Creating Shareable Models

By: Eric Hutton

CSDMS is the Community Surface Dynamics Modeling System

(pronounced ˈsɪstəms)

Image by Flickr user Let There Be More Light

A model can more easily be used by others if it is *readable* and has an *API*

Readable

Someone new to your code should be able to give it a quick read and know what's going on.

Application Programming Interface (API)

The interface will allow someone to use it without worrying about implementation details.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

But before all that though, choose a license and a language

The GNU General Public License is a good choice.

But there *lots* more. For example,

- MIT
- Apache
- LGPL
- OSL
- GPLv3

As far as languages, C is a good choice.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Writing readable code mostly means sticking to a standard

The code should speak for itself. That doesn't just mean adding more comments.

Super simple things help:

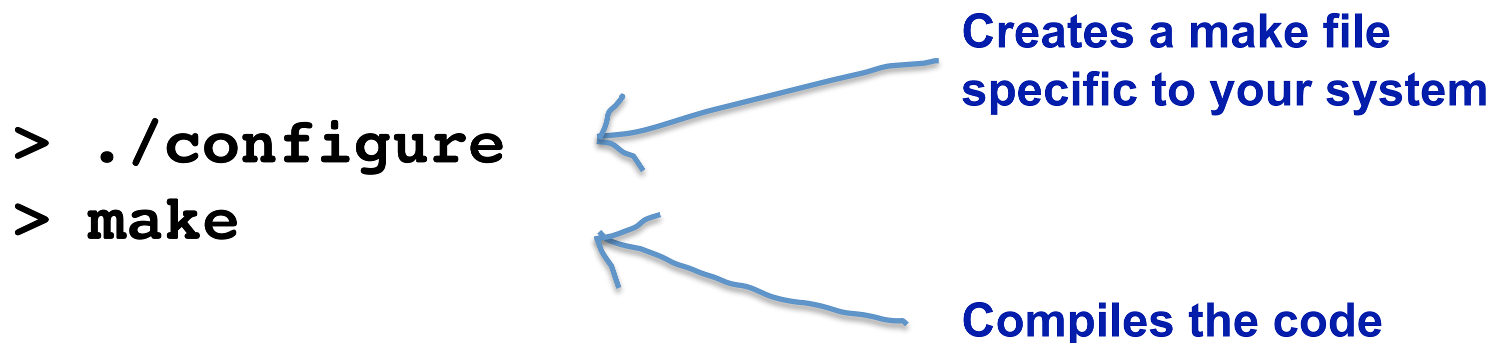
- Spaces instead of tabs
- Useful variable names – especially for global variables and function names
- Underscores in names – water_density is better than iCantReadThis

In the end though, just be *consistent*.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Makefiles and configure scripts help to make code portable

Software distributions on UNIX are usually compiled with,



Easy, right? Well… the magic that goes on behind the scenes to make this tick is *complicated*.

Uses "programming languages" such as: automake, autoconf, m4, make, sh

A better alternative to configure/make is cmake

Cmake is free, open-source software that replaces autotools/make.

- **Simple syntax**
- **Much easier to learn than autotools**
- **Supports nearly every Unix, Windows, Mac OSX**
- **Features a testing framework**
- **Features a packaging framework**
- **am2cmake script to convert automake files**

Random compile tip:

compile with –`Wall` compile flag and *pay attention* to the warnings.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

If your code comes with documentation, it is more likely to be used by others

There are tools that help with this

These all generate documentation from comments within your code.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

For example, a C function can be documented directly above it’s definition/declaration.

```
/** Calculate the area of a circle.

@param r Radius of the circle

@returns The circle’s area
*/
double area_of_circle ( double r );
```

This is similar to annotations in Java. We’ll come back to this later.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

It would be nice if there existed a set of standard annotations for our modeling community.

Some examples of annotation that would be useful to someone wanting to use your model:

- **@initialize, @run, @finalize**
- **@author, @keyword, @version**
- **@in, @out, @unit**

Your source code could then be parsed for these annotations.

Since they are ignored by the compiler, they inject very little of the framework into your code.

Annotations could be used to describe three categories of meta data (this list comes from OMS)

Component	Field	Method
@description	@description	@run
@author	@unit	@initialize
@bibliography	@in	@finalize
@status	@out	
@version	@range	
@source	@role	
@keywords	@bound	
@label	@label	

Others?

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Example annotation for a component that calculates the area of a circle

```
@description("Circle computation")
@author("me")
Public class CircleArea {
   @description("Radius")
   @range(min=0)
   @in public double r;
   @out public double area;

   @run
   public void runme() {
      area = Math.PI * r * r;
   }
}
```

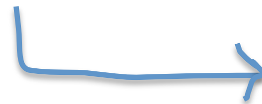
CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Our models are difficult to couple because of implementation details

Some reasons that models are difficult to link or share:

- Languages
- Time steps
- Numerical schemes
- Complexity

These are all implementation details. The user shouldn't have to worry about these details.

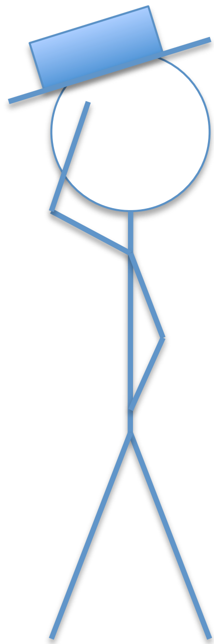


Design an interface!

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

A modeling interface hides implementation details from the application developer

You

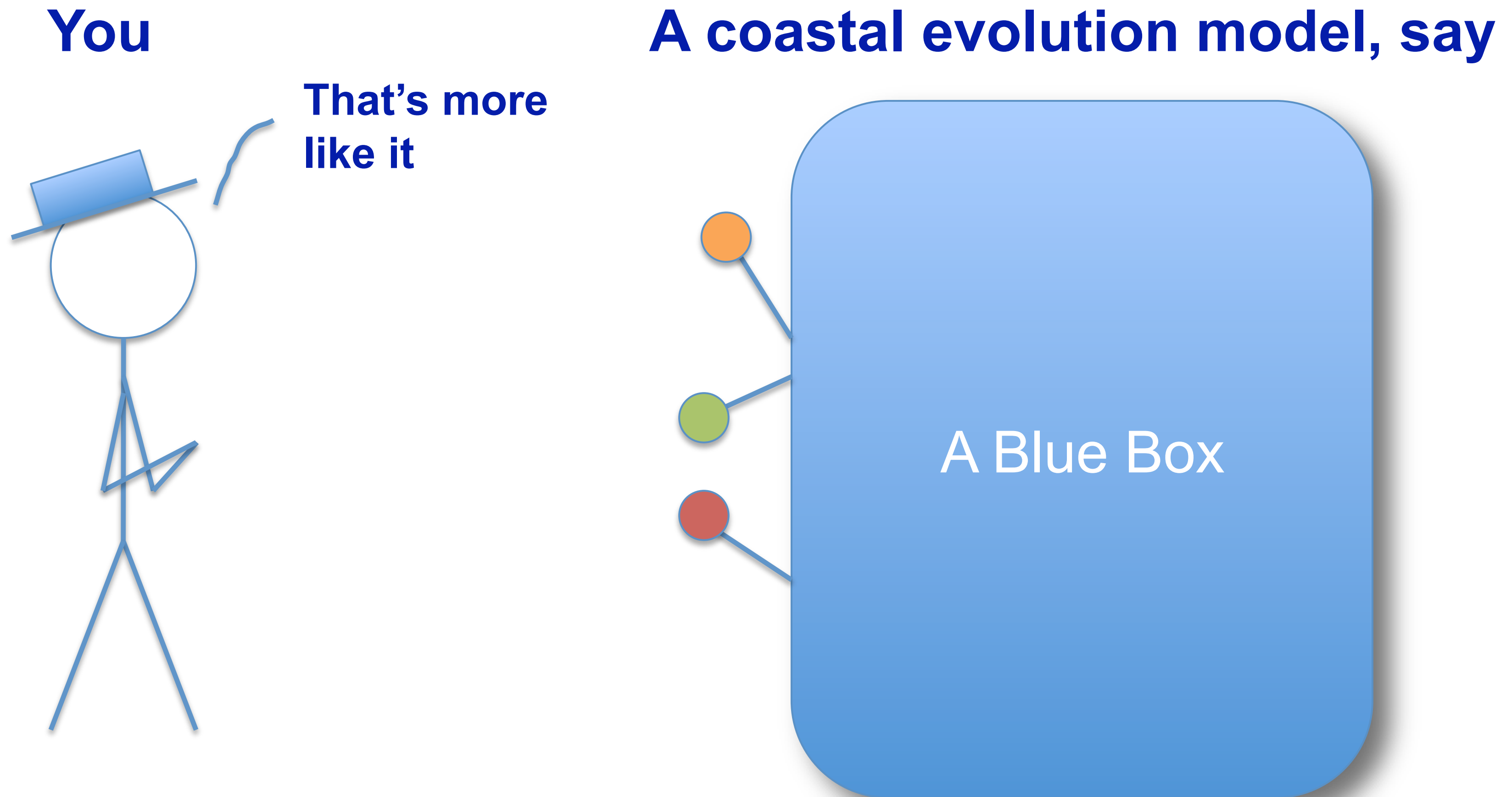


A coastal evolution model, say

Lots of crazy, hard-to-understand code

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

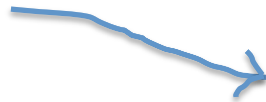
A modeling interface hides implementation details from the application developer



CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

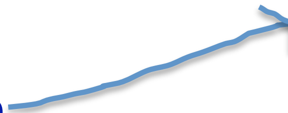
For our models a useful interface is IRF

I is for *Initialize*



R is for *Run*

F is for *Finalize*



```
int
main()
{
    initialize_stuff();
    run_until_stop_time();
    print_output();
    return SUCCESS;
}
```

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

When designing interfaces it is often easiest to start at the end

```
int
main()
{
    cem_state *s = cem_init ();
    double *d0, *d1;

    cem_run_until (s, 25.);
    d0 = cem_get_water_depth (s);

    cem_run_until (s, 25.);
    d1 = cem_get_water_depth (s);

    cem_finalize (s);
    return EXIT_SUCCESS;
}
```

Setup the model run

Run the model for 25 years

Get the water depths

Run the model for an additional 75 years and get water depths

Do whatever needs to be done before quitting

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

The initialize step sets up a new model run (instance)

Things that might be done within the initialize method:

- Create a new state for the model
- Allocate memory
- Set initial conditions
- Pretty much anything that is done before the time looping begins

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

The initialize step shouldn’t interfere with another program’s initialize step

Things that probably shouldn’t be done within the initialize method:

- Gather input from the command line (argc, argv)
- Most any user interface
- Anything that might interfere with another component

The finalize step destroys an existing model run (instance)

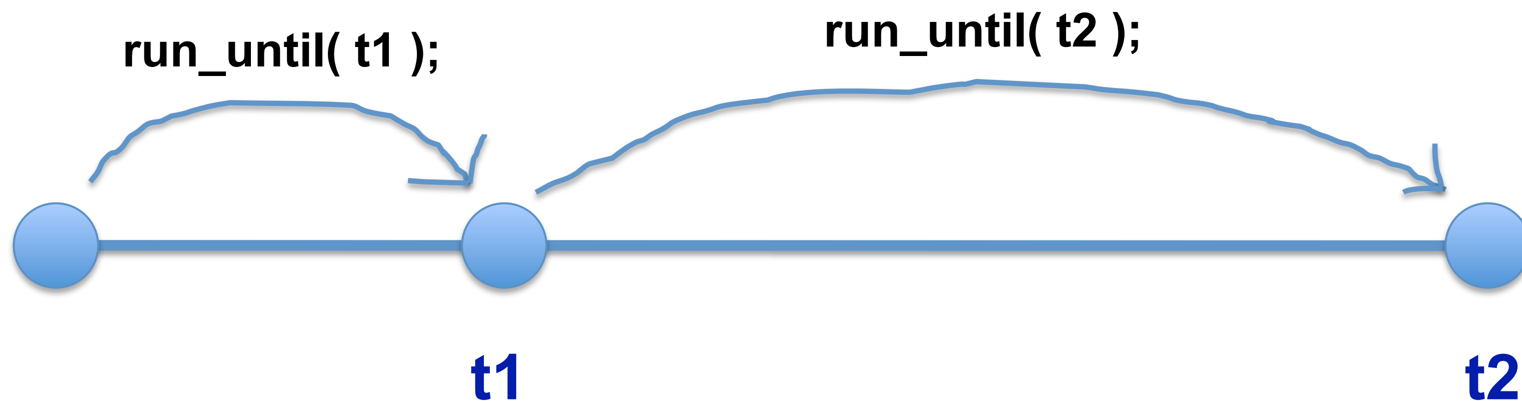
Things that might be done within the finalize method:

- Free resources
- Close files
- Print final output
- Pretty much anything that is done after the time looping ends

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

The run step advances the current state of the model forward in time

The run method advances the model from its current state to a future state.



Oftentimes it is useful to create a data type that holds the current state of the model

This is the hard part.

Identify all of the variables that need to be remembered to advance the model forward in time.

Collect these variables (I generally wrap them in a struct or class). I suppose a common block or a set of global variables would do.

This data structure can then be passed to the various methods of the API.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Getters and setters are used to interact with the model's state

The state structure should be opaque to the user – that is implementation stuff

Instead, use get and set functions. For instance,

```
get_water_depth( );

set_input_file( "input" );
```

It is up to the API designer to decide what variables are allowed to be get/set.

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Refactoring your code in this way should not change model output

…unless you find (and fix) a bug.

To make sure that you don’t add bugs when refactoring,

Make lots of tests

Run these tests often

Keep in mind that output may not be byte-for-byte comparable to your benchmarks

In conclusion, there are really only two "rules"

Write code that someone else can understand. Pick a standard and stick to it.

Create an API.

Leave implementation details to the experts.

Worry about your own model.

Questions?

CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM