Version 9 of Icon
for Microsoft Windows

Clinton L. Jeffery
February 13, 1998
Technical Report CS-97-9

Abstract

This document is your one-stop guide to installing and running the Icon programming language under the Microsoft Windows environment. It serves as the primary reference for the Wi programming environment and the Windows-specific language extensions to Icon.

Icon Project Document IPD271a
http://www.cs.utsa.edu/research/icon/ipd271.htm

Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249

Contents

1. Introduction

The Microsoft Windows implementation of Version 9 of Icon runs on 386 or higher PCs or PC clones with an operating system that can execute 32-bit Windows binaries, including Microsoft Windows 3.1, Windows95, and Windows NT. 4 MB of RAM is required, and 8 or more MB of RAM is strongly recommended. 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.

The basic reference for Version 9 of Icon is the third edition of the book The Icon Programming Language [1]. This book is available from the Icon Project at The University of Arizona and in better bookstores. The graphics facilities are detailed in a separate report [2]. Windows Icon and the Wi environment are products of Clinton Jeffery of the University of Texas at San Antonio. Send requests, and bug reports to `jeffery@cs.utsa.edu`. General Icon language questions can be sent to `icon-group@cs.arizona.edu`.

2. Installing Windows Icon

Insert diskette #1 and run `a:\setup.exe` to install Windows Icon from drive a:. You may substitute a different drive letter and directory name if your source files are located elsewhere. For example, on a CD-ROM distribution you might run `d:\mswin\setup.exe`.

During installation, you will be asked for a drive and directory into which the Icon files will be installed, which defaults to `C:\WINICON`. Files will be installed into several subdirectories of the location you specify. Installation also results in the creation of a Windows Icon program group with a set of icons that allow you read on-line documentation, uninstall the software, and launch Windows Icon. The collection of Windows Icon icons will look something like this:

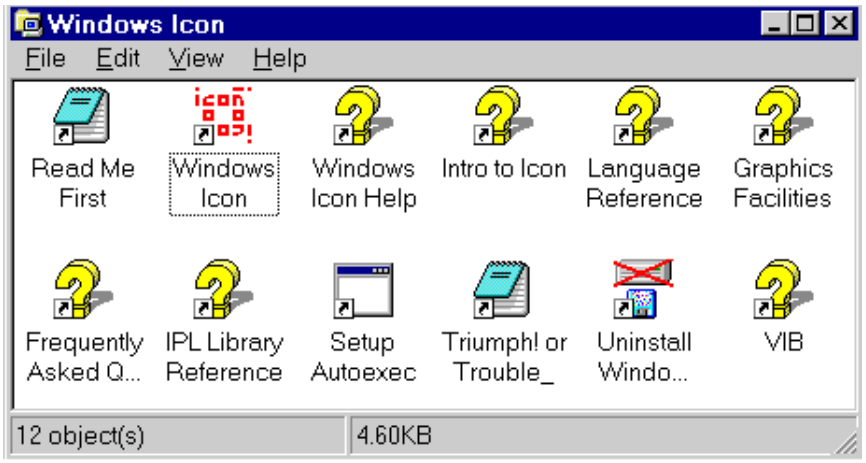

Figure 1: After Icon Installation

Complete installation also requires setting some environment variables, such as IPATH and LPATH, to indicate the location of the Icon Program Library directories. This is done differently on different versions of Windows. See Section 5.3 below for details. To install Icon on Windows 3.1 or Windows for Workgroups, you must also install OS support for the Microsoft Win32 API, provided by a package from Microsoft called "Win32s". Many people already have Win32s as a result of installing other 32-bit binaries. If you do not have a version of Win32s installed, you can obtain one from Microsoft or by FTP to ringer.cs.utsa.edu in directory /pub/icon/nt/win32s. Copy all the files there onto a floppy disk, insert the disk in your Windows 3.1 machine, launch Windows, and run a:setup.exe (or b:setup.exe if the floppy is in your b: drive). Follow the setup program's instructions.

Within the directory that Icon was installed, the most important subdirectory is the `bin` directory, which contains the Icon system binaries. Two executable binary files are required to use Icon, the executor and the compiler that produces virtual machine code files for the executor. The executor and compiler are normally invoked by the integrated development environment called *Wi* (pronounced "Wye" or sometimes "why?"), an editor/project builder that provides a visual interface to the program development process. Applications that provide a graphical interface are usually constructed by means of a visual interface builder called VIB, a drawing program that generates Icon code for a program's interface. The following table summarizes the contents of the `bin` directory:

wiconx.exe
: Win32 executor, with graphics facilities

wicont.exe
: Win32 compiler, with graphics facilities

nticonx.exe
: Win32 command-line executor, no graphics, no Win3.1 support

nticont.exe
: Win32 command-line compiler, no graphics, no Win3.1 support

wi.bat
: Windows Icon integrated development environment (IDE)

vib.bat
: Visual Interface Builder, constructs graphical user interfaces

noop.bat

no-op batch file, invoked for direct (command-line) execution

Icon's `bin` directory, normally `C:\WINICON\BIN` should be added to your PATH specification. The names start with an initial w- (or nt-) that distinguishes them from Icon for MS-DOS.

The distribution is contained in several files a compressed archive format. Files are uncompressed and extracted automatically by the installation program.

The distribution files are organized for distribution on four 1.44MB diskettes. They may also be installed from a single combined directory on a hard disk or CD-ROM.

Diskette 1

setup.exe
: Installation program

setup.cfg
: Installation configuration file

winicon.ar
: Windows Icon binaries

readme.txt
: Installation overview and recent notes

trouble.txt
: Trouble Report form

thisdisk
: Installation disk #1 identification

Diskette 2

uninstal.exe
: UnInstallation program

uninstal.cfg
: UnInstallation configuration file

ipl.ar
: Icon Program Library, non-graphics files

thisdisk
: Installation disk #2 identification

Diskette 3

gipl.ar
: Icon Program Library, graphics files

thisdisk
: Installation disk #2 identification

Diskette 4

data.ar
: Icon Program Library, data files

thisdisk
    Installation disk #2 identification

3. Running Windows Icon

Windows Icon is normally used by means of the programming environment, Wi. Windows95 and NT users may also invoke the compiler and executor directly on the command-line. This section includes instructions on using the programming environment, followed by a brief discussion of the command-line tools.

3.1 Editing, Compiling, and Executing Programs in Wi

Double-click the Windows Icon icon to launch *Wi*, the Windows Icon programming environment. *Wi* is written in Icon and allows you to edit, compile, and execute programs from a standard Windows interface. Wi's detailed documentation (this file) is accessed on-line through its Help menu. To start, you must select the name of a file to edit, in following dialog:

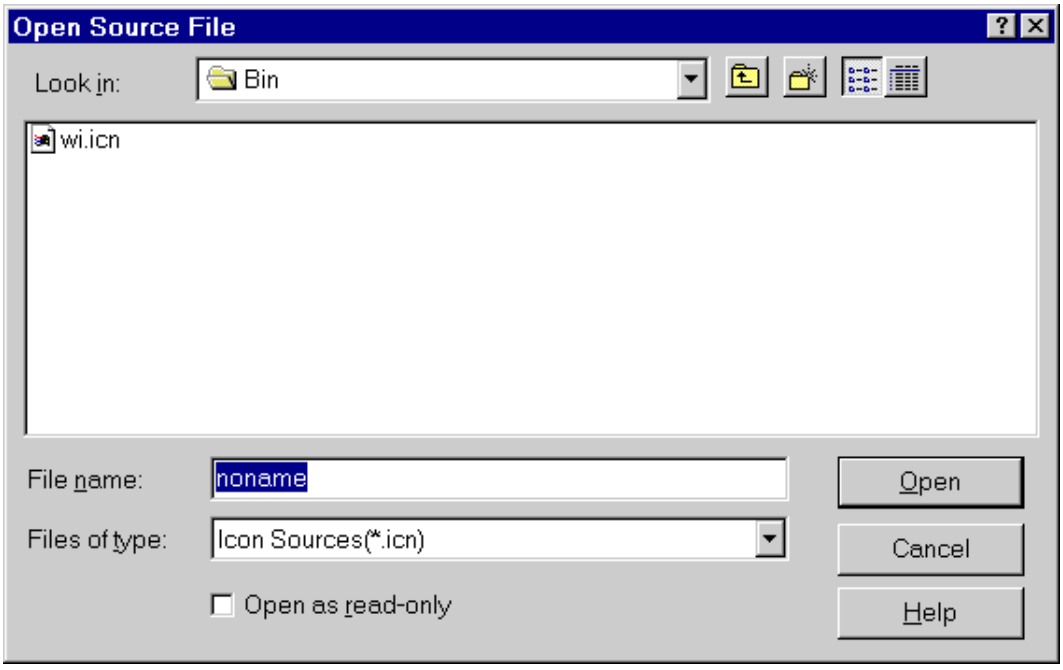

Figure 2: Opening a File within Wi

You can easily select an existing Icon source file, or name a new one. If you click "Open" without choosing a name, you will be given the default name of "noname.icn". Icon source files generally must use the extension `.icn` and should be plain text files without line numbers or other extraneous information. Editing your program occurs within the main *Wi* window, which might look like this:

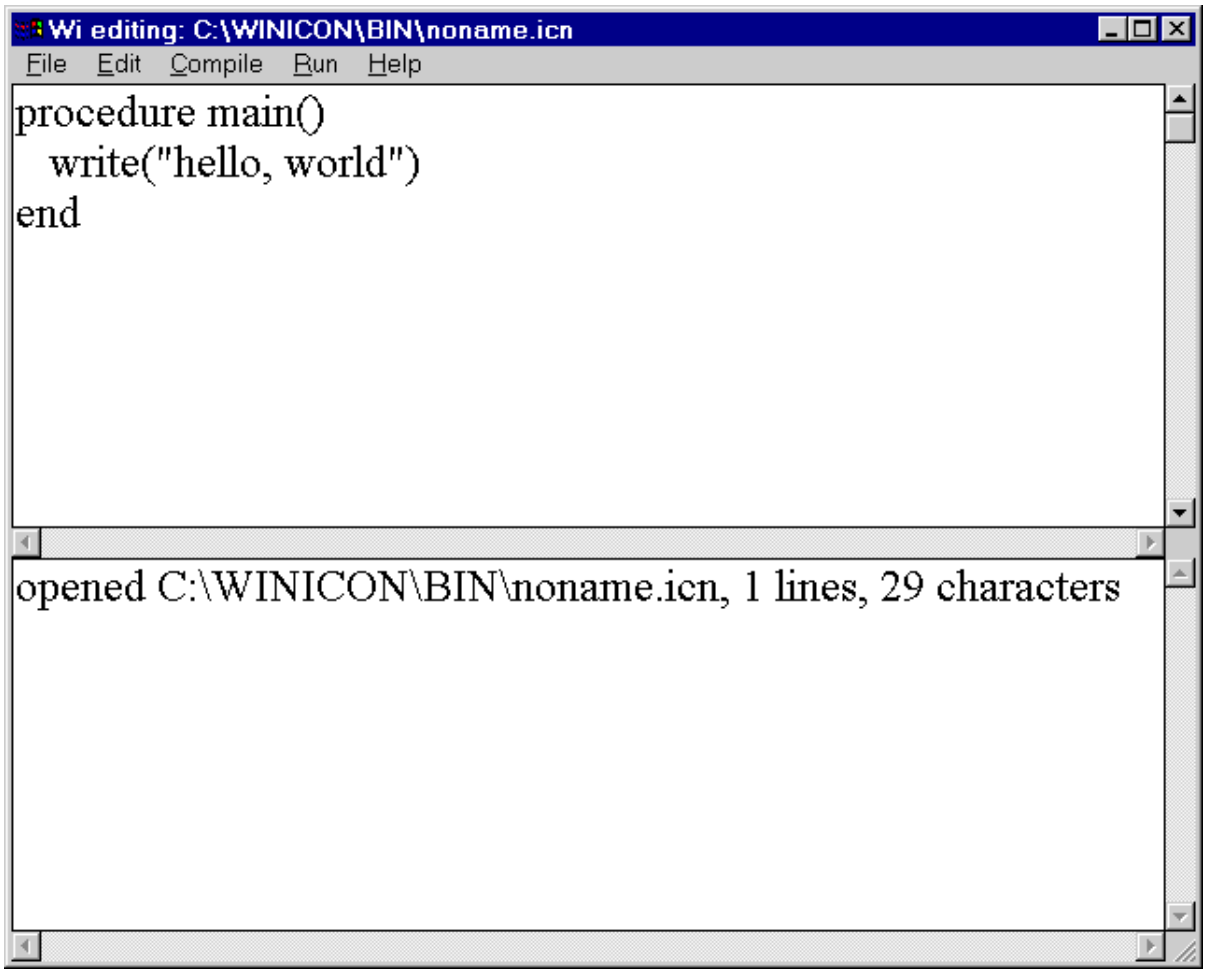

Figure 3: Wi's Editing Interface

The top area shows program source code, while the bottom portion shows messages such as compiler errors. You can change the font and the number of lines used to show messages from the Edit menu.

When you are done editing your program, you can save it, compile it, or just "make" (save, compile and link an executable) and run your program with menu options. The Arguments command in the Run... menu let's you specify any command-line arguments the program should be given when it is executed.

3.2 Error Handling

Compile errors result in a message in which the editor highlights the line at which the error was detected, like this:

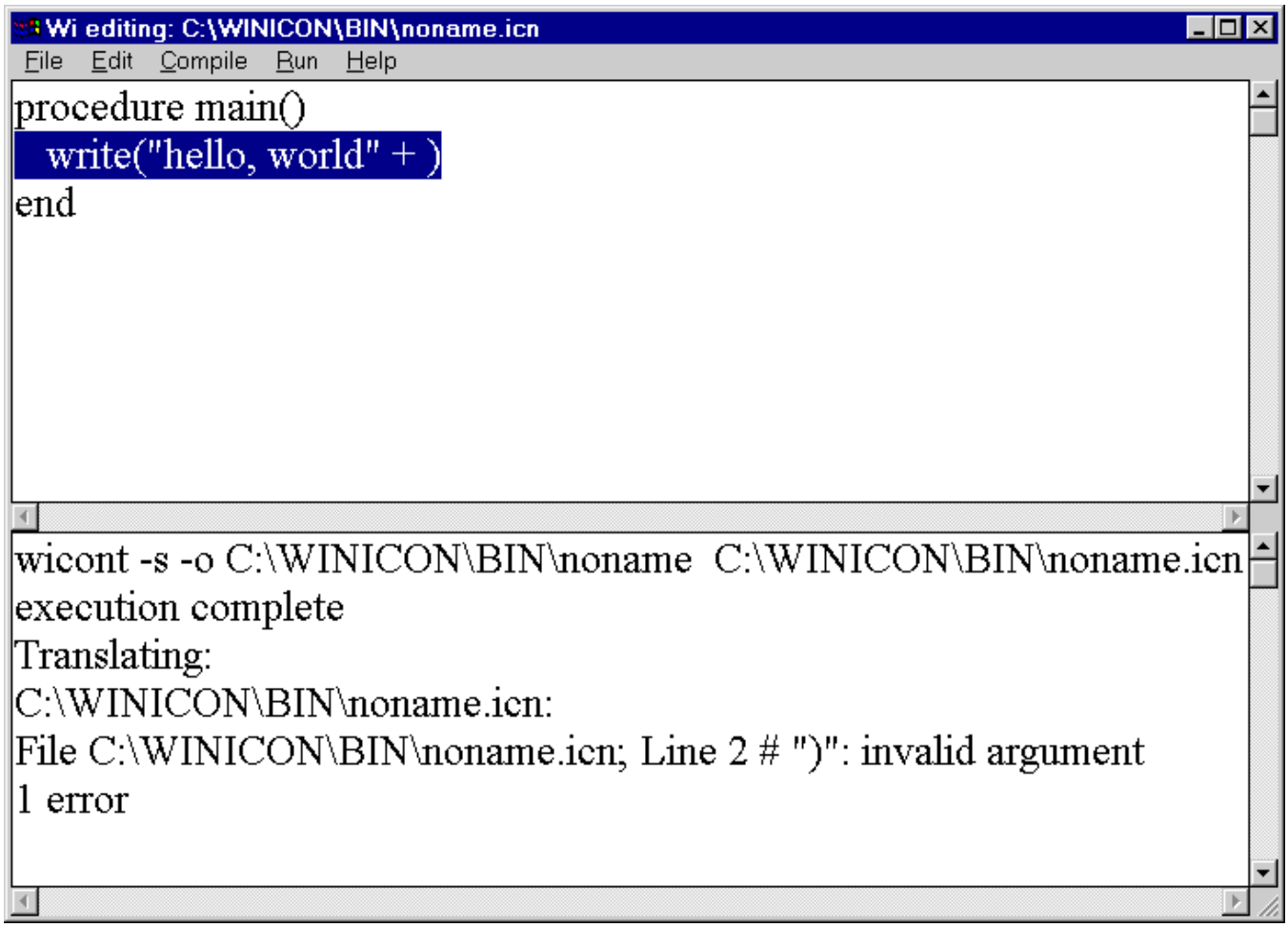

Figure 4: Reporting Compiler Errors in Wi

Run-time errors also result in a message for which the source line is highlighted. The message for a run-time error includes Icon's standard traceback of procedures from `main()` to the procedure in which the error occurred. When the error messages get long, you can either increase the number of lines for the message window (as was done here) or scroll through the message window's entire text using the scrollbar.

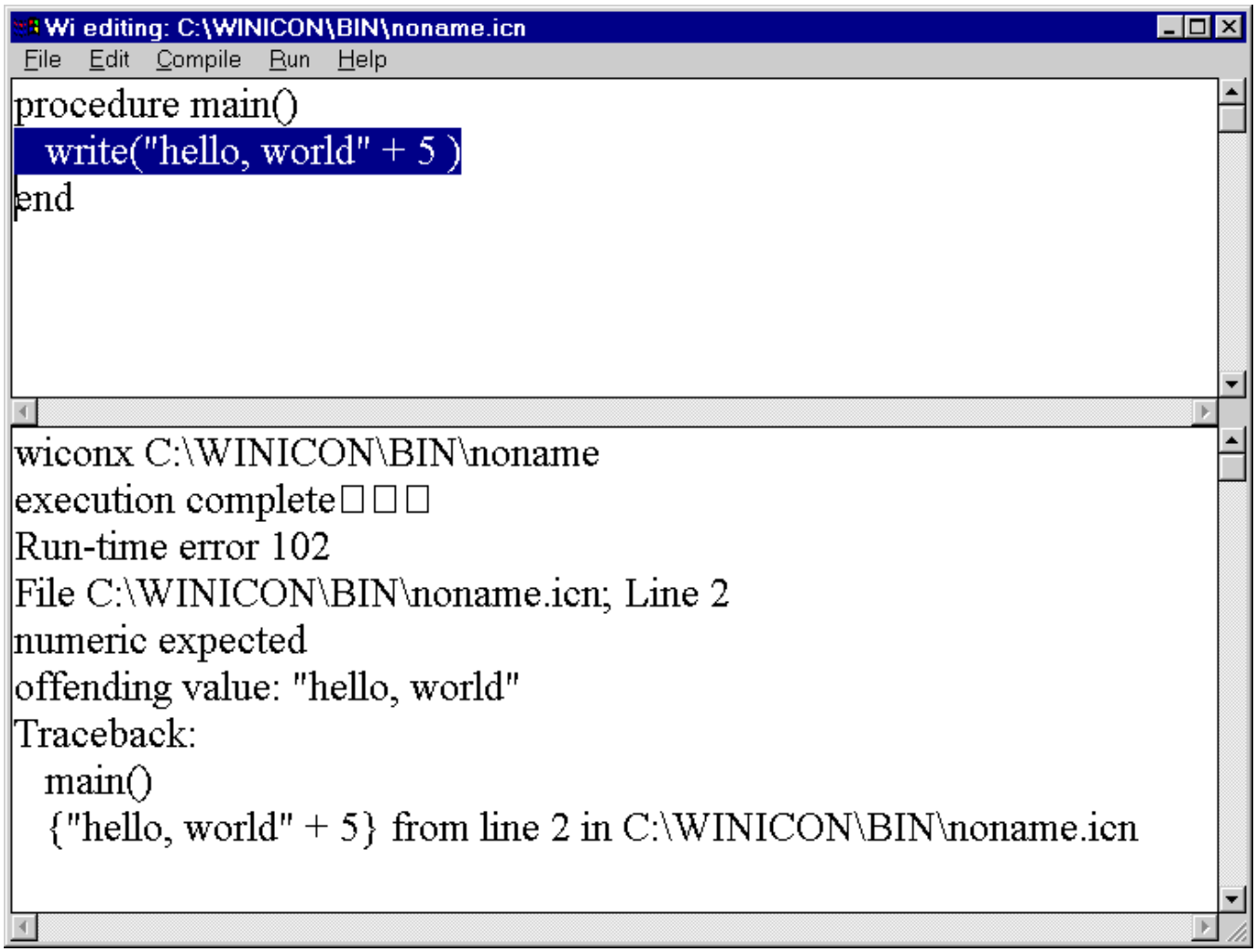

Figure 5: A Run-time Error in Wi. The message window's size is adjustable.

3.3 Projects

Wi works on programs comprised of any number of source files. By default, Wi assumes a single file program consisting of the current source file. If you are working on a single-file program, and you open a new `.icn` source file, Wi switches the editor and compile and link commands to work on a different program.

Project files are plain text files with extension `.icp`. They list source files, one file per line. When you open a project file, Wi goes into "project mode", and adds the source files in the project to your File menu, allowing you to switch easily between files in the project. If you subsequently open a source file not in the project, Wi asks if you want to add that source file to the project, or switch out of project mode and edit that file as a separate program. In general, project files allow you to "make" large projects efficiently. Underneath the covers, Wi invokes the other Icon program executables to do the work of compiling and running programs, described below.

When Wi "makes" a program executable, it recompiles those modules listed in the project file whose

modified time is newer than their corresponding object files. On the other hand, Wi knows nothing about link declarations embedded in source files, and recompilation will not be triggered by such dependencies. When you use Wi, you should generally use link statements for library files (such as Icon Program Library modules), and use `.icp` files for your own sources. Files listed in the .icp file must not also be referenced in a link statement; linking the same module twice causes link errors. For projects, the executable `.bat` that is produced by "make" is named after the first program in the project file.

3.4 Wi Options

Wi supports command-line options to be passed to wicont with the Compiler Options... menu item in the Compile menu. Command-line arguments passed to the Icon program when it is run are supplied via the Arguments... menu item in the Run menu.

The font in the edit window can be picked from the Font... command in the Edit menu. The number of lines to use for messages can be picked with the Message window... menu item in the Edit menu.

The font and message line options, as well as a default window width and height, may be specified in a file called `WI.INI` that Wi reads when it starts up. The file `WI.INI` from the current directory is used, unless a WICONINI environment variable is set, in which case it is taken to be the pathname of the initialization file that Wi is to use. An example `WI.INI` file is the following:

```
width=800
height=800
font=times,28
msglines=3
```

3.5 Running Windows Icon - Command Line Tools

Command line execution of Icon programs is similar on many platforms and is described in the Icon Programming Language book and other reports. Windows Icon includes a command-line compiler (nticont) and virtual-machine interpreter (nticonx) that do not support graphics facilities, but do run from the command-line on Windows95 and NT. For example, an Icon program in the file prog.icn is compiled into virtual machine "icode" by the command:

```
nticont prog.icn
```

Command line execution does not work under Windows 3.1, whose MS-DOS prompts cannot invoke 32-bit Windows binaries. If command line execution is desired, such machines should use MS-DOS/386 Icon instead.

Windows Icon also includes a compiler (wicont) and virtual-machine interpreter (wiconx) that do support graphics facilities and do run on Windows 3.1; while these tools can be invoked from the command-line under Windows95 and NT, they are usually invoked by Wi. The instructions below refer to nticont and nticonx, but generally would also be true for wicont and wiconx. The Windows icode files that result of compilation have the extension .bat, for example, the above compilation would produce a file prog.bat. Under Windows NT or Windows95 this file can be run by

```
prog
```

Alternatively, nticont can be instructed to execute the icode file after translation by appending a -x to the

command line, as in

```
nticont prog.icn -x
```

If nticont is run with the -x option, the file prog.bat is left and can be run subsequently using an explicitly named executor as described above.

The extension .icn is optional on the command line. For example, it is sufficient to use

```
nticont prog
```

4. 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:

hello.icn

This program prints the Icon version number, time, and date. To run this test, launch the Wi program, open file hello.icn, and select the Make and Run menu options. To test the command line tools:

```
nticont hello
hello
```

Note that this can be done in one step with

```
nticont hello -x
```

roman.icn

This program converts Arabic numerals to Roman numerals. To run this test, launch the Wi program, open file roman.icn, and select the Make and Run menu options. To test the command line tools:

```
nticont roman -x
```

and provide some Arabic numbers from your console.

If these tests work, your installation is probably correct and you should have a running version of Windows Icon.

5. More on Running Icon

For simple applications, the instructions for running Icon given in Section 3 may be adequate. The Icon compiler 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.

5.1 Arguments

Arguments can be passed to the Icon program by entering them in Wi's Arguments... item in the Run menu, or 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.bat, passing its main procedure a list of two strings, "text.dat" and "log.dat". The program also can be translated and run with these arguments with a single command line by putting the arguments after the -x:

```
nticont prog -x text.dat log.dat
```

These arguments might be the names of files. For example, the main procedure might begin as follows:

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

5.2 The Compiler

The Icont compiler 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,

```
wicont prog1 prog2
```

translates the files prog1.icn and prog2.icn and produces one icode file, prog1.exe. In addition to supplying files on the command line, files may be linked or included using appropriate commands in the source file.

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

```
wicont -o probe prog
```

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

If the -c option is given to wicont, the translator stops before producing an icode file and intermediate ‘‘ucode’’ files with the extensions left for future use (normally they are deleted). For example,

```
wicont -c prog1
```

leaves prog1.u1 and prog1.u2, instead of producing prog1.bat. These ucode files can be used in a subsequent wicont command by using the .u1 name. This saves translation time subsequently. For example,

```
wicont prog2 prog1.u1
```

translates prog2.icn and combines the result with the ucode files from a previous translation of

prog1.icn. Note that only the .u1 name is given; the .u2 name is implied. The extension can be abbreviated to .u, as in

```
wicont prog2 prog1.u
```

Ucode files also can be added to a program using the link declaration.

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

The -t option causes &trace to have an initial value of -1 when the icode file is executed. Normally, &trace has an initial value of 0.

The option -u causes warning messages to be issued for undeclared identifiers in the program.

5.3 Environment Variables

When an icode file is executed, several environment variables are examined to determine execution parameters. The values assigned to most of these variables should be numbers.

Environment variables are particularly useful in adjusting Icon's storage requirements. Particular care should be taken when changing default values: unreasonable values may cause Icon to malfunction.

The following environment variables can be set to adjust Icon's execution parameters. Their default values are listed in parentheses after the environment variable name:

BLKSIZE (500000)
: This variable determines the size, in bytes, of the initial region in which Icon allocates lists, tables, and other objects. If additional block regions are needed, they may be smaller.

COEXPSIZE (2000)
: This variable determines the size, in 32-bit words, of each co-expression block.

ICONFONT (fixed)
: This variable contains the name of the default font used by Windows that are opened. Any fixed-pitch font may be the default. Example syntax is:

```
set ICONFONT=Lucida Sans Typewriter
```

IPATH (undefined)
: This variable contains a list of directories, separated by spaces. The directories are searched for library files specified by link declarations.

LPATH (undefined)
: This variable contains a list of directories, separated by spaces. The directories are searched for header files specified by preprocessor $include directives.

MSTKSIZE (10000)
: This variable determines the size, in words, of the main interpreter stack.

NOERRBUF (undefined)
: If this variable is set, &errout is not buffered.

STRSIZE (500000)
: This variable determines the size, in bytes, of the initial region in which strings are stored. If

additional string regions are needed, they may be smaller.

TRACE (undefined)
: This variable initializes the value of &trace. If this variable has a value, it overrides the translation-time -t option.

WICONLOG (undefined)
: This variable contains the name of a file to which output from wicont and wiconx will be written after their execution completes.

6. Features of Windows Icon

Windows Icon supports all the features of Version 9.3 of Icon, including graphics facilities, with the following exceptions and additions:

- Path specifications can be entered using either a / or a \. Examples are:

```
A:\ICON\TEST.ICN
A:/ICON/TEST.ICN
```

- Wildcards are allowed in filename command-line arguments.
- Files with extension .icp are treated as command-line argument lists (one argument per line). Set up .icp files for batch jobs, associate .icp extensions with wicont.exe, and you can run the batch job by double-clicking it in the file manager.
- wicont allows an option -q that causes it to exit on termination without waiting for a user keystroke; this is useful in batch compilation.
- system() supports launching Windows applications from inside Icon.
- A console window opens if any of the (non-redirected) standard input, output, or error files is accessed.
- .bmp image files are supported for reading and writing.

6.1 Native Windows User Interface Access

Version 9.3 of Windows Icon provides the following built-in user interface components that make use of native Windows features. Additions are still being made to this set, and the facilities described here being further integrated into the Icon Program Library so that they are used automatically in Icon programs that provide a visual interface. This section summarizes features; the following section provides a reference guide.

The goal of the native facilities is not to provide the entire Windows repertoire, any more than the entire X Window repertoire is provided to UNIX users. Instead, features have been chosen that are (1) important to the Windows look and feel, (2) general enough to be implementable on other platforms, and (3) can co-exist or be integrated with existing IPL facilities.

Menus

A menu bar is created with a call like:

```
WinMenuBar(W, ["&File", "&Open", "&Save", "E&xit"],
           ["&Edit", "C&ut", "&Paste", "C&opy"],
           ["&Help", "&About"])
```

This function converts approximately the top text line of the window into a menu bar. The appearance of the above example is given in Figure 6. When menu items are selected, they are produced as entire strings (such as `"&Open"`) by Event().

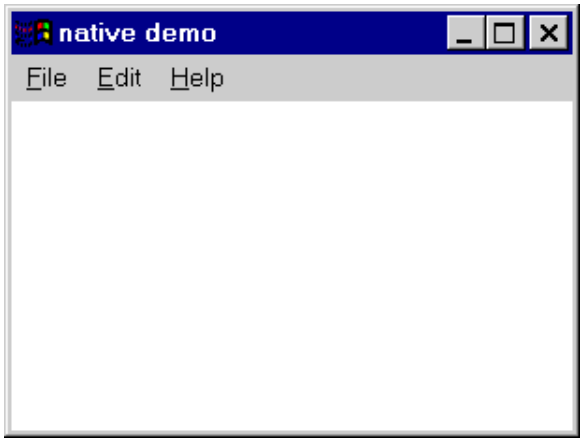

Figure 6: a Windows menu bar

Scroll Bars

A scroll bar is created with a call like

```
WinScrollBar(W, "sb_1", x, y, wd, ht)
```

This function places a scrollbar with a particular size and position, which default to a standard size on the right edge of the window. The appearance of a typical scroll bar is illustrated in Figure 7. When scroll bar activity takes place, the scroll bar's string id is produced (in this case, `"sb_1"`) by Event(), and `&x` and `&y` are both set to the scroll bar's position.

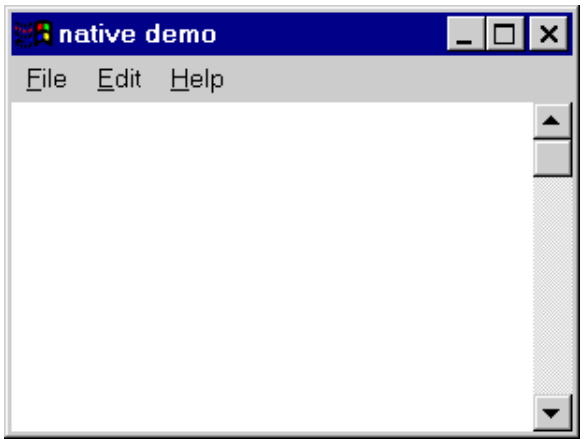

Figure 7: a Windows scroll bar

Buttons

A button is created with a call like

```
WinButton(W, "hello", x, y, wd, ht)
```

This function places a button with a particular size and position. The size defaults to a standard size large enough display the button's label. The appearance of a pair of buttons is illustrated in Figure 8. When a button is pressed, the button's string label is produced (in this case, `"hello"`) by Event().

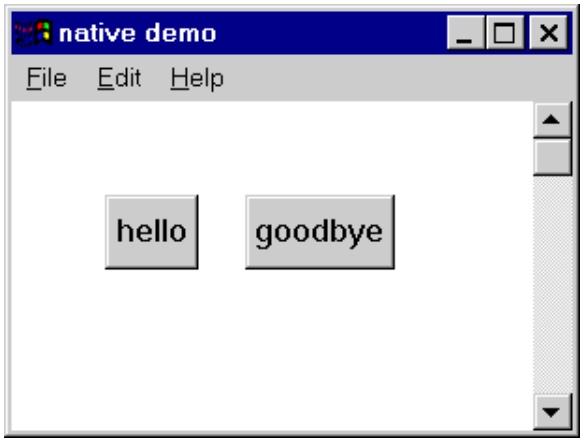

Figure 8: a pair of Windows buttons

Common Dialogs

Several common dialogs are provided for selecting colors, fonts, and files to open or save. These functions return an attribute value or a file name. Examples are illustrated in Figures 9-12.

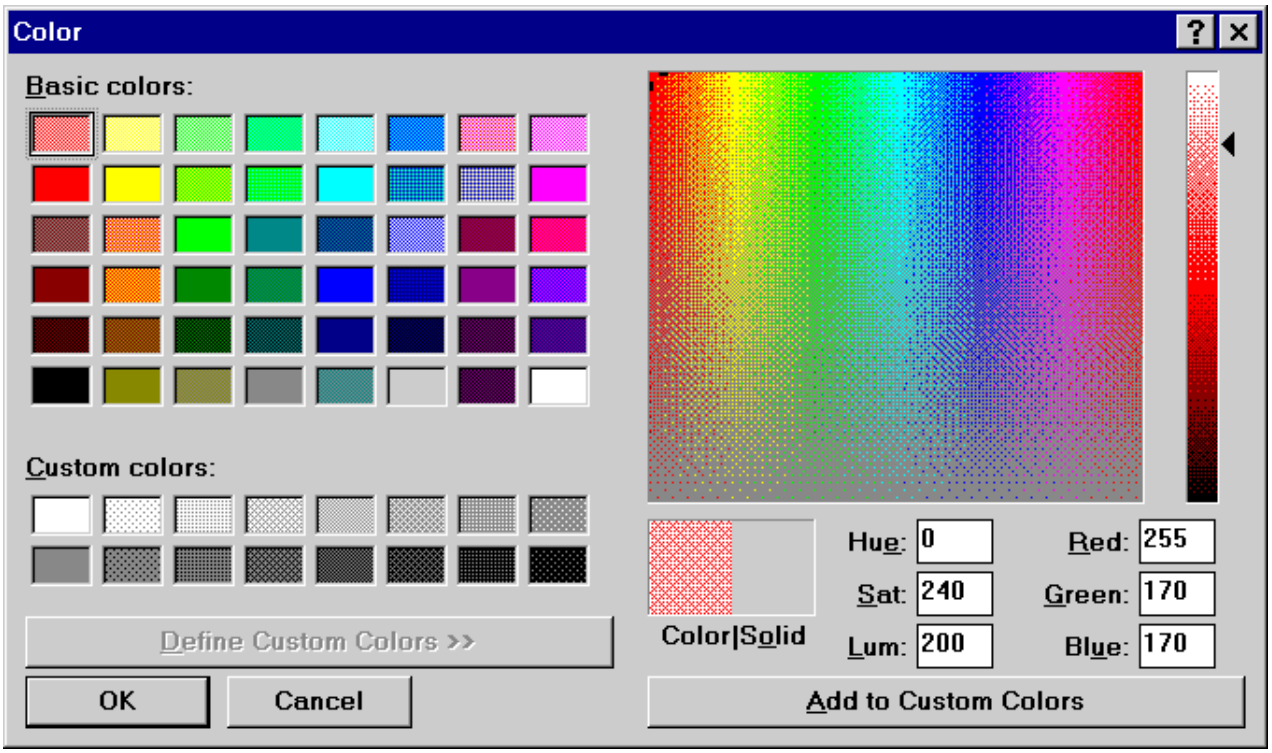

Figure 9: the Windows color dialog

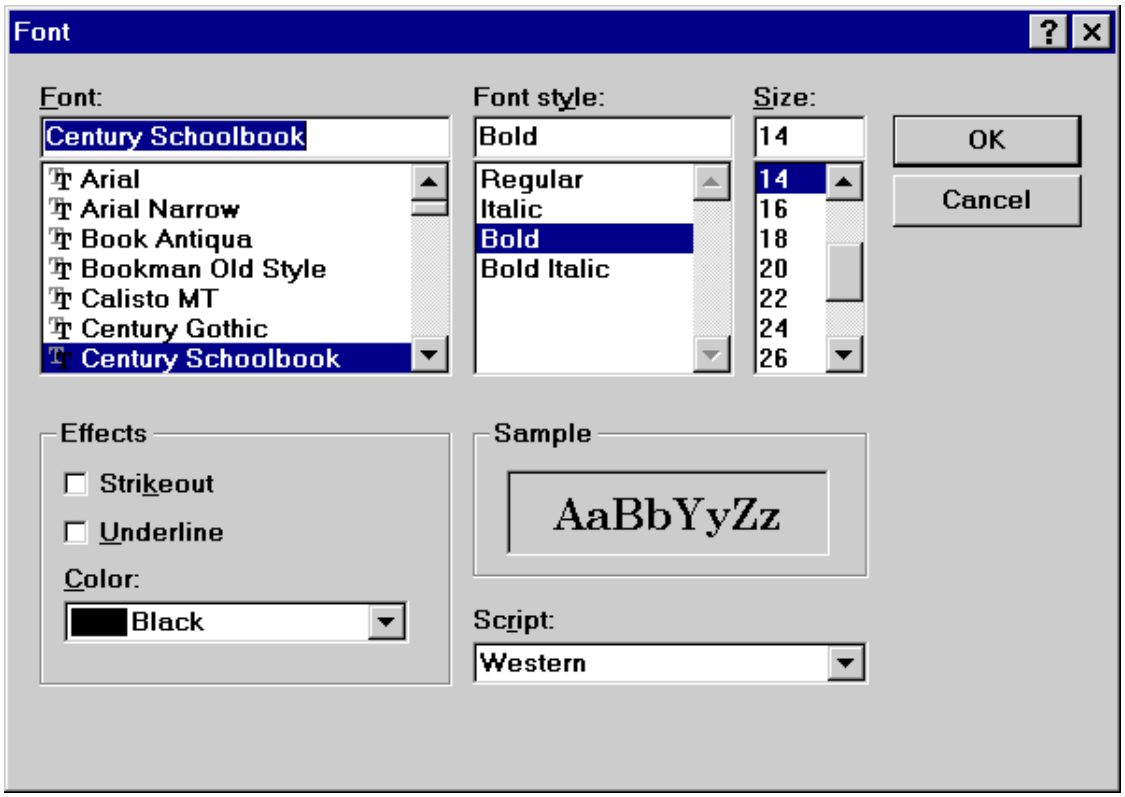

Figure 10: the Windows font dialog

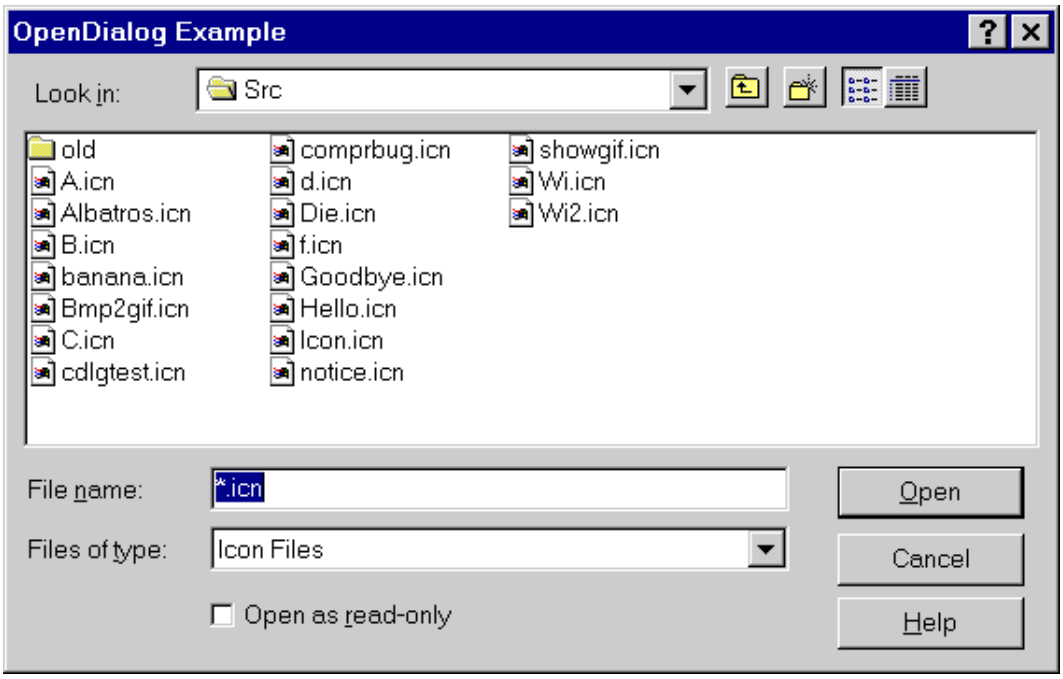

Figure 11: the Windows open dialog

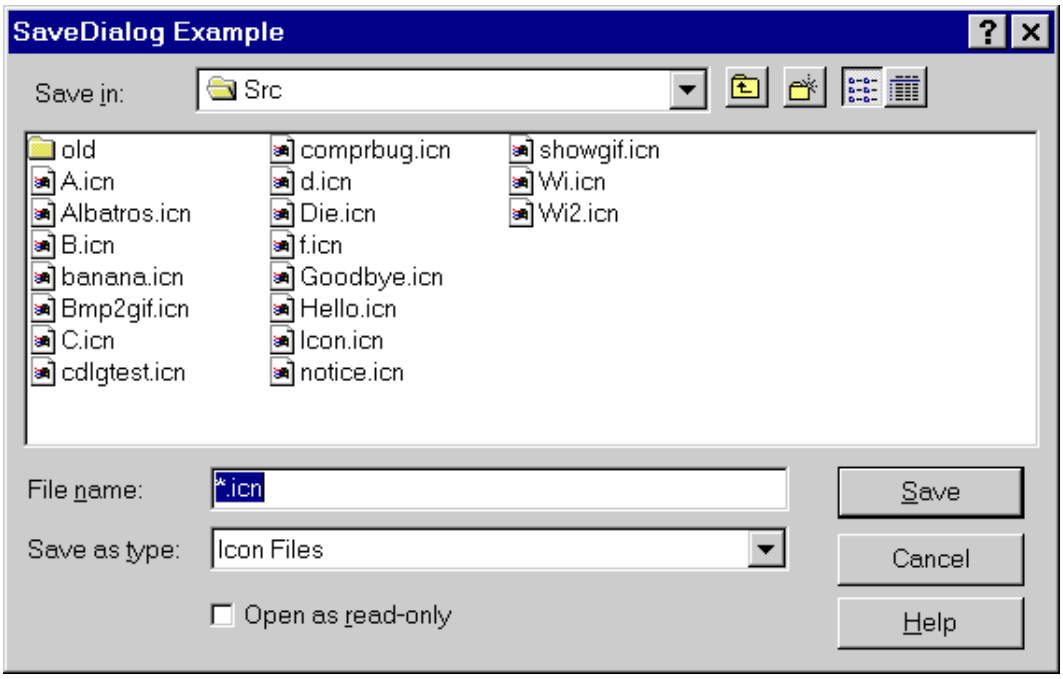

Figure 12: the Windows save dialog

6.2 Native Windows Facilities Function Reference

The following features provide a useful subset of the native Windows interface in a manner that will allow multi-platform implementation. The primary demonstration of these facilities thusfar is the Windows Icon integrated development environment, wi.

---

fattrib(x, s) - get file attribute

fattrib(x, s) returns the value of file attribute s on file or string filename x. Valid attributes include "size" (in bytes), "status" (permissions string in format rwxrwxrwx), and "mtime" (time last modified).

WinButton(s, x, y, wd, ht) - install button

WinButton(s, x, y, wd, ht) installs a pushbutton on w with label s. Whenever the button is clicked, the string label s is placed on the window's event queue.

Defaults:	wd	width of text in system font + 10 pixels
	ht	height of text in system font * 7/4

WinColorDialog(s) - select color

WinColorDialog(s) executes the Windows common dialog for color selection with default color s. Returns a string attribute value corresponding to user's selection, or fails if the user selects Cancel. This

function is called from the Icon Program Library procedure of the same name.

WinEditRegion(s, s2, x, y, wd, ht) - manipulate edit region

WinEditRegion(s, s2, x, y, wd, ht) manipulates a Windows edit region named s. The operation performed depends on argument s2. If argument s2 is omitted, WinEditRegion(s) returns a string containing the current contents of region s. If s2 is supplied and does not start with a !, it is a string to be edited; lines are separated by \r\n. s2 strings starting with ! are commands:

- WinEditRegion(s, "!clear") clears the current selection..
- WinEditRegion(s, "!copy") copies the current selection.
- WinEditRegion(s, "!cut") cuts the current selection.
- WinEditRegion(s, "!paste") pastes the current selection.
- WinEditRegion(s, "!modified") succeeds if region s has been modified since it was last assigned a value
- WinEditRegion(s, "!setsel") sets the selection using parameters x and y as indices..
- WinEditRegion(s, "!undo") undoes the previous operation, if possible.

WinFontDialog(s) - Execute Windows common dialogs for font selection,

with default font s. Returns a string attribute value corresponding to user's selection, or fails if the user selects Cancel.

WinMenuBar(w, L1, L2, ...) - install a menu bar on w.

Each list, which presently may contain only strings, describes one popup menu on the menu bar. The first element of the list is the menu's name that appears on the bar. The remaining elements are the menu options for that popup menu. After a menu bar is installed, Event() returns the strings. For example, after a call: WinMenuBar(["File", "Open", "Save"], ["Help", "About"]) the menu bar would show File and Help menus, and anytime the user selected the Open, Save, or About menu options, "Open", "Save", or "About" would be queued on the event list. Ampersand is used to specify keyboard shortcuts, as in ["&File", "&Save", "Save &As..."].

WinOpenDialog(s1,s2,i,s3,j), WinSaveDialog(s1,s2,i,s3,j) - Execute Windows common dialogs for opening and saving files.

These functions are similar to Icon Program Library procedures with similar names. s1, s2, and i are the caption, default value, and text entry field size. s3 and j specify the filter string and its index. s3 is a string of alternating names and filters, separated and ending in |, of the form "name1|filter1|name2|filter2|...|". It defaults to "All Files(*.*)|*.*|". j supplies the default extension index within s3; it defaults to first pair in filter string. These functions return the file name chosen, rather than assigning it to the global variable dialog_value as the IPL procedures do. They fail if the user selects Cancel.

WinPlayMedia(w, s1, s2, ...) - play a multimedia resource.

String arguments ending in .wav are presumed to be wave sound file names. Strings ending in .mid or .rmi are presumed to be MIDI files. All other strings are treated as MCI command strings, and processed by the Windows Media Control Interface.

7. Known Bugs, Limitations, and Problem Areas

Windows Icon has a number of limitations inherent to the platform, as well as some remaining bugs that may get fixed.

General

- The Wi editor cannot edit files greater than approximately 32K bytes-- the limit of Windows' edit control. Larger source files can be edited in another programmer's editor and still translated and linked by Wi, but jumping to translation or runtime errors in such files will not work. If you have source files >32k you can also break them into smaller pieces, or use Icon's command line tools. See Section 3.3 above.
- Pipes are not supported. A file cannot be opened with the "p" option.
- For files opened in the translated mode, the position produced by seek() may not reflect the actual number of characters written because of the translation of carriage-return/line-feed sequences to line-feed characters.
- system() cannot launch DOS or NT console applications. Such applications can often be run by launching a command-line interpreter to execute them, as in

```
system("cmd /C /Q " || command)
```

- system() may return before the launched application completes if that application opens no window, closes its main window before its execution is finished, or if several windows are opened shortly after the application is launched.
- Under Windows 95, the focus does not return to the command line window after wicont or an Icon program executed by wiconx is launched. Redirecting standard input without also redirecting standard output can cause spurious runtime errors. Redirecting standard input from an empty file is not detected; it is treated as no redirection and a console window is opened for input if the standard input file is read.

Graphics Facilities

- Attribute linestyle is ignored by Windows when the linewidth is greater than 1; linewidths greater than 1 are always drawn using a solid line style.
- Attribute fillstyle does not correctly interpret the value masked. When masked fills are requested, textured fills are performed instead.
- In addition to the standard values "dashed", "striped", "solid", attribute linestyle has values "longdashed", "dashdotted", and "dashdotdotted" corresponding to Windows' standard pen styles for longer dashes and alternating longer and shorter (Icon-standard size) dashes in lines.
- Mutable colors do not work unless the window is the active window, and sometimes do not work as expected.

- Colors are never freed.
- Windows may use dithered colors, resulting in an unattractive appearance in applications where solid colors are expected. Most colors are dithered on 16-color machines, and color-intensive applications are ugly or unusable on those systems.
- References to attributes display, iconpos, iconimage, iconlabel, and visual always fail, as do calls to function Default().
- Attribute pointer supports only "arrow", "cross", "ibeam", "uparrow", and "wait", depicted by the following images. The appearance of these pointers varies slightly on different versions of Windows.

- Although some program examples using attribute drawop work, drawop does not work the same way as on other platforms.

8. Reporting Problems...or Successes

Problems with Windows Icon should be noted on a trouble report form (TROUBLE.TXT, included with the distribution) and sent to

```
Clinton Jeffery
Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249
U.S.A.

(210) 691-5557 (voice)
(210) 691-4437 (fax)
jeffery@cs.utsa.edu
```

In order to minimize response times, the preferred method for reporting problems is by e-mail. I would also like to hear suggestions and success stories from satisfied users; e-mail is great but letters and postcards are even better for this kind of feedback. Think of TROUBLE.TXT as a "registration" for your free software.

Acknowledgements

The design and implementation of the Icon programming language was supported, in part, by grants from the National Science Foundation.

Ralph Griswold and Gregg Townsend collaborated with the author in the development of Version 9.3. Bob Goldberg and Steve Schiavo wrote prototype program launchers that inspired the Wi integrated development environment.

This work was made possible in part by a software donation from Microsoft, a hardware donation from Icon Project, and a UTSA faculty research award. The author is supported by National Science Foundation award CCR-9409082.

References

1. R. E. Griswold and M. T. Griswold, The Icon Programming Language, Peer-to-Peer

Communications, Inc., San Jose, California, third edition, 1996.

2. Gregg M. Townsend, Ralph E. Griswold, and Clinton L. Jeffery, Graphics Facilities for the Icon Programming Language Version 9.3, The Univ. of Arizona Icon Project Document IPD281, 1996.