

## **Version 9.0 of MPW Icon**

Ralph E. Griswold  
Department of Computer Science, The University of Arizona

Robert J. Alexander

August 28, 1994  
IPD252

### **1. Introduction**

Version 9.0 of MPW Icon (Icon for the Macintosh under MPW) is distributed on an 800K diskette, which includes executable binary files, a few sample programs, and documentation in machine-readable form. Printed documentation is included with diskettes distributed by the Icon Project at the University of Arizona.

MPW Icon runs as part of the *Macintosh Programmer's Workshop* (MPW) integrated environment. The MPW Shell is required to run MPW Icon (available for about \$100 from APDA (Apple Programmers and Developers Association), (800) 282-2732). Neither the Icon translator/linker nor programs written in MPW Icon can run independently of the MPW Shell. Stand-alone Macintosh applications *cannot* be created with MPW Icon. Programs produced with MPW Icon run under the MPW Shell as *tools*.

Consult MPW documentation for Macintosh memory requirements. Icon itself presents no exceptional memory demands beyond those of MPW itself, and most Icon programs will run with minimally-configured MPW. Of course, Icon programs that have exceptional memory requirements will require enough real or virtual memory on the Macintosh, and will require that MPW be configured large enough.

Version 9.0 of MPW Icon was built using MPW 3.3.1, and is known to run properly under MPW 3.3.1. It has not been tested with other versions of MPW as of this writing. Past experience suggests that MPW Icon version 9.0 will run okay on any relatively recent MPW version.

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

### **2. Documentation**

The primary reference for the Icon programming language is the book

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

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

Note that the first edition of this book, published in 1983, describes version 5 of Icon and does not contain information about many of the features of later versions.

A brief overview of Icon is contained in technical report TR 90-6 [1] ([Icon Overview](#) on the distribution diskette). Features that have been added to Icon since the book was written are described in IPD236 [2] ([Icon Version 9.0](#) on the distribution diskette). These technical reports, together with this document ([MPW Icon](#) on the distribution diskette), provide enough information to write and run simple Icon programs. Persons who intend to use Icon extensively, however, will need the book.

As mentioned above, printed documentation is provided with copies of Icon obtained from the Icon Project. Machine-readable documentation is also included on the distribution diskette in the form of text and Microsoft® Word files. Text files can be viewed and printed by the MPW Shell. Word documents contain fonts that are optimized for printing on a LaserWriter. To print them on an ImageWriter, they will have the best quality if the fonts Times 12 and 10, Helvetica 12, and Courier 10 are installed (or double those sizes for *best* quality printing). Much of the style of the documents is expressed through its fonts, so it is best not to tamper with them unless absolutely necessary.

### **3. Installing MPW Icon**

A MPW Shell script is provided to perform the installation of Icon into the MPW system on your Macintosh. The script will guide you through the installation, informing you of the recommended places to store your Icon materials, and will allow you to exercise options if you wish to keep Icon in a place different from the default. The procedure is quite simple, and will go something like this:

A dialog will appear saying:



Followed by the following question:



If you already have a version of MPW Icon installed, select the **Existing icon folder** choice. If this is your first Icon, select **New icon folder**.

The procedure will continue in this fashion, with a few more questions (such as *which folder?*) and words of advice, until the installation is complete.

To start the installation procedure: start **MPW Shell**, insert the **MPWIcon** disk (the distribution disk) into a disk drive and enter the command:

**MPWIcon:Installicon**

The following items will be installed:

In the **Icon** folder (or other folder designated during the installation process):

<u>Tools</u>		
	icont	The Icon translator and linker
	iconx	The Icon interpreter
<u>Scripts</u>		
tool.	IconMPWTool	Converts an icode file to a stand-alone MPW
<u>Other</u>		
	icon.help	MPW-style help files for Icon.

In your **MPW** folder:

**UserStartup•Icon** Performs MPW initialization for Icon.

The installation script will attempt to delete obsolete version 7.0 (or earlier) files if it finds them. Those files are **itran**, **ilink**, **iconm**, **iconx.hdr**, and **bin** (the folder and its contents). If you want to save these files, move them to a diskette or another folder before installation.

The installation can be aborted any time a dialog box is displayed by clicking its Cancel button. Installation can be restarted from the beginning with no adverse consequences.

If you are running a version of MPW older than 3.0, you will have to enter the following line into your UserStartup file (in the MPW folder):

#### Execute UserStartup•Icon

The • character is *option-8*. This requirement is discussed in more detail in section 9 of this document. The additional line is not necessary for MPW version 3.0 and later.

### 4. Running Icon on the Macintosh under MPW — Basic Information

Files containing Icon programs must have the suffix .icn. Such files should be plain text files (without line numbers or other extraneous information). The translator/linker program icont produces an executable icode file. For example, an Icon program in the file prog.icn is translated and linked by

```
icont prog.icn
```

The result is an executable icode file with the name prog. Icode files produced by MPW Icon behave identically to MPW tools, and thus are invoked by entering their file names. The icode file just created can be run by

```
prog
```

Use of the suffix .icn in the icont command is optional. For example, it is sufficient to use

```
icont prog
```

icont will supply the .icn suffix if no suffix is entered.

### 5. Testing the Installation

There are a few programs on the distribution diskette that can be used for testing the installation and getting a feel for running Icon. The programs and data files are in the folder Samples.

#### hello.icn

This program prints the Icon version number, time, and date. Run this test as

```
icont hello  
hello
```

#### kross.icn

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

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

### **meander.icn**

This program prints the “meandering strings” that contain all subsequences of a specified length from a given set of characters. Run this test as

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

If these tests work, your installation is most likely correct — you 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/linker 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. If you are new to Icon, you may wish to skip this section on the first reading but come back to it if you find the need for more control over the translation and execution of Icon programs.

### **6.1 Arguments**

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

```
prog text.dat log.dat
```

runs the icode file `prog`, passing its main procedure a list of two strings `["text.dat", "log.dat"]`. These arguments might be the names of files that `prog.icn` reads from and writes to. For example, the main procedure might begin as follows:

```
procedure main(a)  
  in := open(a[1]) | stop("cannot open input file")  
  out := open(a[2], "w") | stop("cannot open output file")  
  ...
```

### **6.2 The Translator and Linker**

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

```
icont prog1 prog2
```

translates and links the files `prog1.icn` and `prog2.icn` and produces one icode file, `prog1`.

If the **-c** option is given to **icont**, only translation is performed and intermediate *ucode* files with the suffixes **.u1** and **.u2** are kept. For example,

```
icont -c prog1
```

leaves **prog1.u1** and **prog1.u2**, instead of linking them to produce **prog1**. (The *ucode* files are deleted unless the **-c** option is used.) These *ucode* files can be used in a subsequent **icont** command by using the **.u1** name. This avoids having to translate the **.icn** file again. For example,

```
icont prog2 prog1.u1
```

translates **prog2.icn** and links the result with the *ucode* files from a previous translation of **prog1.icn**. Note that only the **.u1** name is given. The suffix can be abbreviated to **.u**, as in

```
icont prog2 prog1.u
```

*Ucode* files also can be added to a program when it is linked by using the **link** declaration in an Icon source program as described in [2].

The informative messages from the translator and linker can be suppressed by using the **-S** option. Normally, both informative messages and error messages are sent to standard error output.

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

```
icont -o probe prog
```

produces the *icode* file named **probe** rather than **prog**.

Icon source programs may be read from standard input. The argument **-** signifies the use of standard input as a source file. In this case, the *ucode* files are named **stdin.u1** and **stdin.u2** and the *icode* file is named **stdin**.

Normally, **&trace** has an initial value of 0. The **-t** option to **icont** causes **&trace** to have an initial value of -1 when the program is executed. For example,

```
icont -t prog
```

causes tracing to occur when **prog** is run.

The option **-u** to **icont** causes warning messages to be issued for undeclared identifiers in the program. The warnings are issued during the linking phase.

### 6.3 Environment Variables

When an Icon program is executed, several environment variables (exported shell variables) are examined to determine execution parameters. The values assigned to these variables should be numbers.

Environment variables are particularly useful in adjusting Icon's storage requirements. This may be necessary if your computer does not have enough 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 be set to affect Icon's execution parameters. Their default values are listed in parentheses after the environment variable name.:

TRACE (0).	This variable initializes the value of <b>&amp;trace</b> . If this variable has a value, it overrides the translation-time <b>-t</b> option.
NOERRBUF (undefined).	If this variable is set, <b>&amp;errout</b> is not buffered.
STRSIZE (65000).	This variable determines the initial size, in bytes, of the region in which strings are stored.
BLOCKSIZE (65000).	This variable determines the initial size, in bytes, of the region in which Icon allocates lists, tables, and other objects.
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.
STATSIZE (20480).	This variable determines the size, in bytes, of the static region in which co-expression blocks are allocated.
STATINCR (calculated).	This variable determines the size of the increment used when the static region is expanded. The default increment is one-fourth of the initial size of the static region.

Region sizes expand if more space is needed, but they never shrink.

## 7. Features of Icon for the Macintosh under MPW

MPW Icon supports the features of Version 9.0 of Icon, with the following exceptions and additions:

- **Icont** supports the Commando facility of MPW. If the **icont** command is executed via *option-enter* (instead of just *enter*), or if **icont...** is executed (**icont** followed by the ellipsis character, *option-;*), a dialog box will come up that allows specification of all options permissible as command line parameters to **icont**. (The *option-enter* method of invoking Commando is available only with version 3.x of MPW and later). Since the dialog box has "prompts" for most of the available options, it is an alternative to using the manual or on-line help to remember options that are infrequently used.

- The **I**PATH environment variable, which specifies where the linker should search for library modules referenced in link directives, is a *comma*-separated list of directory (folder) names (just like the **Commands** shell variable (this differs from most versions of Icon, which use a space-separated list). The directory names *must* have a colon as the last character, as is the MPW convention.
- An extensive on-line help facility is available that contains summary information about the **icont** command, the **iconx** command (explicit invocation of the Icon interpreter), and about many aspects of the Icon language itself. To access MPW Icon Help, enter:

**ihelp**

**ihelp** is an alias set up by **UserStartup•Icon** that invokes the MPW help system.

- The background tool execution feature of MPW 3.0 is available in MPW Icon (as it is with most MPW tools). **icont** or an Icon program can run in the background under the Finder (or MultiFinder for pre-System-7).
- The MPW “spinning beachball” cursor spins during execution of **icont** and during execution of Icon programs. During Icon garbage collections, the cursor changes to the wristwatch shape.
- Normally, the Icon interpreter, **iconx**, must be installed for Icon programs to run. MPW Icon now has the facility to bundle the interpreter and icode file into a single Macintosh file, giving it the appearance and functionality of normal non-interpreted MPW tool. Of course, since it contains the interpreter, it is some 150K bytes larger than its corresponding icode-only file. This feature is useful for creating MPW tools to be run on systems that do not have MPW Icon installed. To convert an icode file to a stand-alone tool, execute

**IconMPWTool icode-file-name ...**

- Most of the error messages are formatted such that either part or all of the message can be “executed” under MPW to call up the offending file and line. For example, the interpreter error message

**File x.icn; Line 2 # "b": missing semicolon or operator**

is a valid MPW command that will open the file **x.icn** and select line 2. In a few cases, only a portion of the message is executable and must be selected.

- Files created by execution of an Icon program (by **open(filename,"w")**) are MPW text files; that is they have the same type and creator as files created within MPW: type = 'TEXT' and creator = 'MPS '. Therefore their icons in Finder windows are the same as for MPW text files, and double clicking the icon invokes the MPW Shell to edit the file.
- Ucode files are given file type = 'TEXT', creator = 'UCOD'. Since they are human-readable text files, the type of 'TEXT' makes them accessible to text editors (such as MPW Shell). The type 'UCOD' causes them to have a generic document icon,

rather than the MPW Shell text file icon, to help distinguish them from Icon source files and other “normal” text files.

- The *new line* designation `\n` and *carriage return* `\r` have their usual values reversed in MPW Icon. This practice is “inherited” from MPW C. The codes are reversed because the new line code used in Macintosh text files is the carriage return (rather than the *line feed* as used by UNIX), and will not likely cause any problems. Nonetheless, subtle effects could be introduced, such as `image(string('\n\r'))` producing `"\n\r"` on UNIX systems and `"\r\n"` on Macs.
- I/O to “tty” files (i.e., to windows) can be fully buffered (normally it is “line buffered”). Specifying fully buffered output allows considerably faster output to windows, but has some undesirable side effects.

When fully buffered, buffered window output is not actually written to the window until the buffer is full or an input operation is requested. Thus, output to windows between lengthy operations performed by a program may not be seen in a timely fashion. Also, standard output and error output can be intermingled in strange ways.

When line buffered, buffered window output is written under the conditions above, but the buffer is also written whenever a *new line* character is output. This causes lines written to standard output and standard error, as well as any other output streams open on a window, to be written in the same order as output by the program.

To specify fully buffered window output, set the environment variable `NoLineFlush`:

```
Set NoLineFlush 1 ; Export NoLineFlush
```

It is recommended that `NoLineFlush` be used only when the additional speed is especially advantageous, since it often creates confusing output.

- The Macintosh Programmer's Workshop Shell's default stack size works well for almost all Icon programs. However it may have to be increased for certain exceptional ones. Experience so far shows that this will be required extremely rarely. The default stack size can be increased to handle virtually any requirement — see the MPW documentation for the method of increasing the stack size.
- The `ICONCORE` environment variable, which causes a core dump on error termination in many implementations of Icon, simply causes an *abort* in MPW Icon. The only apparent difference between an abort and a normal termination in the MPW environment is that a sequence of MPW commands will terminate if a program aborts, regardless of the setting of shell variable `Exit`.
- The MPW console (i.e. window) input routines function such that a console input operation transfers the entire line containing the insertion point, or if a range of text is selected, the selected text. This may cause some incompatibility with existing Icon programs that issue a prompt and then accept console input on the same line as the prompt. For example, in

```
writes("How many?")      # writes() outputs no new line character
```

```
n := read()
```

the value of n will contain the text "How many?" followed by the entered text. The problem can be corrected by causing input strings to be on lines by themselves:

```
write("How many?")      # write() outputs a new line character
n := read()
```

- Installation script

An MPW script is provided with MPW Icon to automate the installation process.

- UserStartup•icon

The MPW commands required to initialize MPW for use of Icon are encapsulated in a separate UserStartup file, UserStartup•Icon. This change complements the change in MPW 3.0 to automatically execute all files having names beginning with UserStartup• as MPW initializes upon startup. If you are running under an older version of MPW, insert the command

Execute UserStartup•Icon

into your UserStartup file.

- Command line options

The usual Icon requirement that all command line options must precede file names is relaxed for MPW Icon. The normal Icon option-ordering requirement is inherited from UNIX and is contrary to MPW's command line conventions, and had to be eliminated for Commando support to work.

- The -m option to icont (for macro preprocessing) is not implemented.
- The -x option to icont (for automatic execution after translating/linking) is not implemented.
- The function system() is not supported.
- Pipes are not supported. A file cannot be opened with the "p" option.

## 8. Bugs

A complete list of known bugs in Version 9 of Icon is given in [2].

## 9. Differences Between Version 9.0 and Earlier Versions of MPW Icon

There are a number of differences between Version 9.0 and earlier versions of MPW Icon, but few are likely to cause a difference in execution of Icon programs written under the older versions. Most of the changes are upward compatible extensions [2]. If your

Macintosh has relatively small memory, watch out for problems caused by memory shortage, since successive versions of Icon tend to grow in memory requirements.

Changes in version 9.0:

- The delay() built-in function is now implemented.

Changes in versions 8.x and earlier:

- Several new features were introduced in MPW Icon version 7.5 and are carried into version 9. They are described in detail in section 7 of this document:

Commando dialog box support for `icont`  
Background running under MPW 3.0  
Spinning beachball cursor  
Wristwatch cursor during garbage collection  
On-line help facility  
Ability to create stand-alone MPW tools  
Error messages more MPW-like than in older versions  
Window I/O is line-buffered by default — see `NoLineFlush` option in section 7.  
File type and creator of created files is same as MPW text files (type and creator were null in older versions, making it difficult to edit them)  
File type of ucode files is set to 'TEXT', for easy viewing with text editors  
Changes in method of specifying total Icon memory allocation (`IIconSize`)  
The `-x` option of `iconx` has not been supported by MPW Icon since version 7.5. Its burden to the MPW implementation outweighs its utility.

- File hierarchy changes

File hierarchy changes are nil since version 7.5, but there are some significant changes from earlier versions. The `bin` folder was done away with for version 7.5. The Icon tools `icont` and `iconx` now reside in the `Icon` folder in the default organization for MPW Icon. The `bin` folder used to hide a number of files that are no longer needed, namely `itran`, `ilink`, `iconm`, and `iconx.hdr`. As with all implementations of Icon since version 7.5, the translator (`itran`), linker (`ilink`), and control program (`icont`) have been combined into a single program (`icont`). Besides reducing the number of files needed to work with Icon, other benefits are gained such as increased speed of translation and linking, reduced disk space requirements, and generally simpler management of the Icon system.

- Change of environment variable names

The environment variable `Icon` had been named `IconBin` in previous versions.

- Memory management change

Since version 8.7 of MPW Icon, fixed, multiple memory regions are implemented. This improves Icon's ability to expand memory as needed while the program

executes. Prior versions used a single expandable region, which often could not expand due to non-relocatable regions being allocated nearby. The environment variable ICONSIZE is no longer needed. The other iconx region-allocation environment variables (BLOCKSIZE, STKSIZE, etc.) are still available to tune initial region allocation.

## **10. Reporting Problems**

Problems with MPW Icon should be noted on a trouble report form (Trouble Form???) on the distribution diskette) and sent to

Icon Project  
Department of Computer Science  
Gould-Simpson Building  
The University of Arizona  
Tucson, AZ 85721  
U.S.A.

(602) 621-4049

icon-project@cs.arizona.edu (Internet)  
... {uunet, allegra, noao}!arizona!icon-project (uucp)

If a program is involved, enclose it and any relevant data on a diskette.

## **11. Registering Copies of Icon**

If you received your copy of MPW Icon directly from the Icon Project, it has been registered in your name and you will receive the Icon Newsletter without charge. This Newsletter contains information about new implementations, updates, programming techniques, and information of general interest about Icon.

If you received your copy of MPW Icon from another source, please fill out a registration form (Registration in the documents on the distribution diskette) and send it to the Icon Project at the address listed above. This will entitle you to a free subscription to the Icon Newsletter and assure that you receive information about updates.

## **Acknowledgements**

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

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

Bob Alexander adapted Icon to the Macintosh under MPW.

## **References**

1. R. E. Griswold, *An Overview of the Icon Programming Language*, The Univ. of Arizona Tech. Rep. 90-6c, 1992.
2. R. E. Griswold, C. L. Jeffery, and G. M. Townbsend, *Version 9.0 of the Icon Programming Language*, The Univ. of Arizona IPD236, 1994.