Calling C Functions from Version 9 of Icon

Ralph E. Griswold and Gregg M. Townsend

Department of Computer Science, The University of Arizona

1. Introduction

Version 9 of Icon [1] supports calling C functions from Icon. In its simplest form, this facility can be used with only a little knowledge of how Icon is implemented. Sophisticated uses, however, require a good working knowledge of Icon data structures and Icon's internal operation [2-4] and RTL [5], the superset of C in which the Version 9 run-time system is written.

There are two approaches to adding C functions to Icon. *External functions* can be added to the Icon interpreter; these functions are then available through the callout() interface to any Icon program that uses this customized version of Icon. *Dynamic loading* allows an Icon program to link a C function at execution time by calling loadfunc(); this approach has several advantages but is only available on a few platforms.

2. External Functions

The Icon function callout(x0, x1, …, xn) allows C functions to be called from Icon programs. The first argument, x0, designates the C function to be called. The remaining arguments of callout() are supplied to the C function (possibly in modified form). In order to provide the necessary flexibility, callout() in turn calls a C function extcall(), which has the prototype

```
dptr extcall(dptr argv, int argc, int *ip)
```

where argv is a pointer to an array of descriptors containing the arguments, argc is the number of arguments, and ip is a pointer to an integer status code. The value returned by extcall() is a pointer to a descriptor if the computation is successful or NULL if it fails (which causes callout() to fail).

A stub for extcall() is provided in extcall.r. This stub should be replaced by an appropriate C function, after which the Icon run-time system must be rebuilt. Although extcall() normally is written entirely in C without the use of RTL constructs, it needs to be processed by rtt, the translator from RTL to C, to insure appropriate definitions and declarations are included.

Designating C Functions

The method of specifying C functions varies with system and application. A simple mechanism is to associate an integer with each function that can be called and use a C switch statement in extcall() to select the desired function. This method is used in the first example in Appendix 1. A better method is to use string names, as illustrated by the second function in that appendix. The C functions to be called must be linked with Icon (presumably through references in extcall()).

Error Handling

The integer status code pointed to by ip is used for error handling. It is –1 when extcall() is called, indicating the absence of an error. If an error occurs in extcall(), the status code should be set to the number of an Icon run-time error [6]. Error 216 should be used if the designated C function is not found.

If there is a descriptor associated with the error, a pointer to that descriptor should be returned by extcall(). If there is no specific descriptor associated with the error, extcall() should return NULL. See the examples in Appendix 1.

If the status code is not –1 when extcall() returns, callout() terminates program execution with a run-time error message corresponding to the value of the status code.

3. Dynamic Loading

On systems that support dynamic loading, the function loadfunc(libname, funcname) loads the C function funcname from the library libname and returns a procedure value. This value can then be used to call the function in the usual manner. To the Icon program, loaded functions appear similar to built-in functions. Appendix 2 presents an example of an Icon program with a dynamically loaded function.

Functions loaded by Icon must provide a particular interface, described below, and so they are usually written specifically for use with Icon. Data structure definitions can be obtained by including the file src/h/rt.h that is distributed with the Icon source code. This file also declares macros and functions that may be useful for data conversion.

C functions must be compiled and installed in a library before they can be loaded by an Icon program. The method for creating a library is system dependent. Here are examples for five particular systems.

SunOS 4.x:	ld –o lib.so file1.o file2.o
Solaris 2.x:	cc –G –K pic –o lib.so file1.o file2.o
Dec OSF1 v2.x:	ld –shared –expect_unresolved '∗' –o lib.so file1.o file2.o –lc
SGI Irix 5.x:	ld –shared –o lib.so file1.o file2.o
FreeBSD:	ld –Bshareable –o lib.so file1.o file2.o –lc

These examples create a file named lib.so from the functions contained in file1.o and file2.o.

The C Function Interface

A C function loaded by Icon has the prototype

```
int funcname(int argc, dptr argv)
```

where argc is the number of arguments and argv is an array of descriptors for the arguments. The first element argv[0] is not included in the count argc. It is used to return an Icon value, and is initialized to a descriptor for the null value. The actual arguments begin with argv[1].

If the C function returns zero, the call from Icon succeeds. A negative value indicates failure. If a positive value is returned, it is interpreted as an error number and a fatal error is signalled. In this case, if argv[0] has been changed, it is printed as the ‘‘offending value’’. There is no way for a C function to suspend, and no way to indicate a null value as an offending value in the case of an error.

4. Data Interface

For either method of calling C from Icon, arguments to the C function are passed as Icon descriptors. The Icon run-time system contains type-checking and conversion facilities for the manipulation of descriptors. Some useful conversion functions are:

cnv_int(dp1, dp2)	Converts the value in the descriptor pointed to by dp1 to an integer descriptor pointed to by dp2, returning 0 if the conversion cannot be performed.
cnv_str(dp1, dp2)	Converts the value in the descriptor pointed to by dp1 to a string string descriptor (qualifier) pointed to by dp2, returning 0 if the conversion cannot be performed.
cnv_real(dp1, dp2)	Converts the value in the descriptor pointed to by dp1 to a real number descriptor (floating-point double) pointed to by dp2, returning 0 if the conversion fails.

Some other useful macros and functions are:

Qual(d)	Tests if d is a descriptor for a string.
IntVal(d)	Accesses the (long) integer value of the integer descriptor d.
MakeInt(i, dp)	Constructs a integer descriptor pointed to by dp from the (long) integer i.
StrLen(d)	Accesses the length of the string in the descriptor d.

StrLoc(d)	Accesses the address of the string in the descriptor d.
qtos(dp, sbuf)	Constructs a C-style string from the descriptor pointed to by dp, placing it in sbuf, a buffer of length MaxCvtLen, if it is small enough or in the allocated string region if it is not. If there is not enough sapce available in the allocated string region, Error is returned.
alcstr(sbuf, i)	Copies the string of length i in sbuf to the allocated string region, returning NULL if the requested amount of space is not available.
GetReal(dp, r)	Places the floating-point double from the descriptor pointed to by dp into r.

Conversion between Icon's structure values and C structs is more complicated and must be handled on a case-by-case basis.

There are several global descriptors that may be useful in external functions:

nulldesc	descriptor for the null value
zerodesc	descriptor for the Icon integer 0
onedesc	descriptor for the Icon integer 1
emptystr	descriptor for the empty string

See runtime/data.r for others.

5. Acknowledgements

The external function facilities described in Section 2 were were based on ones written by Bill Griswold, using earlier work of Andy Heron. The original implementation for Version 8.0 of Icon was done by Sandra Miller and the first author. Some of the material in this report was adapted from implementation notes provided by Bill Griswold.

References

1. R. E. Griswold, C. L. Jeffery and G. M. Townsend, *Version 9.1 of the Icon Programming Language*, The Univ. of Arizona Icon Project Document IPD267, 1995.
2. R. E. Griswold and M. T. Griswold, *The Implementation of the Icon Programming Language*, Princeton University Press, 1986.
3. R. E. Griswold, *Supplementary Information for the Implementation of Version 8 of Icon*, The Univ. of Arizona Icon Project Document IPD112, 1995.
4. R. E. Griswold, *Supplementary Information for the Implementation of Version 9 of Icon*, The Univ. of Arizona Icon Project Document IPD239, 1995.
5. K. Walker, *The Run-Time Implementation Language for Icon*, The Univ. of Arizona Icon Project Document IPD261, 1994.
6. R. E. Griswold and M. T. Griswold, *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1990.

Appendix 1 — External Function Examples

Example 1: Functions Designated by Numbers

```
#if !COMPILER
#ifdef ExternalFunctions
/*
 * Example of calling C functions by integer codes.  Here it's
 *  one of three UNIX functions:
 *
 *     1: getpid (get process identification)
 *     2: getppid (get parent process identification)
 *     3: getpgrp (get process group)
 */

struct descrip retval;                          /* for returned value */

dptr extcall(dargv, argc, ip)
dptr dargv;
int argc;
int *ip;
   {
   int retcode;
   int getpid(), getppid(), getpgrp();

   if (!cnv_int(dargv, dargv)) {                /* 1st argument must be a string */
      *ip = 101;                                /* "integer expected" error number */
      return dargv;                             /* return offending value */
      }

   switch ((int)IntVal(*dargv)) {
      case 1:                                   /* getpid */
         retcode = getpid();
         break;

      case 2:                                   /* getppid */
         retcode = getppid();
         break;

      case 3:                                   /* getpgrp */
         if (argc < 2) {
            *ip = 205;                          /* no error number fits, really */
            return NULL;                        /* no offending value */
            }
         dargv++;                               /* get to next value */
         if (!cnv_int(dargv, dargv)) {          /* 2nd argument must be integer */
            *ip = 101;                          /* "integer expected" error number */
            return dargv;
            }
         retcode = getpgrp(IntVal(*dargv));
         break;
```

```
        default:
            *ip = 216;                          /* external function not found */
            return NULL;
        }

    MakeInt(retcode, &retval);                  /* make an Icon integer for result */
    return &retval;
    }
#else ExternalFunctions
static char x;                                  /* prevent empty module */
#endif                                          /* ExternalFunctions */
#endif                                          /* COMPILER */
```

Example 2: Functions Designated by Name

```
#if !COMPILER
#ifdef ExternalFunctions
/*
 * Example of calling C functions by their names.  Here it's just
 *  chdir (change directory) or getwd (get path of current working directory).
 */

struct descrip retval;                          /* for returned value */

dptr extcall(dargv, argc, ip)
dptr dargv;
int argc;
int *ip;
    {
    int len, retcode;
    int chdir(), getwd();
    char sbuf[MaxCvtLen];

    *ip = -1;                                   /* anticipate error-free execution */

    if (!cnv_str(dargv, dargv)) {               /* 1st argument must be a string */
        *ip = 103;                              /* "string expected" error number */
        return dargv;                           /* return offending value */
        }
```

```
    if (strncmp("chdir", StrLoc(*dargv), StrLen(*dargv)) == 0) {
        if (argc < 2) {                         /* must be a 2nd argument */
            *ip = 103;                          /* no error number fits, really */
            return NULL;                        /* no offending value */
            }
        dargv++;                                /* get to next argument */
        if (!cnv_str(dargv, dargv)) {           /* 2nd argument must be a string */
            *ip = 103;                          /* "string expected" error number */
            return dargv;                       /* return offending value */
            }
        qtos(dargv,sbuf);                       /* get C-style string in sbuf2 */
        retcode = chdir(sbuf);                  /* try to change directory */
        if (retcode == -1)                      /* see if chdir failed */
            return (dptr)NULL;                  /* signal failure */
        return &zerodesc;                       /* not a very useful result */
        }
    else if (strncmp("getwd", StrLoc(*dargv), StrLen(*dargv)) == 0) {
        dargv++;                                /* get to next argument */
        retcode = getwd(sbuf);                  /* get current working directory */
        if (retcode == 0)                       /* see if getwd failed */
            return NULL;                        /* signal failure */
        len = strlen(sbuf);                     /* length of resulting string */
        StrLoc(retval) = alcstr(sbuf,len);  /* allocate and copy the string */
        if (StrLoc(retval) == NULL) {           /* allocation may fail */
            *ip = 0;
            return (dptr)NULL;                  /* no offending value */
            }
        StrLen(retval) = len;
        return &retval;                         /* return a pointer to the qualifier */
        }
    else {
        *ip = 216;                              /* name is not one of those supported here */
        return dargv;                           /* return pointer to offending value */
        }
    }
#else                                           /* ExternalFunctions */
static char x;                                  /* avoid empty module */
#endif                                          /* ExternalFunctions */
#endif                                          /* !COMPILER */
```

Appendix 2 — Dynamic Loading Example

Icon Program

```
#  Demonstrate dynamic loading of bitcount() function

global bitcount

procedure main()
   local i

   bitcount := loadfunc("./lib.so", "bitcount")
   every i := 500 to 520 do
      write(i, "   ", bitcount(i))
end
```

C Function

```
/*
 *  bitcount(i) -- count the bits in an integer
 */

#include "rt.h"

int bitcount(argc, argv)
int argc;
struct descrip *argv;
   {
   struct descrip d;
   unsigned long v;
   int n;

   if (argc < 1)
      return 101;                          /* integer expected */

   if (!cnv_int(&argv[1], &d)) {
      argv[0] = argv[1];                   /* offending value */
      return 101;                          /* integer expected */
      }

   v = IntVal(argv[1]);                    /* get value as unsigned long */
   n = 0;
   while (v != 0) {                        /* while more bits to count */
      n += v & 1;                          /*    check low-order bit */
      v >>= 1;                             /*    shift off with zero-fill */
      }

   MakeInt(n, &argv[0]);                   /* construct result integer */
   return 0;                               /* success */
   }
```