Writing Dana native libraries

Dana native libraries allow you to call C code from Dana, as if that C code were itself a Dana component. This is useful when you need to interface directly with OS or hardware-level functionality in a way not provided by the Dana standard library.

The process to create a native library involves a Dana interface, a wrapper component implementing that interface and talking to the native library through an internal interface, and a native library which implements the internal interface.

We begin by defining a Dana interface for our wrapper component:

```
interface MyWrapper{
   void doSomething()
   }
```

Write the above code in a text file, save it as **MyWrapper.dn**, and place it somewhere in the Dana central source tree (in the resources directory tree).

Next we create the wrapper component which implements the above interface:

```
library interface MyLib{
   void libFunction()
   }

component provides MyWrapper, Service requires NativeLoader loader {
   static library MyLib myLib
   implementation MyWrapper {
      void MyWrapper:doSomething() {
         myLib.libFunction()
         }
      }

   implementation Service {
      void Service:start() {
         myLib = new MyLib() from loader.load("mylib") :< MyLib
         }

      void Service:stop() {
         }
      }
   }
```

We put this component in a symmetrical place to the MyWrapper.dn interface in the Dana central source tree, within the components directory tree.

Finally we create the native library itself.

Create a new file **mylib.c** in the libraries repository. In this file, write the following code:

```
#include "dana_lib_defs.h"
#include "nli_util.h"
#include "vmi_util.h"

#include <string.h>
#include <stdlib.h>
#include <stdio.h>

static CoreAPI *api;

INSTRUCTION_DEF op_my_function(INSTRUCTION_PARAM_LIST)
      {
      printf("A native call!");
      return RETURN_DIRECT;
      }

Interface* load(CoreAPI *capi)
      {
      api = capi;
      setInterfaceFunction("libFunction", op_my_function);
      return getPublicInterface();
      }

void unload()
      {
      }
```

Next we need to use the Dana compiler to generate a C file containing all of the glue code between our native library and the Dana runtime. To do this we open a command prompt in the same directory as our wrapper component (yes, the component, not the interface), and type:

```
dnc MyWrapper.dn -gni
```

Assuming that you saved your component in a file called **MyWrapper.dn**. This will create a new text file called **MyLib_dni.c**. Copy this file to the libraries repository (i.e. the same folder as your library code).

Now we create a makerule for this library in the Makefile of the libraries repository. Open Makefile in a text editor and add the rule:

```
my_new_lib:
      $(CC) -Os -s MyLib_dni.c vmi_util.c mylib.c -o
mylib[$(PLATFORM).$(CHIP)].dnl $(STD_INCLUDE) $(CCFLAGS)
      $(CP_CMD) mylib[$(PLATFORM).$(CHIP)].dnl
"$(DANA_HOME)/resources-ext"
```

And run the makerule with:

```
make my_new_lib
```

Now, if you write a quick test program to call your wrapper component's *doSomething()* function, you should see the text printed from the native library. Simple :-)

The creation of every native library follows the same procedure.

After that, for now, you're on your own! Use the existing libraries as a reference - they collectively do just about everything you might want to, so play around and figure things out, and ask questions when you get stuck. We'll write more documentation when we can!