



ContextTracking.jl



This is a **production-quality** package for keeping tracking of contextual information. The context data is accessible from all functions called from the current execution context, regardless of how deep the nested calls are. The library is thread-safe and usable from async processes.

The problem

The program contains 4 functions such that

- A calls B
- B calls C
- C calls D

Some data were gathered in A and they need to be available in D, **but maybe not** in B or C.

The old way...

1. Define a Context data type.
2. Create the context object in function A.
3. Add a new context argument in function B & C & D.
4. Pass the object down the chain from A to B to C to D.

ContextTracking!

1. Annotate the functions A, B, C, and D with **@ctx** macro.
2. Record context data using **@memo** macro in A.
3. Access data in function D using **context** function.



What happens during execution?



Code

```
@ctx function foo()  
# 1  
@memo x = 1  
# 2  
bar()  
# 5  
end  
  
@ctx function bar()  
# 3  
@memo y = 2  
# 4  
@info context().data  
end
```

Use **@ctx** macro to participate in context tracking.

Use **context()** function to access information.

Use **@memo** macro to remember contextual information.

What's the context?

Point #1 (foo)

Empty

Point #2 (foo)

:x => 1

Point #3 (bar)

:x => 1

Point #4 (bar)

:x => 1

:y => 2

Point #5 (foo)

:x => 1

The context *unwinds* as the call returns.