INTRO TO CLOJURE-CLR

CLOJURE FOR THE .NET DEVELOPER

WHAT IS CLOJURE?

- Derivative of Lisp Created by Rich Hickey that targets the JVM, CLR and JavaScript
- Code as data – (println “Hi, Richmond!”)
- ‘Variables’ are immutable by default
- Dynamic

ENVIRONMENT SETUP

1. Download from https://github.com/clojure/clojure-clr

2. Unzip to a directory of your choice

Update PATH variable to include directory mentioned above

Set CLOJURE_LOAD_PATH to the directory where clojure.main.exe lives.

VARIABLES AND FUNCTIONS

Declaring Variables:

- Namespace wide:
  - (def my-var “Value here”)
- Local:
  - (let [my-var “Value here”] <code here>)
  - Variables defined in let calls are scoped within the let parenthesis

Declaring functions:

- (defn my-func [param] <body of function>)
- (fn [p1 p2 p3] <body of function>)
- #(println “Hi from my anonymous fn”)
- Called by fully qualified name => namespace/function name

NAMESPACES

- **Defining a namespace: (ns db.queries)**
- **Namespaces are used to set scope for functions, variables, keywords, etc.**
- **To include a namespace you can**
  - Call the use function: (use <namespace>) brings all fn, defs into your namespace
  - Call the require makes them available but requires you to use qualified name
- **Map to the file system.**
  - (ns db.queries) indicates that the the queries.clj file lives in the db directory

COLLECTIONS

- **Vectors – collection of values indexed by continuous integers.**
  - (def my-vec [0 “One” 2.0])
- **Maps – Key/Value collection**
  - (def my-map {:one 1 :two “Some String”})
- **Sequences – a logical, immutable list. Most sequences are lazy.**
  - (def my-seq ‘(1 2 3 “4” 5.0 my-map))

PROCESSING COLLECTIONS

- **map – like a foreach loop processes a sequence item by item**

  (map #(println “processed %”) my-seq)
- **first and rest**

  (first my-seq) returns the first item in my-seq.

  (rest my-seq) returns a new sequence with all but the first item of the sequence
- **filter**

  (filter my-filter-fn my-seq) – filter is similar to map that it processes each item. All items that cause a true value to be returned by the testing fn are returned.

THE REPL

- **Read-Eval-Print Loop**
- **Start the reply by running %CLOJURE_LOAD_PATH% \clojure.main.exe**
- **REPL Demo**
    - Basics
        - Variables
        - Functions
    - Sequences and maps
        - Map
        - filter
    - .NET interaction
        - Instantiating a class
        - Using .Net assemlies
        - Get/Set Properties

THE HOCKEY PLAYER LOOKUP APPLICATION

BUILDING AN APPLICATION

- Connect to and query a Postgresql database using ADO.NET
- Create a Windows.Forms based User Interface
- Display the results of user's query
- Demo

THE DB LAYER - LOADING

Loading the PostgreSQL and ADO.NET libraries:

```
; the first two assemblies are required for using the mono
;version of the postgresql data provider
(assembly-load-from ".\\deps\\mono.security.dll")
(assembly-load-from ".\\deps\\npgsql.dll")
```

Loading the ADO.NET library

```
(System.Reflection.Assembly/LoadWithPartialName "System.Data")
```

THE DB LAYER – NAMESPACES, IMPORT, AND GEN-CLASS

- Define the db.queries namespace
- Import the necessary classes - think using in C#
- Gen-class – exposing the function for use in C#

```
(ns db.queries
 (:import (Npgsql NpgsqlConnection NpgsqlCommand)
          (System.Data DataTable))
 (:gen-class
  :methods [ #^{:static true}
           [getPlayer [System.String] System.String]]))
```

THE DB LAYER - GET-PLAYER

```
(defn get-player
  "Gets the players demographics, scoring stats and
goalie stats
   Returned in a map that has the following keys:
   :demog :scoring :goalie"
  [lastname]
    (let [demog (get-player-demog lastname)
          playerid (:playerid demog)]
      {:demog demog
       :scoring (get-scoring-stats playerid)
       :goalie (get-goalie-stats playerid)}))
```

THE DB LAYER - RUN-SQL

```
(defn run-sql [sql-str]
  ; if dbconn isn't open open it
  (if (not= (str (.State dbconn)) "Open")
      (.Open dbconn))
  (let [cmd (NpgsqlCommand. sql-str dbconn)
        reader (.ExecuteReader cmd)
        data-table (DataTable.)]
    (.Load data-table reader)
    (.Close reader)
    (resultset-seq data-table)))
```

THE UI LAYER - LIBRARIES

- Uses System.Forms to create the UI
- After Loading the Assemblies the :import statement brings in all the .NET classes we need
- The :require statement is used to bring the db.queries functions into the hockey namespace
  - (:require [db.queries :as query])
  - All db.queries functions can be called using query instead of db.queries: queries/get-player
- The –main function is what starts off the app.

THE UI LAYER –MAIN FUNCTION

Creating the objects

```
(let [form (Form.)
        dialog (Form.)
        dialog-lbl (Label.)
        player-name-lbl (Label.)
        search-lbl (Label.)
        search-txt (TextBox.)
        search-btn (Button.)
        group-box (GroupBox.)
        background-worker (BackgroundWorker.)
        title-str "MyClojureAdventure.com - Hockey Player
Lookup"]
```

THE UI LAYER - SETTING ATTRIBUTES

- **Setting up the Search Button**

```
(doto search-lbl  (.set_Text "Last Name: ")
   (.set_Location (Point. 12 27))
   (.set_Size (Size. 70 22)))
```

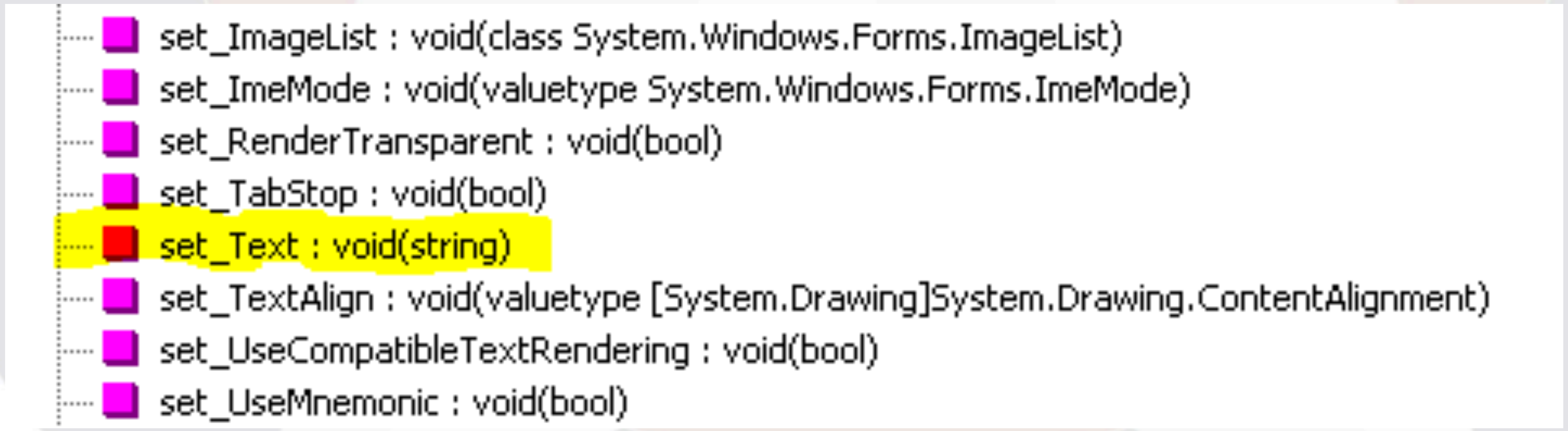
- **Adding the components to the form**

```
(doto (.Controls form)
      (.Add search-lbl)
      (.Add search-txt)
      (.Add search-btn)
      (.Add group-box))
```

- **The doto macro takes the first parameter and applies the trailing calls to that object. Same as:**
  - (.Add (.Controls form) search-lbl)
  - (.Add (.Controls form) search-txt)
  - Etc…

WHERE DID SET_TEXT COME FROM?

- When you look at the Label class in the Object Browser you don’t see a set_Text method
- Clojure-clr access .NET objects and the CLR ‘level’
- The set_Text method is the CLR representation of the Label.Text property.
- You can see what methods are available at the CLR level by using the ildasm tool which is part of the .NET SDK



```
set_ImageList : void(class System.Windows.Forms.ImageList)
set_ImeMode : void(valuetype System.Windows.Forms.ImeMode)
set_RenderTransparent : void(bool)
set_TabStop : void(bool)
set_Text : void(string)
set_TextAlign : void(valuetype [System.Drawing]System.Drawing.ContentAlignment)
set_UseCompatibleTextRendering : void(bool)
set_UseMnemonic : void(bool)
```

THE UI LAYER - EVENT HANDLING

- **The gen-delegate macro creates an EventHandler delegate**
  - First Parameter is type of delegate to create
  - Second Parameter is the delegate's parameter vector
  - The body of the function is passed as the third parameter

```
(.add_Click search-btn
      (gen-delegate EventHandler [sender args]
        (.set_Text dialog-lbl
           (str "Searching for " (.Text search-txt) "..."))
        (.RunWorkerAsync background-worker (.Text search-txt))
        (.ShowDialog dialog)))
```

THE UI LAYER – RETRIEVING THE DATA

- When the ‘Get Stats!’ button is clicked this code retrieves the data

(reset! qry-results (query/get-player name))

- Query/get-player is called to retrieve demographic and stats info
- The results are stored in qry-results
- What is the reset! Function all about?

THE UI LAYER – STORING THE RESULTS WITH AN ATOM

query-results is an atom, which means its state can change.

Defining an atom is very similar to any other def

(def qry-results (atom {}))

The reset! call changes the value of qry-results to the results of the get-player call.

(reset! qry-results (query/get-player name))

To access the data in qry-results use on of the following:

(deref qry-results) or @qry-results

THE UI LAYER - DISPLAY THE RESULTS

create-scoring-grid

The stats are added on a row by row basis using the doseq function.

```
 (doseq [rec (:scoring @qry-results)]
      (add-row-to-grid scoring-grid rec))
```

doseq allows us to process lazy sequences (think foreach)

Each entry in (:scoring @qry-results) will be assigned to the rec variable and passed to the fn that will add the data to the grid.

RUNNING A CLOJURE-CLR APP

- **Directory structure for my Clojure-clr projects:**
  - **bin** – base directory for executable. Contains all hockey related DLLs and executables.
  - **bin\deps and src\deps** – contains the projects dependencies (Npgsql.dll and Mono.Security.dll)

- **Pre-reqs for running hockey.exe**
  - 10 Clojure and 2 Microsoft DLLS need to be copied into the project's bin directory.
  - Src\hockey.clj to bin dir and src\db\queries.clj to bin\db \queries.clj
  - I use a build.bat file to do this

CALLING CLOJURE-CLR FROM C#

CLOJURE-CLR CODE

- **The code we will call from C# is in src/export/html.clj**
- **To make the clj code visible in C# we use :gen-class**

```
(:gen-class
   :methods [#^{:static true}
                   [CreateHtml [System.String] System.String]])
```

- :methods is a vector of vectors that describes each fn to be exposed
- [<function name> [<vector of params>] <return type>]
- By default when the CreateHtml function is called from C# it will look for a fn named –CreateHtml in the clojure code.
- -CreateHtml is a ‘normal’ Clojure fn

C# PROJECT SETUP

- **Required References**
  - All Clojure-clr dlls in the project’s bin directory
  - Microsoft.Scripting.dll and Microsoft.Dynamic.dll
  - Bin\export.html.clj.dll and bin\export.html.exe (You MUST reference both files)
  - All Dlls in the bin\deps directory

CALLING CREATEHTML

- **The C# Code**

```
static void Main(string[] args)
{
  var fileName = "ricci.html";
  var content  = export.html.CreateHtml("Ricci");

  System.IO.File.WriteAllText(fileName, content);
  System.Diagnostics.Process.Start(fileName);
}
```

RESOURCES

Me

@rippinrobr / rippinrobr@gmail.com

My Clojure Blog: www.myclojureadventure.com

My General Dev blog: http://progadventure.blogspot.com/

Github: github.com/rippinrobr

github.com/rippinrobr/intro-to-clojureclr-talk

Clojure and Clojure-clr Resources

Clojureclr.blogspot.com

https://github.com/clojure/clojure-clr/

Planet.clojure.in <- A TON of clojure related blogs