U170T0Y3H : How can I refer to a var inside ns1 when the macro defined in ns1 is called from ns2?```
(ns ns1)

(defn a-fn* [] "hello")

(defmacro a-marco []
  `(defn a-fn [] (a-fn*)))

(ns ns2)

(ns1/a-marco) ;=> Can't refer to qualified var that doesn't exist
```

U060FKQPN : that's going to work
U060FKQPN : I don't believe you see that error message on a fresh repl, you must have some stale state
U060FKQPN : also that macro is slightly wrong, should be ```(defmacro a-macro [] `(defn ~'a-fn [] (a-fn*)))```

U060FKQPN : needless to say macros like that are discouraged in clojure
U0JFCEH9P : I'm playing with an event sourcing/CQRS style system in Clojure. It involves some number of load-balanced app servers. My idea is to have local in-memory caches, and then "catch up" by applying pending events before any read operations.
U0JFCEH9P : I'm trying to avoid having any extra pieces like a message queue
U0JFCEH9P : does that seem sane?
U170T0Y3H : <@U060FKQPN> It worked, with the unquote-quote. But now I'm pretty discouraged.
U060FKQPN : generally, macros that inject global names into a namespace are not idiomatic in clojure
U060FKQPN : a slightly better version would be e.g. ``` (defmacro a-macro [name] `(defn ~name [] (a-fn*)))``` but still, this doesn't look like a very useful macro
U5UP845LY : they have their uses, probably finding out what is the goal is a step to take before judging something as not idiomatic
U060FKQPN : why would you intern a var in a namespace that just delegates to a var in another? just refer that one directly
U060FKQPN : coming off as judgmental was not my intention, if that's how it came across
U5XMV6DQT : ```#'ns1/a-marco
```

U170T0Y3H : delegating is part of if, passing some variables, but not all is the other half (something like partial, but on macro level)
U170T0Y3H : I couldn't use `partial` because what's passed might be an atom that needs to be derefed later.
U170T0Y3H : Needless to say my example was stripped down to the bare minimum to illustrate the problem I was having. :slightly_smiling_face:
U09LZR36F : Wondering what people do for translations in their application? The key based stuff puts me off (<https://translation.io/blog/gettext-is-better-than-rails-i18n>), and I'd like to use industry-accepted systems (PO, XLIFF, MessageFormat). Doing this for cljs & clj.
U0NBGRGD6 : How can I extend a type, like `java.util.HashMap` to support `conj`. Which is the protocol for that?
U060FKQPN : you can't in clojure
U060FKQPN : clojure implements its basic operations in terms of interfaces not protocols
U11BV7MTK : here's how it was done in the priority-map <https://github.com/clojure/data.priority-map/blob/master/src/main/clojure/clojure/data/priority_map.clj#L255>
U060FKQPN : right, you can do that if you're in control of new types but it's simply not possible to retrofit on existing types like `java.util.HashMap`