INTERSCI

A Scilab interfacing tool

Scilab Group

INRIA Meta2 Project/ENPC Cergrene

INRIA - Unité de recherche de Rocquencourt - Projet Meta2
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
E-mail : scilab@inria.fr
Home page : http://www-rocq.inria.fr/scilab

Contents

Intersci is a program for building an interface file between Scilab and Fortran subroutines or C functions. This interface describes both the routine called and the associated Scilab function. The interface is automatically

1 Interface description files

To use Intersci one has first to write an interface description file.

The file <*interface name*>`.desc` is a sequence of descriptions of pairs formed by the Scilab function and the corresponding C or Fortran procedure (see table 1). In the following, we will essentially consider Fortran subroutine interfacing. The process is nearly the same for C functions (see 1.8).

<*Scilab function name*> <*function arguments*>
<*Scilab variable*> <*Scilab type*> <*possible arguments*>
⋮ ⋮ ⋮ ⋮ ⋮
<*Fortran subroutine name*> <*subroutine arguments*>
<*Fortran argument*> <*Fortran type*>
⋮ ⋮ ⋮ ⋮
out <*type*> <*formal output names*>
<*formal output name*> <*variable*>
⋮ ⋮ ⋮ ⋮
******

Table 1: Description of a pair of Scilab function and Fortran subroutine

Each description is made of three parts:

- description of Scilab function and its arguments
- description of Fortran subroutine and its arguments
- description of the output of Scilab function.

1.1 Description of Scilab function

The first line of the description is composed by the name of the Scilab function followed by its input arguments.

The next lines describe Scilab variables: the input arguments and the outputs of the Scilab function, together with the arguments of the Fortran subprogram with type `work` (for which memory must be allocated). It is an error not to describe such arguments.

The description of a Scilab variable begins by its name, then its type followed by possible informations depending on the type.

Types of Scilab variables are:

any any type: only used for an input argument of Scilab function.

column column vector: must be followed by its dimension.

list list: must be followed by the name of the list, <*list name*>. This name must correspond to a file <*list name*>`.list` which describes the structure of the list (see 1.7).

matrix matrix: must be followed by its two dimensions.

imatrix complex matrix: must be followed by its two dimensions.

bmatrix boolean matrix: must be followed by its two dimensions.

polynom polynomial: must be followed by its dimension (size) and the name of the unknown.

row row vector: must be followed by its dimension.

scalar scalar.

string character string: must be followed by its dimension (length).

vector row or column vector: must be followed by its dimension.

sparse sparse matrix: must be followed by its dimensions, the number of non zero elements and its real/complex flag

work working array: must be followed by its dimension. It must not correspond to an input argument or to the output of the Scilab function.

A blank line and only one ends this description.

1.2 Optional input arguments

Optional arguments are defined as follows:

- `[c val]` . This means that `c` is an optional argument with default value `val`. `val` can be a scalar: e.g. `[c 10]`, an array: e.g. `[c (4)/1,2,3,4/]` or a chain: e.g. `[c pipo]`
- `{b xx}`. This means that `b` is an optional argument. If not found, one looks for `xx` in current existing Scialb variables.

1.3 Description of Fortran subroutine

1.3.1 Fortran calling sequence description

The first line of the description is composed by the name of the Fortran subroutine followed by its arguments.

1.3.2 Fortran variables description

The next lines describe Fortran variables: the arguments of the Fortran subroutine. The description of a Fortran variable is made of its name and its type. Most Fortran variables correspond to Scilab variables (except for dimensions, see 1.5) and must have the same name as the corresponding Scilab variable.

Types of Fortran variables are:

char character array.

double double precision variable.

int integer variable.

real real variable.

Other types types also exist, that are called “external” types see 1.6.

A blank line and only one ends this description.

1.4 Description of the output of Scilab function

The first line of this description must begin by the word `out` followed by the type of Scilab output.

Types of output are:

empty the Scilab function returns nothing.

list a Scilab list: must be followed by the names of Scilab variables which form the list.

sequence a Scilab sequence: must be followed by the names of Scilab variables elements of the sequence. This is the usual case.

This first line must be followed by other lines corresponding to output type conversion. This is the case when an output variable is also an input variable with different Scilab type: for instance an input column vector becomes an output row vector. The line which describes this conversion begins by the name of Scilab output variable followed by the name of the corresponding Scilab input variable. See 3.3 as an example.

A line beginning with a star “`*`” ends the description of a pair of Scilab function and Fortran subroutine. This line is compulsory even if it is the end of the file. Do not forget to end the file by a carriage return.

1.5 Dimensions of non scalar variables

When defining non scalar Scilab variables (vectors, matrices, polynomials and character strings) dimensions must be given. There are a few ways to do that:

- It is possible to give the dimension as an integer (see 3.1).
- The dimension can be the dimension of an input argument of Scilab function. This dimension is then denoted by a formal name (see 3.2).
- The dimension can be defined as an output of the Fortran subroutine. This means that the memory for the corresponding variable is allocated by the Fortran subroutine. The corresponding Fortran variable must necessary have an external type (see 1.6 and 3.3).

Intersci is not able to treat the case where the dimension is an algebraic expression of other dimensions. A Scilab variable corresponding to this value must defined.

1.6 Fortran variables with external type

External types are used when the dimension of the Fortran variable is unknown when calling the Fortran subroutine and when its memory size is allocated in this subroutine. This dimension must be an output of the Fortran subroutine. In fact, this will typically happen when we want to interface a C function in which memory is dynamically allocated.

Existing external types:

cchar character string allocated by a C function to be copied into the corresponding Scilab variable.

ccharf the same as `cchar` but the C character string is freed after the copy.

cdouble C double array allocated by a C function to be copied into the corresponding Scilab variable.

cdoublef the same as `cdouble` but the C double array is freed after the copy.

cint C integer array allocated by a C function to be copied into the corresponding Scilab variable.

cintf the same as `cint` but the C integer array is freed after the copy.

csparse C sparse matrix allocated by a C function, to be copied into the corresponding Scilab variable. the copy.

csparsef the same as `csparse` but the C sparse array is freed after the copy.

In fact, the name of an external type corresponds to the name of a C function. This C function has three arguments: the dimension of the variable, an input pointer and an output pointer.

For instance, below is the code for external type `cintf`:

```
#include "../machine.h"

/* ip is a pointer to a Fortran variable coming from SCILAB
which is itself a pointer to an array of n integers typically
coming from a C function
   cintf converts this integer array into a double array in op
   moreover, pointer ip is freed */

void C2F(cintf)(n,ip,op)
int *n;
int *ip[];
double *op;
{
  int i;
  for (i = 0; i < *n; i++)
    op[i]=(double)(*ip)[i];
  free((char *)(*ip));
}
```

For the meaning of `#include "../machine.h"` and `C2F` see 1.8.

Then, the user can create its own external types by creating its own C functions with the same arguments. Intersci will generate the call of the function.

1.7 Using lists as input Scilab variables

An input argument of the Scilab function can be a Scilab list. If <*list name*> is the name of this variable, a file called <*list name*>`.list` must describe the structure of the list. This file permits to associate a Scilab variable to each element of the list by defining its name and its Scilab type. The variables are described in order into the file as described by table 2.

<comment on the variable element of the list> *<name of the variable element of list> <type> <possible arguments>* **********

Table 2: Description of a variable element of a list

Then, such a variable element of the list, in the file *<interface name>*`.desc` is referred to as its name followed by the name of the corresponding list in parenthesis. For instance, `la1(g)` denotes the variable named `la1` element of the list named `g`.

An example is shown in 3.4.

1.8 C functions interfacing

The C function must be considered as a procedure i.e. its type must be `void` or the returned value must not be used.

The arguments of the C function must be considered as Fortran arguments i.e. they must be only pointers.

Moreover, the name of the C function must be recognized by Fortran. For that, the include file `machine.h` located in the directory *<Scilab directory>*`/routines` should be included in C functions and the macro `C2F` should be used.

2 Writing compatible code

2.1 Messages and Error Messages

To write messages in the Scilab main window, user must call the `out` Fortran routine or `cout` C procedure with the character string of the desired message as input argument.

To return an error flag of an interfaced routine user must call the `erro` Fortran routine or `cerro` C procedure with the character string of the desired message as input argument. This call will produce the edition of the message in the Scilab main window and the error exit of Scilab associated function.

2.2 Input and output

To open files in Fortran, it is highly recommended to use the Scilabroutine `clunit`. If the interfaced routine uses the Fortran `open` instruction, logical units must in any case be greater than 40.

```
call clunit( lunit, file, mode)
```

with:

- `file` the file name `character string`
- `mode` a two integer vector defining the opening mode `mode(2)` defines the record length for a direct access file if positive. `mode(1)` is an integer formed with three digits `f`, `a` and `s`
  - `f` defines if file is formatted (`0`) or not (`1`)
  - `a` defines if file has sequential (`0`) or direct access (`1`)

- `s` defines if file status must be new (`0`), old (`1`), scratch (`2`) or unknown (`3`)

Files opened by a call to `clunit` must be close by

```
call clunit( -lunit, file, mode)
```

In this case the `file` and `mode` arguments are not referenced.

3 Examples

3.1 Example 1

The Scilab function is `a=calc(str)`. Its input is a string and its output is a scalar.
The corresponding Fortran subroutine is

```
      subroutine fcalc(str,a,n)
      character*(*) str
      integer a
c
c
      if (str(1:n).eq.'one') then
         a=1
      elseif (str(1:n).eq.'two') then
         a=2
      else
         a=-1
      endif
      end
```

Its arguments are a string `str` (used as input) and an integer `a` (used as output) and the string dimension. This last argument is useful if the subroutine has to be called by a C program. The description file is the following:

```
calc    str
str     string  n
a scalar

fcalc   str     a n
str     char
a       integer
n integer

out sequence a
**********************
```

3.2 Example 2

The name of the Scilab function is `c=som(a,b)`. Its two inputs are row vectors and its output is a column vector.
The corresponding C procedure is:

```
int csom_(n, a, b, c)
     int *n;
     float *a, *b, *c;
{
  int k;
  for (k = 0; k < *n; ++k)
      c[k] = a[k] + b[k];
  return(0);
}
```

. Its arguments are a real array with dimension n (used as input), another real array with dimension m (used as input) and a real array (used as output). These dimensions `m` and `n` are determined at the calling of the Scilab function and do not need to appear as Scilab variables.

Intersci will do the job to make the necessary conversions to transform the double precision (default in Scilab) row vector `a` into a real array and to transform the real array `c` into a double precision row vector.

The description file is the following:

```
som     a       b
a       row     m
b       row     n
c       column  n

csom    a       n       b       m       c
a       real
n       integer
b       real
m       integer
c       real

out     sequence        c
**********************
```

3.3 Example 3

The Scilab function is `[o,b]=ext(a)`. Its input is a matrix and its outputs are a matrix and a column vector.

The corresponding Fortran subroutine is `fext(a,m,n,b,p)` and its arguments are an integer array (used as input and output), its dimensions m,n (used as input) and another integer array and its dimension p (used as outputs).

The dimension `p` of the output `b` is computed by the Fortran subroutine and the memory for this variable is also allocated by the Fortran subroutine (perhaps by to a call to another C function). So the type of the variable is external and we choose `cintf`.

Moreover, the output `a` of the Scilab function is the same as the input but its type changes from a $m \times n$ matrix to a $n \times m$ matrix. This conversion is made my introducing the Scilab variable `o`

The description file is the following:

```
ext     a
a       matrix  m       n
```

```
b       column  p
o       matrix  n       m

fext    a       m       n       b       p
a       integer
m       integer
n       integer
b       cintf
p       integer

out     sequence        o       b
o       a
*******************************
```

3.4 Example 4

The name of the Scilab function is `contr`. Its input is a list representing a linear system given by its state representation and a tolerance. Its return is a scalar (for instance the dimension of the controllable subspace).

The name of the corresponding Fortran subroutine is `contr` and its arguments are the dimension of the state of the system (used as input), the number of inputs of the system (used as input), the state matrix of the system (used as input), the input matrix of the system (used as input), an integer giving the dimension of the controllable subspace (used as output), and the tolerance (used as input).

The description file is the following:

```
contr   sys     tol
tol     scalar
sys     list    lss
icontr  scalar

contr   nstate(sys)     nin(sys)     a(sys)  b(sys)  icontr  tol
a(sys)  double
b(sys)  double
tol     double
nstate(sys)     integer
nin(sys)        integer
icontr  integer

out     sequence        icontr
*****************************
```

The type of the list is `lss` and a file describing the list `lss.list` is needed. It is shown below:

```
1 type
type    string  3
*****************************
```

```
2 state matrix
a       matrix  nstate  nstate
*****************************
3 input matrix
b       matrix  nstate  nin
*****************************
4 output matrix
c       matrix  nout    nstate
*****************************
5 direct tranfer matrix
d       matrix  nout    nin
*****************************
6 initial state
x0      column  nstate
*****************************
7 time domain
t       any
*****************************
```

The number of the elements is not compulsory in the comment describing the elements of the list but is useful.

4 Building and using the interface

4.1 Calling intersci

To use Intersci execute the command:
`intersci-n` <*interface name*> where <*interface name*>`.desc` is the file describing the interface (see above).

The `intersci-n` script file are located in the directory SCIDIR/bin.

Using `intersci-n` two files are created : <*interface name*>.c and <*interface name*>_builder.sce are created. The file T<*interface name*>.c contains the interfacing procedures for each new Scilab function. The file <*interface name*>_builder.sce is a Scilab script which realizes the incremental linking of the gateway file. It has to be customized to set the `file` variable to the names of the object file associated to the gateway and the users objects and library files needed.

Example: Suppose that the descriptions given in the example 1 and 2 above are written down in the file `myex.desc`.
Running `intersci-n` one obtains:

```
$% intersci-n myex

INTERSCI Version 3.0 (SEP 2000)
    Copyright (C) INRIA/ENPC All rights reserved

*************************
processing SCILAB function "calc"
  generating C interface for function (fcalc) Scilab function "calc"
*************************
```

```
processing SCILAB function "som"
  generating C interface for function (fsom) Scilab function "som"
*************************
C file "myex.c" has been created
Scilab file "loader.sce" has been created

file "myex_builder.sce" has been created
```

The Scilab script `myex_builder.sce` contains:

```
// generated with intersci
ilib_name = 'libmyex'// interface library name

table =["calc","intscalc";
"som","intssom"];
ilib_build(ilib_name,table,files,libs);
```

Suppose that the procedures `fcalc` and `csom` are defined in the Fortran file `fcalc.f` and the C file `csom.c`
The file `ex01fi_builder.sce` has to be customized as follow:

```
// generated with intersci
ilib_name = 'libmyex';           // interface library name
files=['myex','fcalc','csom'] ;
libs=[]; //no libs required
table =["calc","intscalc";
        "som","intssom"];
ilib_build(ilib_name,table,files,libs);
```

4.2 Compiling and building library

This builder file is to be executed by Scilab:

```
-->exec myex_builder.sce;
   generate a gateway file
   generate a loader file
   generate a Makefile: Makelib
   running the makefile
```

The generated file lib<*interface name*>.c is the C gateway needed for interfacing all routines defined in <*interface*> name with Scilab.

A dynamic library is then created as well as a file `loader.sce`.

4.3 Loading in Scilab

Executing `loader.sce` loads the library into Scilab and executes the `addinter` command to link the library and associate the names of the functions to it. We can then call the new functions :

```
-->exec loader.sce;
Loading shared executable ./libmyex.so
shared archive loaded
Linking libmyex (in fact libmyex_)
Interface 0 libmyex
```

Of course this instruction can be written in one of the Scilab startup files not to have to re-enter it each time Scilab is started.

These new functions can then be used as the others:

```
-->a=[1,2,3];b=[4,5,6];
-->c=som(a,b)
 c  =

!   5. !
!   7. !
!   9. !
-->calc('one')
 ans  =

    1.
```

4.4 Adding a new interface to the Scilab kernel

It is possible to add a set a new built-in functions to Scilab by a permanent link the interface program. For that, it is necessary to update the files `default/fundef` and `routines/callinter.h`.

When `intersci` is invoked as follows:

`intersci` <*interface name*> <*interface number*>

`intersci` then builds a `.fundef` file which is used to update the `default/fundef` file.

To add a new interface the user needs also to update the `routines/callinter.h` file with a particular value of `fun` Fortran variable corresponding to the new interface number.

Two unused empty interface routines called by default (`matusr.f` and `matus2.f`) are predefined and may be replaced by the interface program. Their interface numbers `14` and `24` respectively. They can be used as default interface programs. The executable code of Scilab is then made by typing “make all” or “make bin/scilex” in Scilab directory.