Foreign Language Interfaces

Valerie Obenchain

Fred Hutchinson Cancer Research Center

17-18 February, 2011

Overview

- Motivation
- Interface functions
- Compile and load dynamic libraries
- Using .C()
- Register native routines

Motivation

- Areas where the R implementation is suboptimal :
  - sliding window algorithms
  - calculations that are difficult to vectorize
- Implement third party algorithms or libraries (e.g., GSL, BOOST, BGL, SAMtools, affxparser)

Interface Functions

The following functions provide a standard interface to compiled code that has been linked into *R*:

- .C
- .Call
- .Fortran

Advantages of .Call() vs .C()

- Less copying
- Memory allocation in C
- Pass and receive R objects instead of primitive types
- Access to the attributes of the vectors (i.e., names)
- Ability to handle missing values easily

C code

- Compiled code should not return anything except through the arguments
- C functions should be of type `void`

```
/* composite_linkage_disequilibrium.c */

void composite_linkage_disequilibrium(
          unsigned char *snp, /* matrix indiv x snp */
          int *n_ind,         /* # individuals */
          int *n_snp,         /* # snps */
          int *width,         /* adjacent snp window */
          double *delta)      /* result */
{
    ...
}
```

Compile and load dynamic libraries : *R* Session

- A shared object can be created with

  `R CMD SHLIB composite_linkage_disequlibrium.c.`

- From within an *R* session the shared object can be loaded with `dyn.load`. The functions in the compiled code are now available for use in the *R* session.

  ```
  > dyn.load("composite_linkage_disequilibrium.so")
  ```

Compile and load dynamic libraries : *R* Package

- Load with `useDynLib(mypkg)` in the NAMESPACE
- Other instructions can be put in `.onLoad` and `.onUnload` functions in a zzz.R file.

Using .C()

- The first argument is a character string of the C function name. The remainder of the arguments are *R* objects to be passed to the C function.
- Arguments should be coerced to the *R* storage mode that corresponds to the data type defined in the C code

```
## Create sample data
snps <- matrix(sample((1:3), replace=TRUE),
                nrow=10, ncol=4)
width <- 3
delta <- rep.int(0, (ncol(snps)-width)*width)
out <- .C("composite_linkage_disequilibrium",
          snp = as.raw(snps),
          n_ind = as.integer(nrow(snps)),
          n_snp = as.integer(ncol(snps)),
          width = as.integer(width),
          delta = as.double(delta))
```

Register Native Routines

Motivation :

- Platform-independent mechanism for finding routines in shared objects
- Information about a native routine made available within *R*

Steps :

- Create an initialization file called R_init_mypkg.c
- Create an array describing the function with R_CMethodDef
- Register the function with R_registerRoutines

Register Native Routines

- Create an array describing the C routine with R_CMethodDef:

```
typedef struct {
    const char *name; DL_FUNC fun; int numArgs;
    R_NativePrimitiveArgType *types;
    R_NativeArgStyle         *styles;
} R_CMethodDef;
```

- *R* types and corresponding type identifiers :

`numeric'	`REALSXP'
`integer'	`INTSXP'
`logical'	`LGLSXP'
`character'	`STRSXP'
`raw'	`RAWSXP'

Register Native Routines

- Given the original C function

```
void composite_linkage_disequilibrium(
          unsigned char *snp,
          int *n_ind,
          int *n_snp,
          int *width,
          double *delta)
```

- We create the R_CMethodDef array in R_init_mypkg.c

```
R_CMethodDef cMethods[] = {
    {"composite_linkage_disequilibrium",
     (DL_FUNC) &composite_linkage_disequilibrium, 5,
     {RAWSXP, INTSXP, INTSXP, INTSXP, REALSXP}
    },
    {NULL, NULL, 0}
};
```

Initialization Function

- The initialization file contains the R_CMethodDef array and the R_registerRoutines function wrapped in the R_init_mypkg function.

```
void R_init_mypkg(DllInfo *info)
{
    /* Create the R_CMethodDef array */
    R_CMethodDef cMethods[] = {
        {"composite_linkage_disequilibrium",
         (DL_FUNC) &composite_linkage_disequilibrium,
         5,{RAWSXP, INTSXP, INTSXP, INTSXP, REALSXP}
        },
        {NULL, NULL, 0}
    };

    /* Register the routine */
    R_registerRoutines(info, cMethods,
                       NULL, NULL, NULL);
}
```

Resources

- Writing *R* Extensions Manual
  Section 5: System and Foreign Language Interfaces
  http://www.r-project.org/

Wrap Up

- C code belongs in src directory of package
  - function type void
  - return results through arguments
- By including `useDynLib(mypkg)` in the NAMESPACE the R CMD INSTALL process compiles and links the C code into shared object
- Call C routine using .C interface using correct data types
- Register C method with R in the R_init_mypkg.c file
  - R_CMethodDef defines the C function
  - R_registerMethods registers routine we defined with R_CMethodDef
  - R_CMethodDef and R_registerMethods belong in R_init_mypkg function