Vim Macros

Thomas Thornton (@tltjr)

Recording & Playback

- Begin recording with ‘q’ (normal mode) and selecting a register
- There are many registers in Vim, but typically we will use a letter when recording macros
- Playback the macro with ‘@{reg}’

Replay Multiple

- n@a where ’n’ is the number of times to replay, and ‘a’ is the register which contains your macro

Example - Header to POCO

poco-example.cs

```
1  public int WaterAccountNumber { get; set; }
2  public string RecentTransactionNumber { get; se
3  public string WaterAccountUsageTypeCode { get;
4  public DateTime AccountCreationDateTime { get;
5  public DateTime AccountTerminationDateTime { ge
6  public string CutToCityDate { get; set; }
7  public string AccountDescription { get; set; }
8  public int AccountLegalDescription { get; set;
9  public string DoNotSplitReasonCode { get; set;
10 public string ReceiveBeneficialUseWaterFlag { g
11 public double CombineWaterAccountNumber { get;
12 public double SpecialBillingWaterAccountNumber
13 public double OriginalWaterAccountNumber { get;
14 public double ExchangeWaterAccountNumber { get;
15 public string ExchangeContractSignedFlag { get;
16 public string ExchangeAgreementDescription { ge
17 public string PropertyAddressID { get; set; }
18 public string WaterDeliveryDistrictID { get; se
19 public string LandUseTypeCode { get; set; }
20 public string LandTypeCode { get; set; }
21 public string TotalLandAcreQuantity { get; set;
22 public string SpecialSpecialPumpFlag { get; set
23 public string SupplementalSupplyProgramApproval
24 public string WaterProviderCityLocationCode { g
25 public string WaterAccountClassificationCode {
26 public string WaterPostingPatternCode { get; se
27 public string BorrowWaterApprovalFlag { get; se
28 public string GroundwaterBasinCode { get; set;
29 public string GroundwaterEligibilityRestriction
30 public string AllowDemossWaterCode { get; set;
31 public string AllowSpillWaterFlag { get; set; }
32 public string AllowMoneyOverdraftFlag { get; se
33 public string CurrentYearReleaseTypeCode { get;
34 public string SubsequentYearReleaseTypeCode { g
35 public string LastWaterStatementDateTime { get;
36 public string BillingStatusCode { get; set; }
37 public string LastBillingDateTime { get; set; }
38 public string SectionNumber { get; set; }
39 public string TownshipNumber { get; set; }
```

Example - Map your new POCO to a view model

Recognize Queues That a Macro May Be Useful

- Obviously the primary clue that a macro is useful is having a set of keystrokes you will need to repeat
- Structured text is also a key indicator. Do these keystrokes need be repeated every x # of lines? Do they need to repeated for each instance of a search term?

Helpful “tricks”

- Have useful search targets established
- If you have searched for text with ‘/‘ or ‘?’, you will be able navigate the text with ’n’ or ’N’ when recording your macro.
- Same applies for the ‘*’ and ‘#’ keys

Example: Editing HTML

- First, we'll add a class to certain elements
- Next, let's change the name of a class

```
form name="settingsForm">
 <div class="tile is-vertical is-parent">
   <div class="tile is-child">
     <div class="field-body">
       <div class="field">
         <label class="label">Starting Balance</label>
         <div class="control">
           <input class="input" name="startingBalance" type="n
ce">
         </div>
       </div>
       <div class="field">
         <label class="label">Start Date</label>
         <div class="control">
           <input class="input" name="startDateName" mdInput [
s)="startDate.open()" [mdDatepicker]="startDate">
           <md-datepicker #startDate></md-datepicker>
         </div>
       </div>
       <div class="field">
         <label class="label">End Date</label>
         <div class="control">
           <input class="input" name="endDateName" mdInput [(n
endDate.open()" [mdDatepicker]="endDate">
           <md-datepicker #endDate></md-datepicker>
         </div>
       </div>
     </div>
   </div>
   <div class="tile is-child">
     <div class="field-body">
       <div class="field">
         <label class="label">Pay Amount</label>
```

Example: Multiple Files

- Macros aren't limited to the file you're editing
- I used one to copy my notes into an outline
- '/^# ' - search

```
# Vim - A Whirlwind Tour
> Thomas Thornton (@tltjr)

1. Why Vim?
  * Available in most editors and operating systems
  * Visual Studio, Emacs, and Sublime Text all have suppo
  * Available as a standalone editor on Linux, Mac, and W
  * Vim has NO competition as the premier way to efficien
2. [Download Vim!](https://vim.sourceforge.io/download.ph
![alt text](https://github.com/tltjr/vim/raw/master/vim.p

# Modes
  * modes - normal, visual, edit

  ![alt text](https://github.com/tltjr/vim/raw/master/mod
  * enter insert with 'i' (we'll see some more advanced c
  * insert mode is the ONLY mode most text editors have
  * return to normal node with ESC
  * normal mode allows us to use the keyboard for navigat
  * what's the advantage of this?
  * ability to edit and navigate code from the same finge

# Movement in a File
1. searching with / and ?
  * search from command line
2. \* and # with n, N
  * find the word under the cursor
3. :{linenumber}
  * to go to any line number
4. gg, G (gg - G)
  * top or end of file
5. Ctrl + F or B, up or back page
```

Example: Macros within macros

- Macros can contain any keystrokes you choose. This includes keys which replay other macros.

Example: Keep reusable macros in your .vimrc

```
49 "#################################
50 filetype on
51 filetype plugin on
52 filetype indent on
53
54 "#################################
55 "# Down with double quotes!!!
56 let @q = "0:s/\"/'/g^M"
57
58 "#################################
59 "#
60 "# Programming Stuff
61 "#
62 "#################################
63 set path+=include
64 set path+=lib
65 set tabstop=2
66 set softtabstop=2
67 set shiftwidth=2
```

- If you have macros you would like to always have access to; they can be stored in your .vimrc

Example: Edit your macros

- There are several ways to edit existing macros
- Use “ap to paste the macro into the editor (where a is the register where the macro is recorded)
- Edit the macro
- Use “ay$ to copy the edited macro back into your register
- If you need to insert special characters into the macro (like Esc), in insert mode, press Ctrl+v, then the character