# Icon-C Interfaces*

*Ralph E. Griswold*

TR 90-8b

January 1, 1990; last revised March 11, 1990

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grant CCR-8713690.

# Icon-C Interfaces

## 1. Introduction

Version 8 of Icon [1] supports two complementary features for calling C functions from Icon and vice versa. The two facilities are independent, but they may be used in conjunction and recursively.

In their simplest form, these facilities 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, 3].

## 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). The method of specifying C functions varies with system and application. 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. It should be replaced by an appropriate C function. Alternatively, the Icon function callout() can be modified to avoid the intermediate function call.

### Designating C Functions

A simple mechanism for designating C functions is to associate an integer with each one that can be called and use a C switch statement in extcall() to select the desired one. This method is used in the first example in Appendix A. A better method is to use string names, as illustrated by the second function in Appendix A. On most systems, all the C functions to be called must be linked with Icon (presumably through references in extcall()). On a system like OS/2 that supports run-time dynamic linking, C functions can be loaded as needed during program execution.

### Data Interface

The data interface also has to be handled by extcall() (or its equivalent). Arguments provided by Icon are in its descriptor format. Icon contains conversion facilities in its repertoire of macros and utility functions. Some that may be useful in external functions are:

| | |
|---|---|
| cvint(dp) | Converts the value in the descriptor pointed to by dp to an integer, returning CvtFail if the conversion cannot be performed. |
| 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. |
| cvstr(dp,sbuf) | Converts the value in the descriptor pointed to by dp to a string in sbuf, returning CvtFail if the conversion cannot be performed. |
| Qual(d) | Tests if d is a descriptor for a string. |
| 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. |
| strreq(i) | Requests i characters of space in the allocated string region, returning Error if the space is not available. |
| alcstr(sbuf,i) | Copies the string of length i in sbuf to the allocated string region. |
| blkreq(i) | Requests i bytes of space in the allocated block region, returning Error if the space is not available. |
| cvreal(dp) | Converts the value in the descriptor pointed to by dp to a real number (floating-point double), returning CvtFail if the conversion fails. |
| makereal(r,dp) | Constructs a real-number block for r and places a pointer to it in the descriptor pointed to by dp). |
| 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 iconx/idata.c for others.

**Error Handling**

The 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 [1]. Error 216 should be used if the designated C function is not found.

In some cases the error number is set by a utility routine (strreq() is an example). In such cases, the status code should be set to zero. 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 A.

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. Calling Icon from a C Program

The C function icon_call(), which is contained in Icon, is the complement of the Icon function callout(). The prototype for icon_call() is

```
dptr icon_call(char *id, int nargs, dptr argv)
```

where id is the string name of a procedure in the Icon program to be run and nargs is the number of descriptors in the array argv. The procedure is called with the specified arguments. The value returned is a pointer to the descriptor produced by the procedure if it returns or suspends, or NULL if the procedure fails. The global variable call_error is set to a nonzero value if the procedure is not found. See Appendix B for examples.

Before icon_call() is called the first time, Icon must be initialized by calling icont_init(prog), where prog is the name of the icode file to be run. This loads the named icode file, sets up Icon's storage regions, and readies Icon for execution. Subsequently, icon_call() can be called repeatedly.

## 4. Compiling Icon for C Calling

External functions (callout()) normally are enabled when Icon is compiled. They can be disabled by adding

```
#define NoExternalFunctions
```

to define.h and recompiling.

The ability to call an Icon program from C normally is disabled when Icon is compiled. It can be enabled by adding

```
#define IconCalling
```

to define.h and recompiling. Since the ability to call an Icon program from C increases the overhead of calling C functions from Icon (to support possible recursion), the ability to call an Icon program from C should not be enabled unless it is needed.

To add external functions to Icon, it is only necessary to write the appropriate code, place it in a file named extcall.c to replace the distributed stub, and to link Icon with its object module in place of the one for the stub.

To call Icon from a C program, it is necessary to provide the C program and use its object module in place of the one for istart.c, which is used by default (see the second example in Appendix B). It is necessary to link the entire Icon run-time system with the calling program. The resulting executable file is quite large.

## 5. Bugs

There presently is no mechanism for resuming a procedure that suspends as the result of icon_call().

A procedure called by icon_call() suspends by calling the Icon interpreter. There is no mechanism for unwinding the system stack in such a situation.

## 6. Acknowledgements

The facilities described here were based on ones written by Bill Griswold, using earlier work of Andy Heron. The implementation for Version 8 of Icon was done by Sandra Miller and the author. Some of the material in this report was adapted from implementation notes provided by Bill Griswold.

## References

1. R. E. Griswold, *Version 8 of Icon*, The Univ. of Arizona Tech. Rep. 90-1, 1990.
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, 1990.

## Appendix A — Examples of External Functions

### Example 1: Functions Designated by Numbers

```
/*
 * 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)
 */

#include "../h/config.h"
#include "../h/rt.h"
#include "rproto.h"

struct descrip retval;                          /* for returned value */

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

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

   if (cvint(dargv) == CvtFail) {               /* 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 (cvint(dargv) == CvtFail) { /* 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;
    }
```

**Functions Designated by Name**

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

#include "../h/config.h"
#include "../h/rt.h"
#include "rproto.h"

struct descrip retval;                                  /* for returned value */

dptr extcall(dargv, argc, ip)
dptr dargv;
int argc;
int *ip;
    {
    int len, retcode;
    char sbuf1[MaxCvtLen];                              /* for conversion on non–strings */
    char sbuf2[MaxCvtLen];                              /* for C–style string */
    int chdir(), getwd();

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

    if (cvstr(dargv, sbuf1) == CvtFail) {   /* 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 offedning value */
            }
        dargv++;                                        /* get to next argument */
        if (cvstr(dargv, sbuf1) == CvtFail) {   /* 2nd argument must be a string */
            *ip = 103;                                  /* "string expected" error number */
            return dargv;                               /* return offending value */
            }
        qtos(dargv,sbuf2);                              /* get C–style string in sbuf2 */
        retcode = chdir(sbuf2);                         /* 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(sbuf2);                       /* get current working directory */
   if (retcode == 0)                             /* see if getwd failed */
      return NULL;                               /* signal failure */
   len = strlen(sbuf2);                          /* length of resulting string */
   if (strreq(len) == Error) {                   /* need to allocate a copy of result */
      *ip = 0;                                   /* zero since code is set elsewhere */
      return (dptr)NULL;                         /* no offending value */
      }
   StrLoc(retval) = alcstr(sbuf2,len);  /* allocate and copy the string */
   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 */
   }
}
```

## Appendix B — Examples of Calling Icon

### Example 1: Calling Icon Procedures from the Command Line

```
/*
 *  Demonstration program to call an Icon procedure with arguments.  This
 *  program is used as
 *
 *         iconval iprog proc arg1 arg2 ...
 *
 *  where iprog is the name of the Icon icode file, proc is the name of
 *  a procedure in it, and arg1, arg2, ... are arguments passed to proc.
 *  It prints out the result if proc succeeds or notes if the procedure fails.
 *  It prints a diagnostic message if proc is not a procedure in iprog.
 */

#include "../h/config.h"
#include "../h/rt.h"
#include "rproto.h"

extern int call_error;

novalue main(argc,argv)

int argc;
char *argv[];
   {
   int clargc;
   char **clargv;
   dptr retval, iargv;
   int i;
   char sbuf[MaxCvtLen];

   /*
    * Read in the icode file argv[1] and initialize the Icon system.
    *  This must be done for any C program calling Icon.
    */
   icon_init(argv[1]);

   /*
    * Skip the names of the executable and the file it processes.  It
    *  is only necessary to get the the procedure name and its arguments from
    *  the command line.
    */
   clargv = argv + 2;
   clargc = argc - 3;
```

```
    fprintf(stderr,"program=%s0,∗clargv);
    fflush(stderr);
    /∗
     ∗ Malloc space for the list of descriptors and create Icon qualifiers
     ∗  for each argument.
     ∗/
    iargv = (dptr)malloc(clargc ∗ sizeof(struct descrip));
    for (i = 0; i < clargc; i++) {
        StrLoc(iargv[i]) = clargv[i + 1];
        StrLen(iargv[i]) = strlen(clargv[i + 1]);
       }
    retval = icon_call(∗clargv, clargc, iargv);
    if (call_error) {
        fprintf(stderr,"procedure not found0);
        fflush(stderr);
        c_exit(ErrorExit);
        }
    if (retval == NULL)
        fprintf(stdout,"evaluation failed0);
    else {
        /∗ Check type of result returned.  Don′t attempt to print anything
         ∗  but strings and integers here.
         ∗/
        if (Qual(∗retval)) {
          qtos(retval,sbuf);
          fprintf(stdout,"
          }
        else if (Type(∗retval) == T_Integer)
          fprintf(stdout,"%ld0,IntVal(∗retval));
        else
          fprintf(stdout,"type=%d0,Type(∗retval));
        fflush(stdout);
        }
    c_exit(NormalExit);

    }
```

**Example 2: Main Program for Calling Icon**

```
/∗
 ∗  Main program if Icon is called as a subprogram.
 ∗/

#include "../h/config.h"
#include "../h/rt.h"
#include "rproto.h"

#ifdef IconCalling

novalue main(argc,argv)
```

```
int argc;
char *argv[];
   {
   int clargc;
   char **clargv;
   int i;
   struct descrip darg;

   /*
    * Set up standard Icon interface.  This is only necessary so that
    *  Icon can behave normally as if it were the main program.
    *  It is not necessary if Icon is called by a C program for another
    *  purpose.
    */

#if VMS
   redirect(&argc, argv, 0);
#endif                                          /* VMS */

   icon_setup(argc, argv, &i);
   while (i--) {                                /* skip option arguments */
      argc--;
      argv++;
      }

   if (!argc)
      error("no icode file specified");

   /*
    * Read in the icode file argv[1] and initialize the Icon system.
    *  This must be done for any C program calling Icon.
    */
   icon_init(argv[1]);

   /*
    * Skip the names of the executable and the file it processes.  This
    *  is necessary only to get the right arguments from the command line
    *  to call Icon as if it were the main program and hence provide
    *  the correct values in the list that is the argument of Icon's main
    *  procedure. This is not necessary if Icon is called from C for
    *  another purpose.
    */
   clargv = argv + 2;
   clargc = argc - 2;

   /*
    * Set up a temporary stack and build the necessary list
    *  to call main.
    */
   sp = stack + Wsizeof(struct b_coexpr);
```

```
    PushNull;
    argp = (dptr)(sp - 1);
    for (i = 0; i < clargc; i++) {
        PushAVal(strlen(clargv[i]));
        PushVal(clargv[i]);
        }
    Ollist(clargc, argp);

    /*
     * Now that the list is computed, copy its descriptor off the
     *  stack (which is about to be destroyed), reset the argument
     *  pointer, and make the call to the Icon main procedure.
     */

    darg = *argp;
    argp = 0;
    icon_call("main", 1, &darg);                 /* return signal and value ignored */
    c_exit(NormalExit);

    }
#else                                            /* IconCalling */
static char x;                                   /* avoid empty module */
#endif                                           /* IconCalling */
```