that it can be installed, largely automatically, on a variety of computers running different versions of UNIX. This is accomplished by including, in it, configuration information that is needed to tailor the installation to specific computers and versions of UNIX. As of the date given on the cover of this report, the following configurations were included in the distribution:

computer	*UNIX system*	*name*
Amdahl 580	System V	amdahl_sysv
Amdahl 580	UTS	amdahl_uts
AT&T 3B2/5/15	System V	att3b5
AT&T 3B20	System V	att3b20
Codata 3400	Unisis	codata
Gould Powernode	UTX	gould_pn
HP 9000	HP-UX	hp9000
IBM PC/XT/AT	PC/IX	pc_pcix
IBM XT/AT	XENIX (large memory model)	pc_xenix_lmm
IBM XT/AT	XENIX (small memory model)	pc_xenix_smm
IBM RT PC	ACIS	rtpc_acis
IBM RT PC	AIX	rtpc_aix
Masscomp 5500	System V	masscomp
Plexus P60	System V	plexus
PDP-11 (separate I & D spaces)	Berkeley 2.9bsd	pdp11_bsd
PDP-11 (separate I & D spaces)	Version 7	pdp11_v7
Pyramid 90x	Berkeley 4.2bsd	pyramid_bsd
Ridge 32	ROS	ridge
Sun Workstation	UNIX 4.2	sun
UNIX-PC/3B1	System III	unixpc
VAX-11	Berkeley 4.1bsd	vax_4.1bsd
VAX-11	Berkeley 4.2bsd and 4.3bsd	vax_bsd
VAX-11	System V	vax_sysv
VAX-11	9th Edition	vax_v9

These systems are referred to as "supported" in this report.

If your system is included in this list, the installation of Version 6 of Icon should be as simple as issuing a few make commands. If your system is not in this list, it may have been added since this report was written. See Section 2.2 for information on how to get a current list of configurations and their statuses. In some cases, there may be partial configuration information. If the configuration information for your system is partial or lacking altogether, you still may be able to install Version 6 of Icon by providing the information yourself; see Section 8.

2. The Installation Process

2.1 Unloading Files

The standard location for all files, including executable binaries, is in the directory /usr/icon/v6[†]. If you want source files or executable binary files at other locations, you must edit a file before proceeding. Think twice about this. Executable binary files for Icon are referenced by full path names; they cannot be moved, so it is important to select the correct paths at the start. See Appendix A if you want to change the location of the files.

The Icon distribution is rooted in v6. Unload the distribution files into the installation directory (the standard installation directory is /usr/icon). See Appendix B, if necessary, for information on unloading the distribution files. See Appendix C for a listing of the distribution hierarchy.

2.2 Building Icon

In the description that follows, paths are relative to v6. For example, Makefile refers to the Makefile in the top level of the distribution hierarchy.

There are four phases of the installation: setup, compilation, installation, and testing. All are done using Makefile in v6.

Before proceeding, check the status of the system you plan to install. If your system is one listed in the table in Section 1, do

```
make Status name=name
```

in v6, where *name* is one of those given in the table. For example,

```
make Status name=sun
```

gives the status of Version 6 on the Sun Workstation.

If your system is not listed in the table, do

```
ls setup
```

which lists all the configuration directories (ignore Makefile, common, and generic). If you find your system listed, proceed as above. If there is no configuration directory for your system, or if there is, but the status information indicates it is not sufficiently complete for an installation, go to Section 8, which tells how to provide configuration information for a new system.

There are some supported systems for which not all features of Version 6 are implemented. If the status information shows this for your system, proceed with the installation, but you may wish to implement the missing features later. For this, see Section 8 after completing the basic installation.

Assuming enough configuration information exists to install Version 6 on your system, start the installation with

```
make Setup name=name
```

where *name* is the name of your system as described above. For example,

```
make Setup name=vax_bsd
```

configures Version 6 of Icon for a VAX running Berkeley 4.*n*bsd.

Compilation and installation are done by

```
make Icon Install
```

This takes a while. There may be warning messages on some systems, but there should be no fatal errors.

For systems for which the configuration information is well-established, a few simple tests are sufficient to check that Icon is running properly. The following does the job:

[†]For HP–UX, the standard location is /users/icon/v6.

```
make Samples
```

This test compares local program output with the expected output. There should be no differences. If there are not, you presumably have a running Version 6 Icon.

You may wish to copy the manual page icont.1 in v6/docs to your local public manual area. That's all there is to the installation, although you may wish to install the optional components of Icon described in the next two sections.

3. Personalized Interpreters

Version 6 contains a ''personalized interpreter'' facility [3] that allows an individual to modify and extend a private copy of Icon's run-time system. To install this optional component of Icon,

```
make PI
```

For testing, do

```
make Test-pi
```

Expect some differences in this test, since one program checks local environment variables and the output of another depends on the local memory configuration.

You may want to copy the shell script icon_pi, which makes personalized interpreters, from v6 to a public area for general use. You also may want to copy the manual page icon_pi.1 in v6/docs to the public manual area.

4. Variant Translators

Version 6 contains a ''variant translator'' facility [4] that facilitates the construction of preprocessors for variants of the Icon programming language. To install this optional component of Icon,

```
make VT
```

For testing, do

```
make Test-vt
```

You may want to copy the shell script icon_vt, which makes variant translator systems, from v6 to a public area for general use. You also may want to copy the manual page icon_vt.1 in v6/docs the the public manual area.

5. Icon Program Library

The Icon program library [5] contains a variety of programs and procedures. To install this optional component of the Version 6 distribution,

```
make lpl
```

This puts compiled programs in v6/ipl/progs and translated procedures in v6/ipl/procs. To test the library,

```
make Test-ipl
```

You can copy the programs in v6/ipl/progs and the translated procedures in v6/ipl/procs to other locations if you want.

6. Cleaning Up

After Icon and any optional components have be installed, you can remove non-source files and test results by

```
make Clean
```

You also can remove source files, but think twice about this, since source files may be useful to persons using personalized interpreters.

7. User Impacts

The language features of Version 6 are very similar to those of preceding Versions 5.9 and 5.10. Therefore, most user programs that run properly under versions 5.9 and 5.10 should run properly under Version 6. However, users should be encouraged to read the Version 6 description [1]. There are two possible sources of operational problems: version checking and link path specifications.

Prior to Version 6, there was no check that ucode files produced by the Icon translator and executable icode files produced by the Icon linker were compatible with the Icon run-time system. Incompatible files simply caused mysterious program malfunction. Version 6 produces version-numbered files and checks compatibility. An attempt to use a file produced by an earlier version results in a error message from Version 6 (ucode file has no version identification).

Since executable files contain the full path name of the run-time system (iconx), proper installation should avoid problems with version numbers in icode files. Ucode files, which are typically used in libraries included by the link declaration, are more troublesome. Incompatible versions may be confusing, since the source of the problem may be hidden. Users should be advised to re-translate all ucode files when Version 6 is installed. Similarly, if an earlier version of the Icon program library is in use, it should be replaced by the Version 6 library, built under Version 6 as described in Section 5. (Some earlier Icon program library material has been deleted in Version 6; if such material is in use, it can be added to the Version 6 library.)

The syntax of the IPATH environment variable, which is used by the linker to search for files given in link declarations, has changed in Version 6. Previously the separator was a colon; in Version 6 it is a blank. Problems with IPATH usually are indicated by the linker error message Can't resolve reference to file ....

8. Configuring Version 6 for a New System

Version 6 of Icon assumes that C *int*s are either 16 or 32 bits long. If your system violates this assumption, don't try to go on — but check back with us, since we are may be able to provide some advice on how to proceed.

To install Icon on a system that is not supported in the distribution, you must create a directory to hold files containing configuration information. In the description that follows, this directory is referred to as your configuration directory.

First you need to select a name for your system. The name should consist of a mnemonic for the computer, which may be followed by an underscore and a mnemonic for the operating system, in case there may be more than one operating system for the computer. Examples are vax_bsd and vax_sysv. You may want to append an additional underscore and a qualification if there is more than more likely implementation for a given computer and operating system. An example is pc_xenix_smm for the small memory model implementation of Icon on XENIX.

To build and initialize a new configuration directory,

```
make System name=name
```

where *name* is the name of your system.

Now

```
cd v6/setup/name
```

where you need to edit some files. The files that may need editing are

paths.h	paths for executable binary files
header.hdr	sizing for a bootstrap header file
icont.hdr	flags for command processor Makefile
iconx.hdr	flags and other definitions for the run–time system Makefile
link.hdr	flags for the linker Makefile
tran.hdr	flags for the translator Makefile
pi.hdr	flags for the personalized interpreter Makefile
vt.hdr	flags for the variant translator Makefile
config.trl	other configuration definitions
rswitch.c	co–expression context switch
rover.c	arithmetic overflow checks
Ranlib	library randomizer for personalized interpreters

8.1 paths.h

If /usr/icon/v6 is acceptable as a location for all Icon files, you don't have to change paths.h. Otherwise, handle the file paths.h as described in Appendix A.

8.2 header.hdr

The file header.hdr contains a definition for MaxHdr, which determines the amount of space that is reserved for iconx.hdr, the bootstrapping program that gets Icon programs into execution. You cannot determine the most appropriate value of MaxHdr until after Icon has been compiled. An overly large value just wastes file space in compiled Icon programs, but a value that is not large enough prevents compiled Icon programs from executing.

The value provided, 4096, is more than enough for most systems and you can wait to change it until later. Remember, however, that if Icon programs fail to execute, this value may be too small.

After all else has been done and Icon has been compiled and tested, come back to this file and change the value of MaxHdr to the size of v6/src/icont/iconx.hdr. You may need to round the size up on some systems. In any event, make MaxHdr a hundred or so larger than the size of iconx.hdr to allow for a somewhat larger size that may be needed for personalized interpreters.

On some systems, particularly UNIX emulators, many routines may be included by the loader regardless of need. In this case, the size of iconx.hdr may be impractically large. If this is the case on your system, the header file can be eliminated altogether by adding

```
#define NoHeader
```

to config.trl (see Section 8.4). The effect of this definition is to render Icon programs non-executable. Instead, they must be run by using the –x option after the program name when icont is used, as in

```
icont prog.icn –x
```

Such a program also can be run as an argument of iconx, as in

```
iconx prog
```

where prog is the result of translating and linking prog.icn as in the previous example.

If NoHeader is defined, the value of MaxHdr is irrelevant.

8.3 Makefile Headers

The files icont.hdr, iconx.hdr, link.hdr, and tran.hdr provide headers for Makefiles in the source directories v6/src/icont, v6/src/iconx, and so on. These headers are prepended to the standard bodies for the Makefiles during setup.

Except for iconx.hdr, these headers serve only to specify flags for *cc(1)* and *ld(1)* via CFLAGS and LDFLAGS. If your C optimizer is robust, you may wish to start with

```
CFLAGS= –O
```

in all these headers. In fact, if you are installing Icon on a computer with a small address space, this flag (as well as

others) may be necessary to obtain modules small enough to load. However, if you encounter problems during testing, suspect your optimizer first and try compiling Version 6 without the –O flag.

Other *cc* and *ld* flags vary considerably from system to system. You may want to review your local manual pages for these processors and look at the header files in the other configuration areas.

There are two other definitions in iconx.hdr: RSWITCH and ROVER, which depend on whether the local co-expression context switch and arithmetic overflow checks are written in C or assembly language. The initial values of these definitions are rswitch.c and rover.c, and dummy C routines are provided. To start out, leave these definitions as they are; the default routines can be replaced later. See Sections 8.7 and 8.8.

The file pi.hdr provides a header for the personalized interpreter Makefile, which is in v6/Pimakefile. In addition to the usual *cc* and *ld* flags, you should provide definitions for XCFLAGS and XLDFLAGS that are the same as those for CFLAGS and LDFLAGS in icont.hdr. This assures that the header file in the personalized interpreter is the same size as the one in the regular version of Icon.

The file vt.hdr provides a header for the variant translator Makefile, which is in v6/Vtmake2. It should have the same *cc* and *ld* flags as tran.hdr.

8.4 config.trl

The remaining configuration information is contained in config.trl. The definitions in this file as provided by Setup are for a "vanilla" 32-bit computer. Changes may be needed as follows:

Fork

If your system supports *vfork(2)*, change the definition of Fork from fork to vfork.

HostStr

Change the definition of HostStr, which provides the value of the Icon keyword &host, to some string that adequately identifies your installation. If you want to use a local system routine to provide the host name automatically, read Appendix D.

Hz

If you are running in a 50-hz environment, change the definition of Hz from 60 to 50.

IntSize, LongSize, and PtrSize

Define these constants to be the sizes, in bits, of your C ints, longs, and pointers, respectively. These values normally are 16 or 32.

NoOver

Initially, config.trl contains

```
#define NoCoexpr
#define NoOver
```

Thses definitions disable co-expressions and arithmetic overflow checks, which must be written in assembly language. Leave these definitions in for the first round, although you may want to remove them later (see Sections 8.7 and 8.8).

SysTime

The system include file time.h is in different locations on different systems — either <time.h> or <sys/time.h>. Define SysTime accordingly.

If you give the incorrect location, a fatal error will occur during the compilation of v6/src/iconx/lmisc.c. The use of this definition also depends on your C preprocessor making macro substitutions in #include directives. Most preprocessors do, but if yours does not, edit /v6/src/iconx/lmisc.c and replace SysTime there by the appropriate value. If you have to do this, make a note to come back later and place the definition under the control of conditional compilation as described in Section 8.5.

Names of Indexing Routines

Different versions of UNIX use different names for the routines for locating substrings within strings. Version 6 of Icon uses index and rindex. The other possibilities are strchr and strrchr. If your system uses the latter names, add

```
#define index strchr
#define rindex strrchr
```

to config.trl.

Other Definitions

There are several other configuration details that are needed for a few systems. Most of these can be handled by optionally defined symbols.

If your system needs a specific check for division by floating-point zero, as opposed to relying on a signal, add

```
#define ZeroDivide
```

to your config.trl file.

If your system requires C *doubles* to be aligned at double-word boundaries, add

```
#define Double
```

to your config.trl file.

Most computers have down-growing C stacks, for which stack addresses decrease as values are pushed. If you have an up-growing stack, for which stack addresses increase as values are pushed, add

```
#define UpStack
```

to your config.trl file.

Icon includes its own versions of malloc, calloc, realloc, and free, so that it can manage its storage region without interference from allocation by the operating system. Normally, Icon's malloc and free are loaded instead of the system library routines. If your system insists on loading its own library routines, multiple definitions will occur as a result of the *ld* in /v6/src/iconx.

If multiple definitions occur in iconx, go back and add

```
#define IconAlloc
```

to config.trl. This definition causes Icon's routines to be named differently to avoid collision with the system routine names.

One possible effect of this definition is to interfere with Icon's expansion of its memory region in case the initial values for allocated storage are not large enough to accommodate a program that produces a lot of data. This problem appears in the form of run-time error 303 or 304 and can be circumvented on a case-by-case basis by increasing the initial values for allocated storage by setting environment variables [6].

The C runtime library routine *atof* is used in the Icon linker to convert strings for real literals to corresponding floating-point numbers. If the version of *atof* on your system does not work properly, add

```
#define NoAtof
```

which replaces the use of *atof* by in-line conversion code.

If your C runtime library does not contain *qsort*, add

```
#define IconQsort
```

which causes a version of *qsort* in iconx to be used.

Icon's dynamic storage allocation system uses three contiguous memory regions that it expands if necessary. This method relies on the use of *brk* and *sbrk* and the system treatment of user memory space as one logically contiguous region. This may not work on some systems that treat memory as segmented or do not support *brk* and *sbrk*. On such systems, it may be necessary to add

```
#define FixedRegions
```

The effect of this definition is to assign fixed-sized regions for Icon's use. They may not be shared or expanded and all of available memory may not be used. This option should be used only if necessary.

8.5 Modifications to the Distributed Source Code

The configuration system is designed to avoid modifications to the distributed source code for Version 6. However, on some systems, it may be necessary to modify the source code.

If you need to modify the source code, do it under the control of conditional compilation keyed to the name of your system. Add

```
#define NAME
```

to config.trl, where *NAME* is an all-uppercase name that identifies your system. For example, the config.trl for Sun Workstations contains

```
#define SUN
```

Then use

```
#ifdef NAME
        .
        .
        .
#endif NAME
```

or similar constructions where you need local source-code modifications. For example, this technique can be used to handle the problem that may arise with SysTime, described in Section 8.4. Note that nested #ifdefs may be needed in places where there are several different local modifications.

It is important to be consistent and careful about the use of such conditional compilations; if done properly, your modifications can be backed into the central version of the source code at the University of Arizona and will be in place for you when subsequent versions are released. See Section 9.

8.6 Initial Installation of a New System

Once you have edited the configuration files as described in the previous sections, proceed with the setup, compilation, and installation as described in Section 2. You may need to iterate. If you make a change in a configuration file after a compilation, be sure to perform the setup again; some aspects of the setup are far-reaching and not obvious.

More testing is recommended for a new installation than for one that has been successfully installed elsewhere. As a start, do

```
make Test-icon
```

If that works,

```
make Test-large
```

or

```
make Test-small
```

depending on whether you defined PtrSize to be 32 or 16 earlier. These tests are quite extensive and contain some real grinders; be prepared to wait a while. See v6/tests/Makefile for more information.

There will be some differences between local and standard results in the program check, since it contains site and time-dependent tests. Other minor discrepancies may occur because of differences in the handling of floating-point arithmetic on different systems.

Do *not* run any other tests until you have implemented the co-expression context switch (see Section 8.7) and decided what to do about arithmetic overflow checking (see Section 8.8).

8.7 Co-Expressions

All aspects of co-expression creation and activation are written in C in Version 6 except for a routine, coswitch, that is needed for context switching. This routine requires assembly language, since it must manipulate hardware registers. It either can be written as a C routine with asm directives or as an assembly language routine.

When a new configuration directory is set up, a file rswitch.c is provided with a version of coswitch that results in error termination if an Icon program attempts to activate a co-expression.

Calls to the context switch have the form coswitch(old_cs,new_cs,first), where old_cs is a pointer to an array of words that contain C state information for the current co-expression, new_cs is a pointer to an array of words that hold C state information for a co-expression to be activated, and first is 1 or 0, depending on whether or not the new co-expression has or has not been activated before. The zeroth element of a C state array always contains the hardware stack pointer (*sp*) for that co-expression. The other elements can be used to save any C frame pointers and any other registers your C compiler expects to be preserved across calls.

The default number of elements for saving the C state is 15. This number may be changed by adding

```
#define CStateSize n
```

to config.trl, where *n* is the number of elements needed.

The first thing coswitch does is to save the current pointers and registers in the old_cs array. Then it tests first. If first is zero, coswitch sets *sp* from new_cs[0], clears the C frame pointers, and *calls* interp. If first is not zero, it loads the (previously saved) *sp*, C frame pointers, and registers from new_cs and returns.

Written in C, coswitch has the form:

```
/*
 * coswitch
 */
coswitch(old_cs, new_cs, first)
int *old_cs, *new_cs;
int first;
{
                                    .
                                    .
                                    .
        /* save sp, frame pointers, and other registers in old_cs */
                                    .
                                    .
                                    .
   if (first == 0) {/* this is first activation */
                                    .
                                    .
                                    .
        /* load sp from new_cs[0] and clear frame pointers */
                                    .
                                    .
                                    .
      interp(0, 0);
      syserr("interp() returned in coswitch");
      }
   else {
                                    .
                                    .
                                    .
        /* load sp, frame pointers, and other registers from new_cs */
                                    .
                                    .
                                    .
      }
}
```

Appendix E contains coswitch for the VAX. Other examples are contained in the configuration directories in v6/setup.

If you do not want to implement the context switch, the only effect will be that Icon programs that attempt to activate a co-expression will terminate with an error message. If you chose to implement the context switch, remove the #define NoCoexpr from config.trl and replace rswitch.c in your configuration directory by either a new

rswitch.c or an assembly language file named rswitch.s. The setup process will copy your file to the appropriate place prior to compilation. If you use rswitch.s, change the definition of RSWITCH in iconx.hdr in your configuration area to

```
RSWITCH=rswitch.s
```

If your assembler requires special flags, add an appropriate definition for OFLAGS to iconx.hdr.

To test your context switch,

```
make Test-Icoexpr
```

or

```
make Test-scoexpr
```

depending on whether you defined PtrSize to be 32 or 16 earlier. Ideally, there should be no differences in the comparison of outputs. However, on systems with limited memory space, some tests may terminate prematurely with an indication that the amount of memory needed is inadequate.

If you have trouble with your context switch, the first thing to do is double-check the registers that your C compiler expects to be preserved across calls — different C compilers on the same computer may have different requirements.

Another possible source of problems is built-in stack checking. Co-expressions rely on being able to specify an arbitrary region of memory for the C stack. If your C compiler generates code for stack probes that expects the C stack to be at a specific location, you may need to disable this code or replace it with something more appropriate.

If your system does not allow the C stack to be at an arbitrary place in memory, there is probably little hope of implementing co-expressions.

8.8 Arithmetic Overflow Checks

C does not provide overflow checking for integer addition, subtraction, or multiplication. Icon, on the other hand, is supposed to check for overflow. This usually requires assembly-language code.

The config.trl file provided when a new configuration area is set up provides the definition

```
#define NoOver
```

which causes overflow checking to be bypassed.

If you do not want to implement overflow checking, you need do nothing. If you want to implement overflow checking, remove the definition of NoOver from your config.trl and write routines ckadd, cksub, and ckmul that call runerr(203,0) in the case of overflow. Appendix F contains the overflow checking routine for the VAX. Other examples are contained in the configuration directories in v6/setup.

If you supply overflow checking routines, put them in the file rover.s in your configuration directory. The setup will copy this file to the appropriate place prior to compilation.

To test overflow checking,

```
make Test-over
```

There should be no differences in the comparison of outputs if overflow checking is working properly.

You should also rerun previous tests at this point to make sure that arithmetic still works properly.

8.9 Local Assembly-Language Code

Occasionally there is a need for some assembly-language code in the run-time system beyond what is provided for in rswitch.s and rover.s. The file rlocal.s is provided for such contingencies. It initially is a place holder that is copied from the configuration file to v6/src/iconx and is compiled and linked when the run-time system is built. It can be replaced by any necessary assembly-language routines.

8.10 Personalized Interpreters

The personalized interpreter system uses *ar(1)*. On most UNIX systems, it is necessary to use *ranlib(1)* so that the loader can access the archive. The script Ranlib that is provided when a new configuration directory is initialized contains calls of *ranlib* for this purpose.

Some UNIX systems, notably System V, handle this problem directly in *ar(1)* and do not have *ranlib(1)*. If your system does not use *ranlib(1)*, change Ranlib to an empty script by

```
echo "" >Ranlib
```

in your configuration directory.

Test your personalized interpreter system as described in Section 3. If the test programs fail to execute, suspect the size of v6/tests/pi/piconx.hdr, the personalized interpreter version of iconx.hdr. If it is larger than MaxHdr, something is wrong. Check the file pi.hdr in your configuration directory as described in Section 8.3.

8.11 Status Information

Each configuration directory contains a file named status that describes the state of the configuration. A placeholder is provided when a new configuration directory is set up using make System. When the configuration is complete, edit status appropriately, using the status files in other configuration directories as models.

9. Trouble Reports and Feedback

If you run into problems, contact the Icon Project:

Icon Project
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

(602) 621-6613

icon-project@arizona.edu
{ihnp4,noao,mcnc,utah-cs}!arizona!icon-project

Please also let us know of any suggestions for improvements to the installation process and corrections or refinements to configuration files for supported systems.

If you installed a previously unsupported system, please send a copy of the files in your configuration directory and any files in v6/src that you modified to the Icon Project so that we can back them in to the central version of the source.

10. Porting Icon to Non-UNIX Operating Systems

Although Version 6 of Icon was developed under UNIX, it has been ported to MS-DOS systems on personal computers and the VAX-11 running under VMS. Version 6 is designed to be portable to other operating systems that have production-quality C compilers with libraries that support a minimal UNIX-like environment.

For information on what is involved, contact the Icon Project.

Acknowledgement

Gregg Townsend made a number of helpful suggestions related to the process for installing Version 6.

References

1. Griswold, Ralph E., William H. Mitchell, and Janalee O'Bagy. *Version 6 of Icon*, Technical Report TR 86-10c, Department of Computer Science, The University of Arizona. 1986.

2. Griswold, Ralph E. *Transporting Version 6 of Icon*, Technical Report TR 86-25, Department of Computer Science, The University of Arizona. 1986.

3. Griswold, Ralph E. *Personalized Interpreters for Version 6 of Icon*, Technical Report TR 86-12b, Department of Computer Science, The University of Arizona. 1986.

4. Griswold, Ralph E. and Kenneth Walker. *Building Variant Translators for Version 6 of Icon*, Technical Report TR 86-26, Department of Computer Science, The University of Arizona. 1986.

5. Griswold, Ralph E. *The Icon Program Library; Version 6, Release 1*, Technical Report TR 86-13b, Department of Computer Science, The University of Arizona. 1986.

6. Griswold, Ralph E. *ICONT(1)*, manual page for *UNIX Programmer's Manual*, Department of Computer Science, The University of Arizona. 1986.

Appendix A — Changing the Locations of Version 6 Files

As mentioned in Section 2.1, the distributed files are rooted in v6. The standard location for this hierarchy is /usr/icon/v6[1]. The hierarchy can be placed somewhere else if you desire, but before performing the setup, you must edit a file that specifies where the binary files are to be installed.

The directory v6/setup contains a subdirectory for each supported system. For example, v6/setup/sun contains the configuration information for the Sun Workstation. To get to the configuration information for your system,

```
cd v6/setup/name
```

where *name* is the name of your system.

The file paths.h, as distributed, is the same for all systems and contains

```
#define RootPath        "/usr/icon/v6"
#define IcontPath       "/usr/icon/v6/bin/icont"
#define TranPath        "/usr/icon/v6/bin/itran"
#define LinkPath        "/usr/icon/v6/bin/ilink"
#define IconxPath       "/usr/icon/v6/bin/iconx"
#define HeaderPath      "/usr/icon/v6/bin/iconx.hdr"
```

RootPath gives the location of the v6 hierarchy; it must be the path of the root directory where the distributed files are located.

The five binary files referenced are:

icont	Icon command processor — all the typical user knows about
itran	Icon translator
ilink	Icon linker
iconx	Icon run-time system
iconx.hdr	bootstrap program that gets Icon programs into execution

The command processor icont calls the other programs.

There are two reasons for changing these paths:

1. If v6 is unloaded in an area other than /usr/icon/v6, you probably want the binary files installed in that area instead of /usr/icon/v6.
2. You may want to install some or all of the binary files in a public area.

For example, if you want to unload Icon in /usr/irving/v6 and have the binaries in /usr/local/icon, edit paths.h to be

```
#define RootPath        "/usr/irving/v6"
#define IcontPath       "/usr/usr/local/icon/icont"
#define TranPath        "/usr/usr/local/icon/itran"
#define LinkPath        "/usr/usr/local/icon/ilink"
#define IconxPath       "/usr/usr/local/icon/iconx"
#define HeaderPath      "/usr/usr/local/icon/iconx.hdr"
```

Caution: If you have a previously installed version of Icon, do not put iconx at that place — iconx for Version 6 is incompatible with iconx for previous versions and replacing a previous iconx by Version 6 iconx will invalidate all previously compiled Icon programs.

[1]For HP-UX, the standard location is /users/icon/v6. The location /usr/icon/v6 is used here for convenience.

Appendix B — Unloading the Distribution Files

The Icon distribution files are distributed in a variety of ways. The usual distribution media is magnetic tape, although it is also available on Sun cartridges and on 5-¼″ diskettes in several formats. The directions that follow refer to magnetic tape distribution only.

The Icon system is provided on tape in *tar* or *cpio* format, recorded at 1600 or 6250 bpi as specified when Version 6 is ordered. Tapes are written in *tar* format at 1600 bpi if no specification is given. The format and recording density are marked on the label on the tape.

To unload the tape, do a *cd* to the directory that is to hold the Icon hierarchy (that is, the directory in which v6 is to be created) and mount the tape. The precise *tar* or *cpio* command to unload the distribution tape depends on the local environment. On a VAX running 4.*n*bsd, use the following command for a 1600 bpi *tar* distribution tape:

```
tar x
```

Similarly, on a VAX running System V with a 6250 bpi *cpio* tape, use:

```
cpio -icdB </dev/rmt/0h
```

The c (compatibility) and B (blocked) options are essential.

Data cartridges are functionally equivalent to magnetic tapes, but they are not blocked. For example, on a Sun Workstation with a *cpio* cartridge, use

```
cpio -icd </dev/rst0
```

Appendix C — The Distribution Hierarchy

The main directories in the Version 6 hierarchy are:

/v6		root of the Version 6 hierarchy
	/bin	standard location for binary files
	/book	programs from the Icon book
	/docs	source text for documents
	/ipl	Icon program library
	/pi	personalized interpreter system
	/samples	sample programs
	/setup	setup and configuration files
	/src	source code for the Icon system
	/tests	test programs

A complete directory tree is contained in v6/docs/v6.dtree.

Appendix D — Host Name

The default method for specifying the host name that determines the value of &host is the definition of HostStr in config.trl in the configuration directory. There are three other ways of specifying the host; only one may be used.

WhoHost

On some versions of UNIX, notably Version 7 and 4.1bsd, the file /usr/include/whoami.h contains the host name. If your system has this file and you want to use this name,

```
#define WhoHost
```

GetHost

Some versions of UNIX, notably 4.2bsd and 4.3bsd, provide the host name via the *gethostname(2)* system call. If your system supports this system call and you want to use this name,

```
#define GetHost
```

UtsName

Some versions of UNIX, such as System V, provide the host name via the *uname(2)* system call. If your system supports this call and you want to use this name,

```
#define UtsName
```

Appendix E — A Sample Co-Expression Context Switch

rswitch.c for the VAX under Berkeley UNIX:

```
coswitch(old_cs, new_cs, first)
int *old_cs, *new_cs;
int first;

{
   asm("   movl 4(ap),r0");
   asm("   movl 8(ap),r1");
   asm("   movl sp,0(r0)");
   asm("   movl fp,4(r0)");
   asm("   movl ap,8(r0)");
   asm("   movl r11,16(r0)");
   asm("   movl r10,20(r0)");
   asm("   movl r9,24(r0)");
   asm("   movl r8,28(r0)");
   asm("   movl r7,32(r0)");
   asm("   movl r6,36(r0)");

   if (first == 0)          { /* this is first activation */
      asm("   movl 0(r1),sp");
      asm("   clrl fp");
      asm("   clrl ap");
      interp(0, 0);
      syserr("interp() returned in coswitch");
      }

   else {
      asm(" movl 0(r1),sp");
      asm(" movl 4(r1),fp");
      asm(" movl 8(r1),ap");
      asm(" movl 16(r1),r11");
      asm(" movl 20(r1),r10");
      asm(" movl 24(r1),r9");
      asm(" movl 28(r1),r8");
      asm(" movl 32(r1),r7");
      asm(" movl 36(r1),r6");
      }
}
```

Appendix F — A Sample Arithmetic Overflow Checking Routine

rover.s for the VAX under Berkeley UNIX:

```
_ckadd:     .word     0
            addl3     4(ap),8(ap),r0        # Perform addition
            jvs       oflow                 # Branch if overflow
            ret                             # Return result in r0

_cksub: .word         0
            subl3     8(ap),4(ap),r0        # Perform subtraction
            jvs       oflow                 # Branch if overflow
            ret                             # Return result in r0

_ckmul:     .word     0
            mull3     4(ap),8(ap),r0        # Perform multiplication
            jvs       oflow                 # Branch if overflow
            ret                             # Return result in r0

oflow:                                      # Got overflow on an operation
            pushl   $0
            pushl     $203
            calls     $1,_runerr    # runerr(203,0)
```