Organization Your Project

A Librarian's Tale

MELVIL DEWEY.

Metaphor

- Library -> Namespace
- Book name -> Symbol
- Card caralog -> Var
- Shelf address -> Memory space

Symbol

```
inc
; => #object[clojure.core$inc 0x4aa0a65c "clojure.core$inc@4aa0a65c"]
'inc
; => inc
```

Var

Storing Objects with def

```
(def great-books ["East of Eden" "The Glass Bead Game"]) ; => #'user/great-books
```

Reader form of a var

```
#'user/great-books
```

Interning a var

1. Update the current namespace's map with the association between great-books and the var.
2. Find a free storage shelf.
3. Store ["East of Eden" "The Glass Bead Game"] on the shelf.
4. Write the address of the shelf on the var.
5. Return the var (in this case, #'user/great-books).

Map of symbols-to-interned-vars

```
(ns-interns *ns*)
```

Deref the var

```
(deref #'user/great-books)
; => ["East of Eden" "The Glass Bead Game"]
```

Namespace

Current namespace

```
(ns-name *ns*) ; => user
```

Creating and switching namespace

```
(create-ns 'cheese.taxonomy) ; => #object[clojure.lang.Namespace 0x221c0ed3 "cheese.taxonomy"]
(in-ns 'cheese.taxonomy)
```

Use functions and data from other namespaces

Fully qualified symbol

```
(in-ns 'cheese.taxonomy)
(def cheddars ["mild" "medium" "strong" "sharp" "extra sharp"])
(in-ns 'cheese.analysis)

; cheddars => java.lang.RuntimeException: Unable to resolve symbol
; cheese.taxonomy/cheddars => ["mild" "medium" "strong" "sharp" "extra sharp"]
```

refer

```
(in-ns 'cheese.taxonomy)
(def cheddars ["mild" "medium" "strong" "sharp" "extra sharp"])
(def bries ["Wisconsin" "Somerset" "Brie de Meaux" "Brie de Melun"])
(in-ns 'cheese.analysis)
(clojure.core/refer 'cheese.taxonomy)

; bries => ["Wisconsin" "Somerset" "Brie de Meaux" "Brie de Melun"]
; cheddars => ["mild" "medium" "strong" "sharp" "extra sharp"]
```

refer

:only

```
(clojure.core/refer 'cheese.taxonomy :only ['bries])
; bries => ["Wisconsin" "Somerset" "Brie de Meaux" "Brie de Melun"]
; cheddars => java.lang.RuntimeException: Unable to resolve symbol: cheddars in this context
```

refer

:exclude

```
(clojure.core/refer 'cheese.taxonomy :exclude ['bries])
; bries => java.lang.RuntimeException: Unable to resolve symbol: bries in this context
; cheddars => ["mild" "medium" "strong" "sharp" "extra sharp"]
```

refer

:rename

```
(clojure.core/refer 'cheese.taxonomy :rename {'bries 'yummy-bries})
; bries => java.lang.RuntimeException: Unable to resolve symbol: bries in this context
; yummy-bries => ["Wisconsin" "Somerset" "Brie de Meaux" "Brie de Melun"]
```

alias

```
(clojure.core/alias 'taxonomy 'cheese.taxonomy)
; taxonomy/bries => ["Wisconsin" "Somerset" "Brie de Meaux" "Brie de Melun"]
```

Loading namespace in your project

Clojure doesn’t automatically evaluate it when it runs your project; you have to explicitly tell Clojure that you want to use it.

Loading namespace in your project

```
(require 'the-divine-cheese-code.visualization.svg)
; the_divine_cheese_code/visualization/svg.clj
```

require

:as

```
(require '[the-divine-cheese-code.visualization.svg :as svg])
```

is equivalent to this:

```
(require 'the-divine-cheese-code.visualization.svg)
(alias 'svg 'the-divine-cheese-code.visualization.svg)
```

use

```
(require 'the-divine-cheese-code.visualization.svg)
(refer 'the-divine-cheese-code.visualization.svg)
```

is equivalent to this:

```
(use 'the-divine-cheese-code.visualization.svg)
```

use

:as :only :exclude

```
(use 'the-divine-cheese-code.visualization.svg :as svg :only [points])
```

The ns Macro

- Create new namespace if needed.
- Refer **clojure.core** namespace by default.

Six possible kinds of references within ns

- (:refer-clojure)
- (:require)
- (:use)
- (:import)
- (:load)
- (:gen-class)

The ns macro (:require)

```
(ns the-divine-cheese-code.core
  (:require the-divine-cheese-code.visualization.svg))
```

is equivalent to this:

```
(in-ns 'the-divine-cheese-code.core)
(require 'the-divine-cheese-code.visualization.svg)
```

The ns macro
(:require) alias a library

```
(ns the-divine-cheese-code.core
  (:require [the-divine-cheese-code.visualization.svg :as svg]))
```

is equivalent to this:

```
(in-ns 'the-divine-cheese-code.core)
(require ['the-divine-cheese-code.visualization.svg :as 'svg])
```

One more thing...

Private function

```
(defn- private-function
  "Just an example function that does nothing"
  [])
```

References

- Ch6 of Clojure for the Brave and True
- Namespaces
- Vars and the Global Environment
- Intern - Clojure Terminology Guide
- Using Libs

THANK YOU