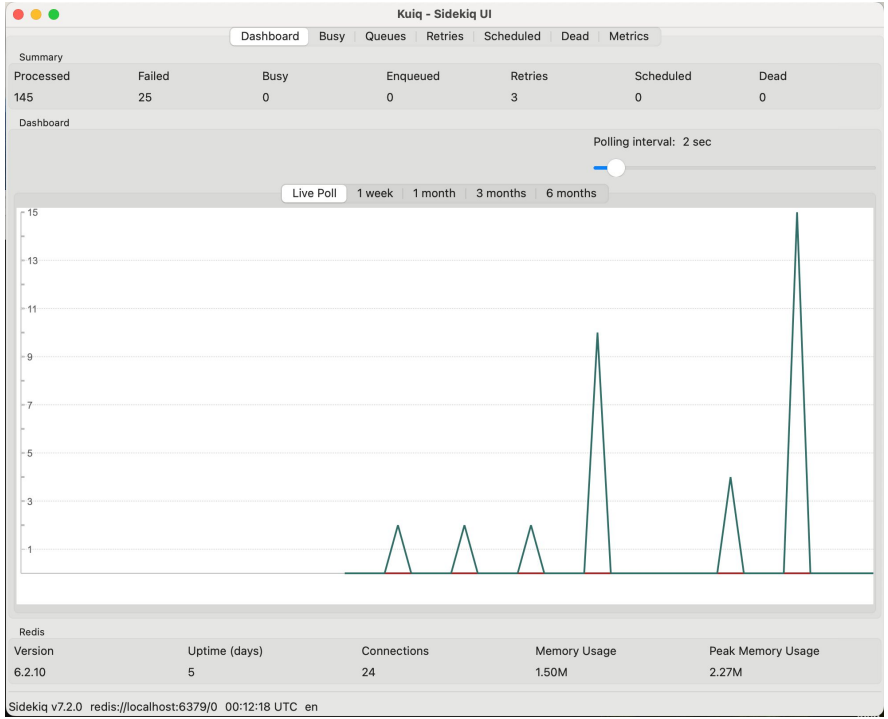
How To Build Basic Desktop Applications in Ruby

RubyConf 2024 Workshop (2h) • Thursday, November 14, 2023 11:15a-1:15p

Andy Maleh • Senior Software Engineer • Lexop • MS in Software Engineering • Fukuoka Ruby 2022 Special Award Winner • RailsConf/RubyConf Speaker

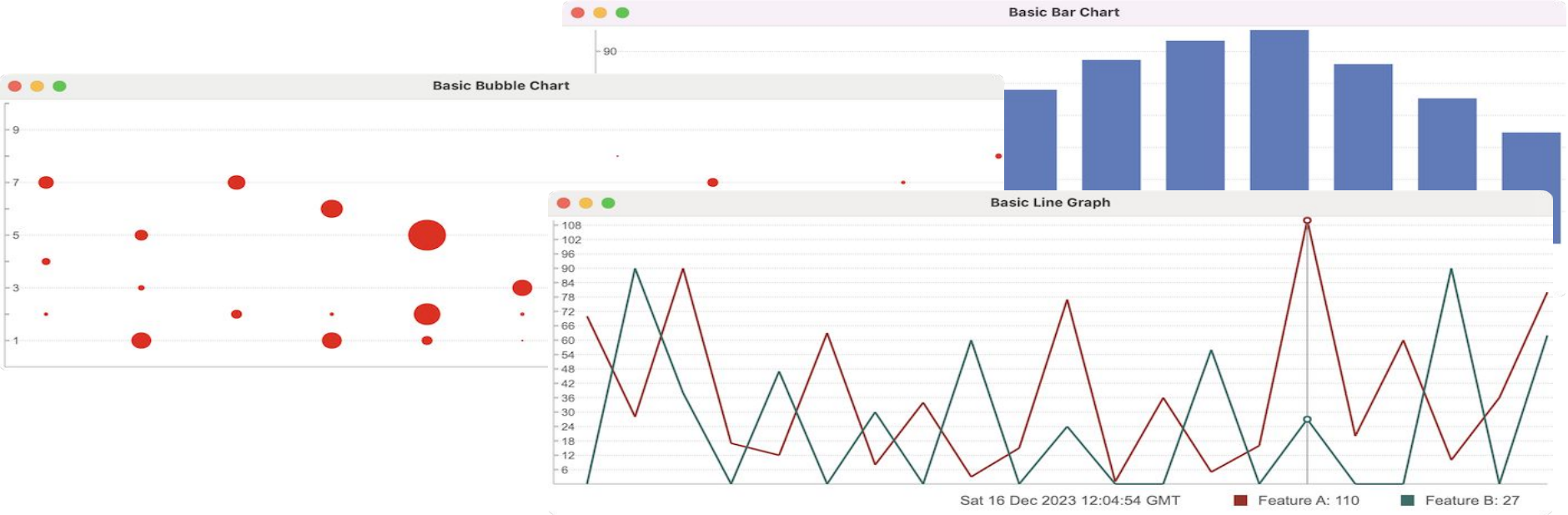
Kuiq

- https://github.com/mperham/kuiq
- Kuiq (UI for Sidekiq)
- Open-source Ruby gem
- Built by Mike Perham after attending this workshop in RubyConf 2023 (in collaboration with me).



Glimmer DSL for LibUI Graphs and Charts

- https://github.com/AndyObtiva/glimmer-libui-cc-graphs_and_charts
- Open-Source Ruby gem extracted from Kuiq



Outline

1. GUI Basics
2. MVC Software Architecture
3. MVP & Data-Binding
4. Advanced Data-Binding
5. Custom Components (Bonus)
6. Scaffolding (Bonus)

https://github.com/AndyObtiva/how-to-build-desktop-applications-in-ruby

GitHub Repo for RubyConf 2024 Workshop: How To Build Desktop Applications in Ruby

Setup Workshop GitHub Repository

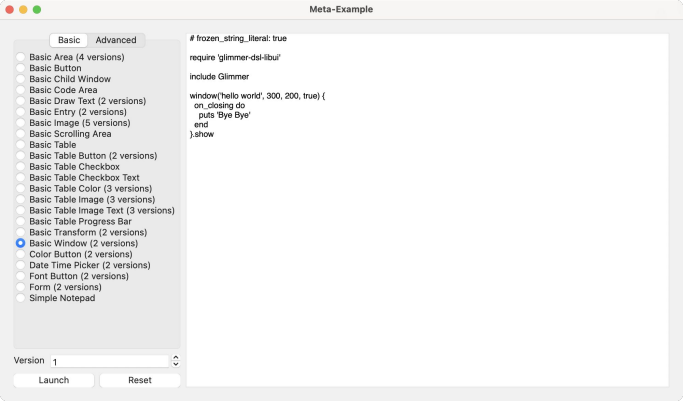
- Before moving forward, please clone this GitHub repo ("⭐" on GitHub):

```
git clone
https://github.com/AndyObtiva/how-to-build-desktop-applications-in-ruby
```

- Enter repo directory and run: **`bundle`**
- This repo contains a link to presentation slides and code for all the exercises of the workshop, organized by section/exercise number.
- The Resources section includes links for helpful references
- Follow presentation slides at: bit.ly/rubyconf2024desktop

Setup Glimmer DSL for LibUI

1. Open Terminal/Command-Prompt/Git-Bash
2. Install this gem:
   - **`gem install glimmer-dsl-libui`**
3. Run this command to launch Glimmer Meta-Example:
   - **`glimmer examples`**



Meta-Example

Basic | Advanced

- Basic Area (4 versions)
- Basic Button
- Basic Child Window
- Basic Code Area
- Basic Draw Text (2 versions)
- Basic Entry (2 versions)
- Basic Image (5 versions)
- Basic Scrolling Area
- Basic Table
- Basic Table Button (2 versions)
- Basic Table Checkbox
- Basic Table Checkbox Text
- Basic Table Color (3 versions)
- Basic Table Image (3 versions)
- Basic Table Image Text (3 versions)
- Basic Table Progress Bar
- Basic Transform (2 versions)
- Basic Window (2 versions)
- Color Button (2 versions)
- Date Time Picker (2 versions)
- Font Button (2 versions)
- Form (2 versions)
- Simple Notepad

Version 1

Launch | Reset

```
# frozen_string_literal: true

require 'glimmer-dsl-libui'

include Glimmer

window('hello world', 300, 200, true) {
  on_closing do
    puts 'Bye Bye'
  end
}.show
```

Meta-Example

Basic | Advanced

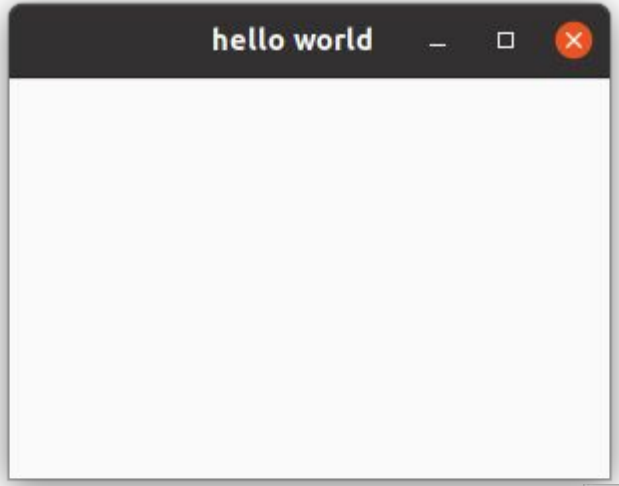
- Area Based Custom Controls
- Area Gallery (4 versions)
- Button Counter
- Class Based Custom Controls
- Color The Circles
- Control Gallery
- Cpu Percentage
- Custom Draw Text (2 versions)
- Dynamic Area (4 versions)
- Editable Column Table
- Editable Table
- Form Table (5 versions)
- Gpt2 Notepad
- Grid
- Histogram (2 versions)
- Login (5 versions)
- Method Based Custom Controls (2 versions)
- Midi Player (3 versions)
- Paginated Refined Table
- Shape Coloring
- Snake (2 versions)
- Tetris
- Tic Tac Toe (2 versions)
- Timer (2 versions)

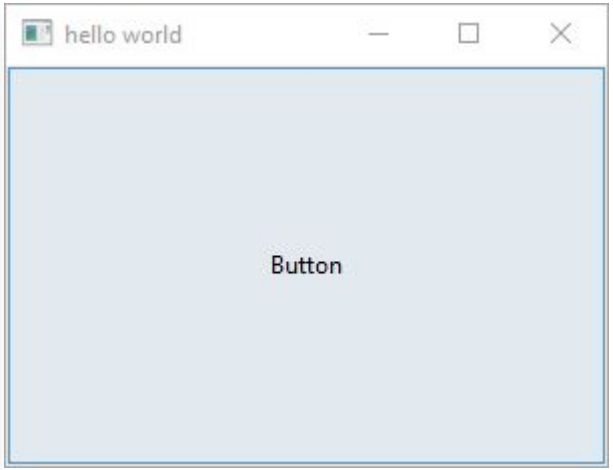
Version 1

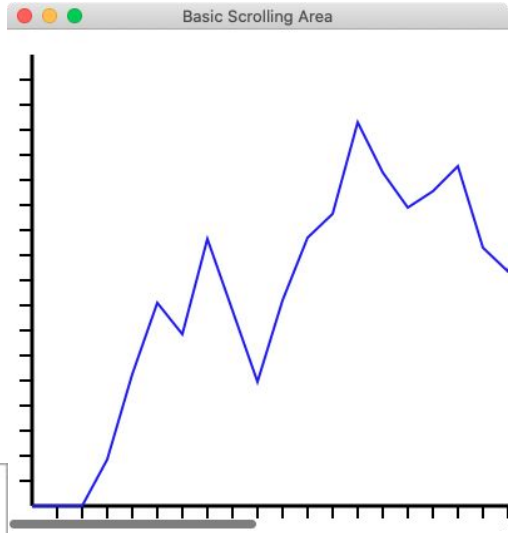
Launch | Reset

```
window('Color The Circles', WINDOW_WIDTH, WINDOW_HEIGHT) {
  margined true

  grid {
    button('Restart') {
      left 0
      top 0
      halign :center

      on_clicked do
        restart_game
      end
    }

    label('Score goes down as circles are added. If it reaches -20, you lose!') {
      left 0
      top 1
      halign :center
    }

    label('Click circles to color and score! Once score reaches 0, you win!') {
      left 0
      top 2
      halign :center
    }

    horizontal_box {
      left 0
      top 3
      halign :center

      label('Score:') {
        stretchy false
      }

      @score_label = label(@score.to_s) {
        stretchy false
```

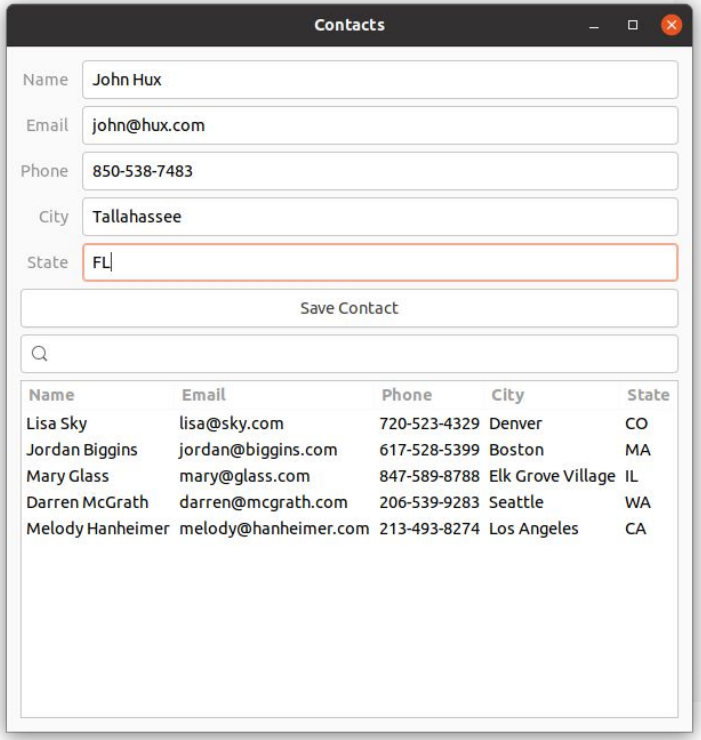
Basic Examples

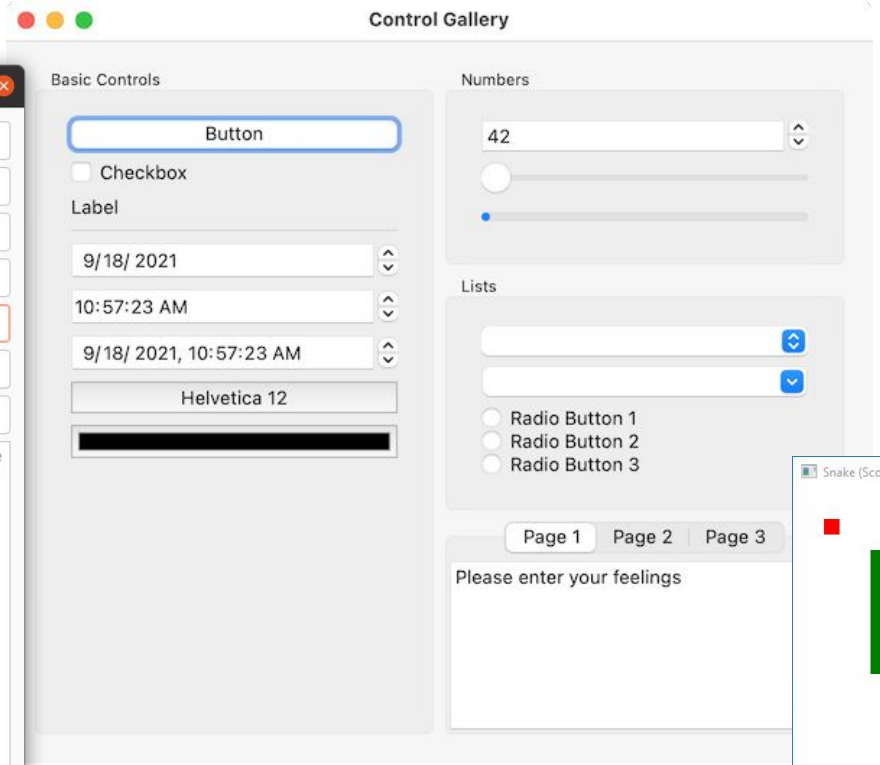


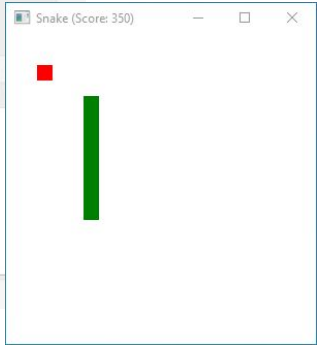




Advanced Examples







Section 1
GUI Basics

Introduction To Glimmer

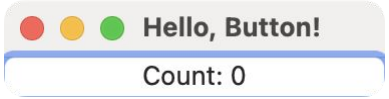
```
1   win
```

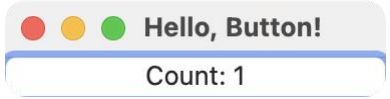
Source: [Glimmer DSL for SWT]

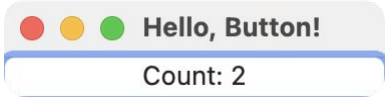
Button Counter Example

```ruby
require 'glimmer-dsl-libui'

class ButtonCounter
  include Glimmer::LibUI::Application

  before_body do
    @counter = Counter.new
  end

  body {
    window('Hello, Button!', 190, 20) {
      button {
        # data-bind button text to @counter count, converting to string on read from model.
        text <= [@counter, :count, on_read: ->(count) {"Count: #{count}"}]

        on_clicked do
          # This change will automatically propagate to button text through data-binding above
          @counter.count += 1
        end
      }
    }
  }
end

ButtonCounter.launch
```

```ruby
class Counter
  attr_accessor :count

  def initialize
    self.count = 0
  end
end
```







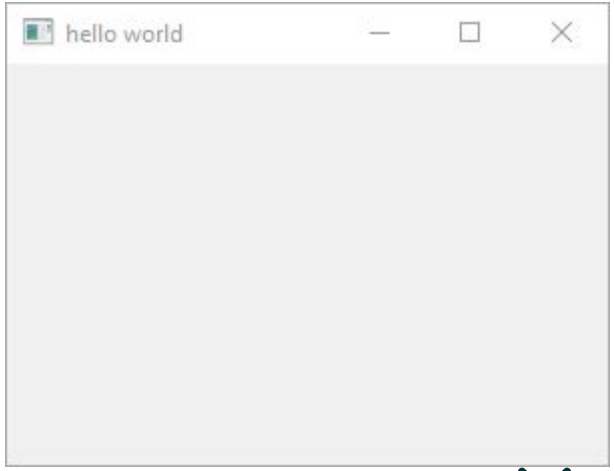
Glimmer DSL for LibUI Design Principles

- The Ruby Way (including TIMTOWTDI: There Is More Than One Way To Do It)
- Requiring the least amount of syntax possible to build GUI (Graphical User Interface)
- Declarative syntax that visually maps to the GUI control hierarchy
- Ability to mix declarative and imperative code conveniently without needing awkward & verbose technologies (e.g. no XML, HTML, ERB scriptlets, JSX)
- Computers serve Software Engineers (not Software Engineers serve Computers)
- Think only about real world concepts directly relevant to the GUI and interacting with it (no weird non-real-world irrelevant concepts like hooks/effects/immutability)
- The Rails Way Convention over Configuration via smart defaults and automation of low-level details
- Modular Software Design (e.g. support for Components)
- No premature optimization

GUI DSL Basics

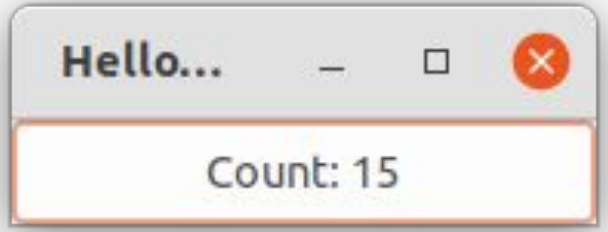
- Glimmer GUI DSL enables building Graphical User Interfaces with a Ruby embedded (internal) Domain Specific Language
- 1) Control Keyword
  - Underscored case
  - Declarative control construction
  - Ruby method behind the scenes

```
window
```



GUI DSL Basics

- 1) Control Keyword
  - Underscored case
  - Declarative control construction
  - Ruby method behind the scenes

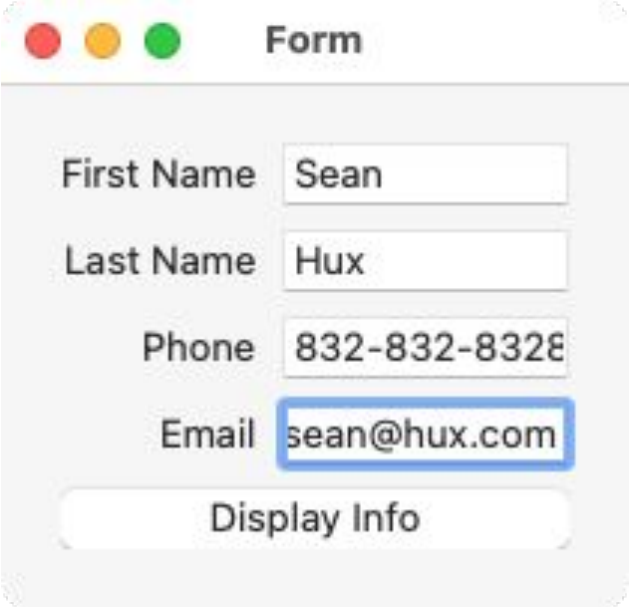



```
button
```

GUI DSL Basics

- 1) Control Keyword
  - Underscored case
  - Declarative control construction
  - Ruby method behind the scenes

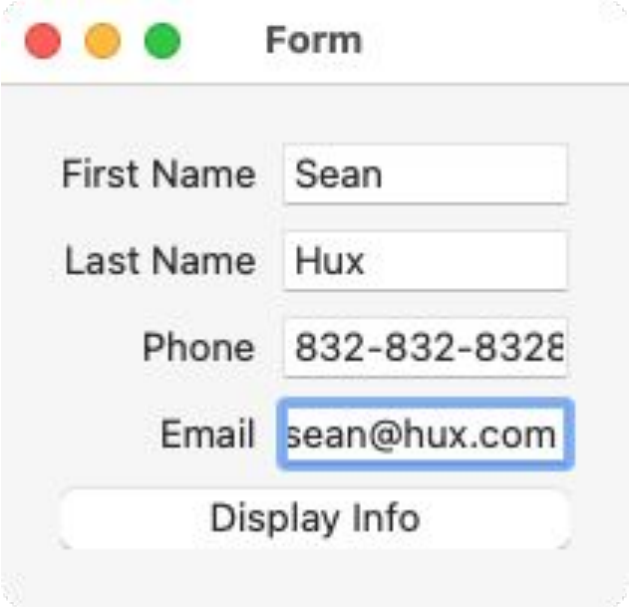
```
label
```



GUI DSL Basics

- 1) Control Keyword
  - Underscored case
  - Declarative control construction
  - Ruby method behind the scenes

```
entry
```



Controls

- area
- button
- checkbox
- code_area
- combobox
- color_button
- date_time_picker
- editable_combobox
- entry
- font_button
- form
- grid
- group
- horizontal_box
- horizontal_separator
- image
- label

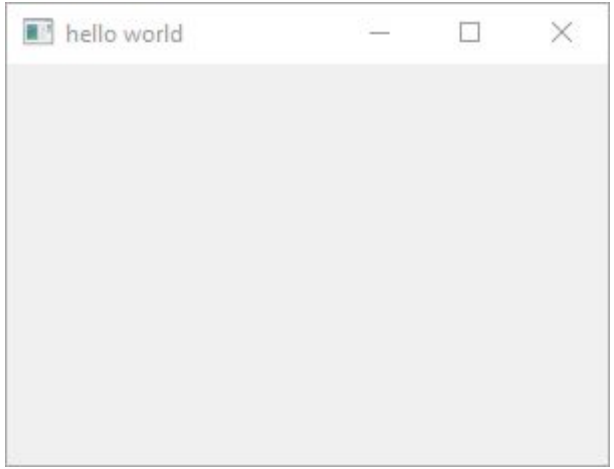
Full list at: Glimmer DSL for LibUI Supported Keywords

More Controls

- menu
- menu_item
- message_box
- multiline_entry
- password_entry
- progress_bar
- radio_buttons
- scrolling_area
- slider
- spinbox
- tab
- tab_item
- table
- time_picker
- vertical_box
- vertical_separator
- window

Full list at: Glimmer DSL for LibUI Supported Keywords

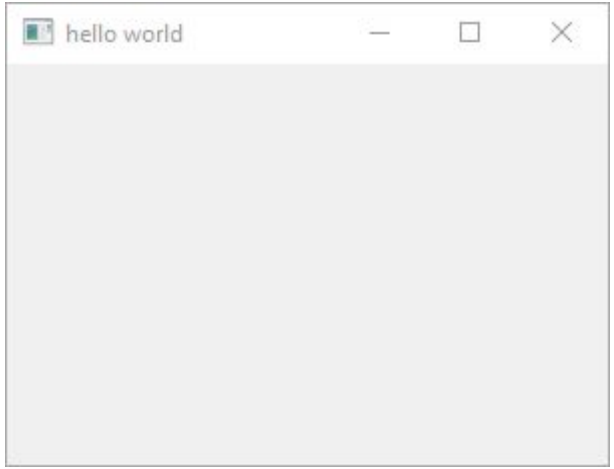
GUI DSL Basics



- 2) Control Arguments
  - Optional
  - Match LibUI C API

```
window('hello world')
```

GUI DSL Basics



- 2) Control Arguments
  - Optional
  - Match LibUI C API

```
window('hello world', 300, 200)
```

GUI DSL Basics

- 3) Control Content Block
  - Properties
  - Listeners
  - Nested Controls

```
window {
    ...
}
```

GUI DSL Basics

- 3) Control Content Block
  - Properties
  - Listeners
  - Nested Controls

```
window('hello world') {
    ...
}
```

GUI DSL Basics

- 3a) Control Content Block - Properties

```
window {
  title 'hello world'
  content_size 300, 200
}
```

Control Properties

- text
- checked
- editable
- selected
- color
- time
- margined
- padded
- value
- items
- read_only
- title
- width
- height

Full list at: Glimmer DSL for LibUI Supported Keywords

GUI DSL Basics

- 3b) Control Content Block - Listeners
  - Have `on_` prefix
  - Observer Pattern
  - Imperative code block

```
window {
  on_closing do
    puts 'Bye Bye'
  end
}
```

GUI DSL Basics

- 3b) Control Content Block - Listeners
  - Have `on_` prefix
  - Observer Pattern
  - Imperative code block

```
button('Click') {
  on_clicked do
    ...
  end
}
```

Control Listeners

- on_clicked
- on_changed
- on_toggled
- on_closing
- on_content_size_changed
- on_destroy
- on_draw
- on_selected
- on_edited
- on_mouse_down
- on_mouse_up
- on_mouse_drag_started
- on_mouse_dragged
- on_mouse_dropped
- on_mouse_entered
- on_mouse_exited
- on_key_down
- on_key_up

Full list at: Glimmer DSL for LibUI Supported Keywords

GUI DSL Basics

- 3c) Control Content Block - Nested Controls
  - Declarative Hierarchy
  - Composite Pattern

```
window {
   button('Click') {
      ...
   }
}
```

GUI DSL Basics

- 3d) Control Content Block - Layouts
  - Layout controls:
    - **`vertical_box`**
    - **`horizontal_box`**
    - **`form`**
    - **`grid`** (unstable)
  - Inside box layouts, **`stretchy`** prop decides taking all space or not
  - Inside form layout, **`label`** prop generates label for each control

```
window('Animals', 50, 50) {
  margined true

  vertical_box {
    label('Animals Form') {
      stretchy false
    }

    form {
      entry {
        label 'Name'
      }
      entry {
        label 'Habitat'
      }
    }
  }
}.show
```

GUI DSL Basics

- 3e) Control Content Block - Table Columns
  - Columns can be nested under **`table`**:
    - background_color_column
    - checkbox_column
    - checkbox_text_column
    - checkbox_text_color_column
    - image_column
    - image_text_column
    - image_text_color_column
    - text_column
    - text_color_column
    - progress_bar_column


```
table {
  text_color_column('Animal')
  text_color_column('Sound')
  checkbox_text_color_column('Description')
  image_text_color_column('GUI')
  background_color_column

  cell_rows animals
}
```

GUI DSL Basics

```
window {
  ...
}.show
```

- 4) Control Operations
  - Invoked through Ruby Methods
  - Match LibUI C API
  - Proxy calls to wrapped LibUI objects
  - Include access to control properties after construction
  - Behavior is sometimes augmented with smart defaults
    - e.g. **`window.show`** starts GUI event loop

GUI DSL Basics

- 4) Control Operations (Ruby object methods)

```
terms_checkbox = checkbox('Agree To Terms')
terms_checkbox.checked? # => false
terms_checkbox.checked = true
terms_checkbox.checked? # => true
```

Section 1
GUI Basics
Exercises

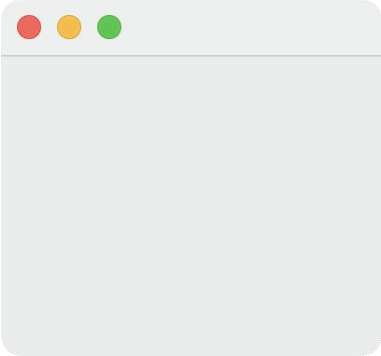
Exercise 1: Empty Window

- Open Glimmer Meta-Example:

  ```
  glimmer examples
  ```

- Go to Basic Window example
- Delete all code below `include Glimmer`
- Add this code in place of old code and launch app:

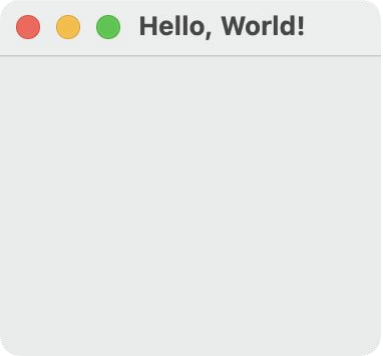
  ```
  window.show
  ```



Exercise 2: Hello, World! Window w/ Args

- Add this code in place of last code below **`include Glimmer`** and launch app:

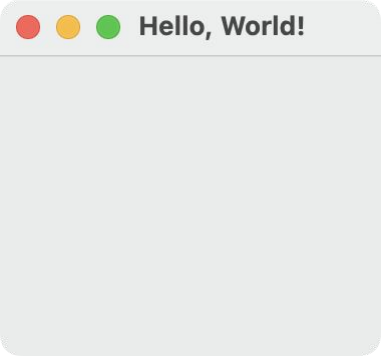
```
window('Hello, World!').show
```



Exercise 3: Hello, World! Window w/ Props

- Add this code in place of last code below **`include Glimmer`** and launch app:

```
window {
  title 'Hello, World!'
}.show
```



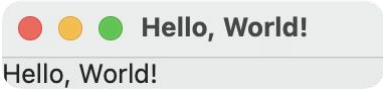
Exercise 4: Hello, World! Window + Label w/ Args

- Add this code in place of last code and launch app:


```
window('Hello, World!') {
  label('Hello, World!')
}.show
```

Exercise 5: Hello, World! Window + Label w/ Props

- Add this code in place of last code and launch app:

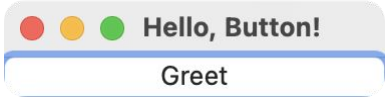



```
window {
  title 'Hello, World!'

  label {
    text 'Hello, World!'
  }
}.show
```

Exercise 6: Hello, Button!

- Add this code in place of last code and launch app:

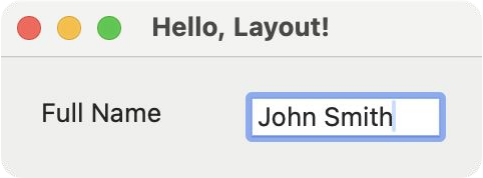
```
window('Hello, Button!') {
  button('Greet') {
    on_clicked do
      msg_box('Greeting', 'Hello!')
    end
  }
}.show
```



Exercise 7: Hello, Layout! w/ Horizontal Box

- Add this code in place of last code and launch app:

```
window('Hello, Layout!') {
  content_size 50, 20
  margined true

  horizontal_box {
    label('Full Name')
    entry {
      text 'John Smith'
    }
  }
}.show
```



Exercise 8: Hello, Layout! w/ Horizontal & Vertical Boxes

- Add this code in place of last code and launch app:

```
window('Hello, Layout!') {
  content_size 50, 20
  margined true
  vertical_box {
    horizontal_box {
      label('Full Name')
      entry {
        text 'John Smith'
      }
    }
    horizontal_box {
      label('DOB')
      date_picker {
        time year: 2004, mon: 11, mday: 17
      }
    }
  }
}.show
```

Exercise 9: Hello, Layout! w/ Box Stretchy

- Add this code in place of last code and launch app:

```
window('Hello, Layout!') {
  content_size 50, 20
  margined true
  vertical_box {
    horizontal_box {
      label('Full Name') {
        stretchy false
      }
      entry {
        text 'John Smith'
      }
    }
    horizontal_box {
      label('DOB') {
        stretchy false
      }
      date_picker {
        time year: 2004, mon: 11, mday: 17
      }
    }
  }
}.show
```

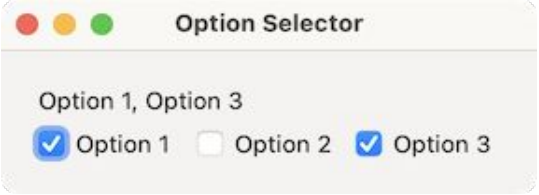
Exercise 10: Hello, Layout! Form

- Add this code in place of last code and launch app:

```
window('Hello, Layout!') {
  content_size 50, 20
  margined true

  form {
    entry {
      label 'Full Name'
      text 'John Smith'
    }

    date_picker {
      label 'DOB'
      time year: 2004, mon: 11, mday: 17
    }
  }
}.show
```

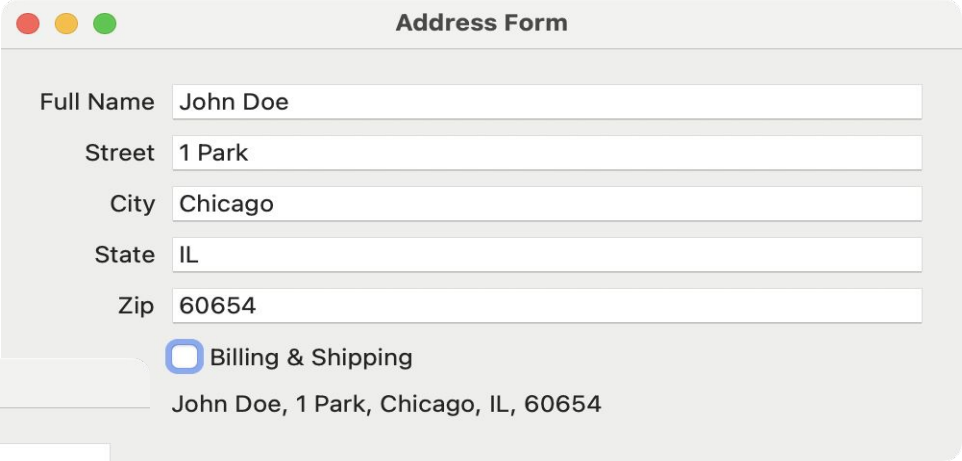
Exercise 11: Option Selector

```
window('Option Selector') {
  content_size 50, 20
  margined true

  vertical_box {
    @selected_options_label = label('None')

    horizontal_box {
      @checkboxes = 3.times.map do |n|
        checkbox("Option #{n+1}") {
          on_toggled do
            puts "Checkbox '#{@checkboxes[n].text}' checked property changed to: #{@checkboxes[n].checked?}"
            checked_checkboxes = @checkboxes.select(&:checked?)
            if checked_checkboxes.empty?
              @selected_options_label.text = 'None'
            else
              @selected_options_label.text = checked_checkboxes.map(&:text).join(', ')
            end
          end
        }
      end
    }
  }
}.show
```



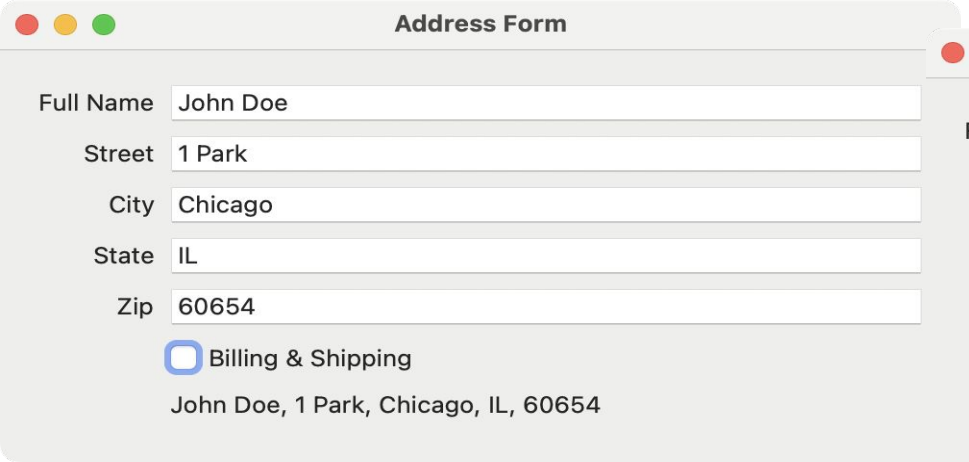
Section 1 Test

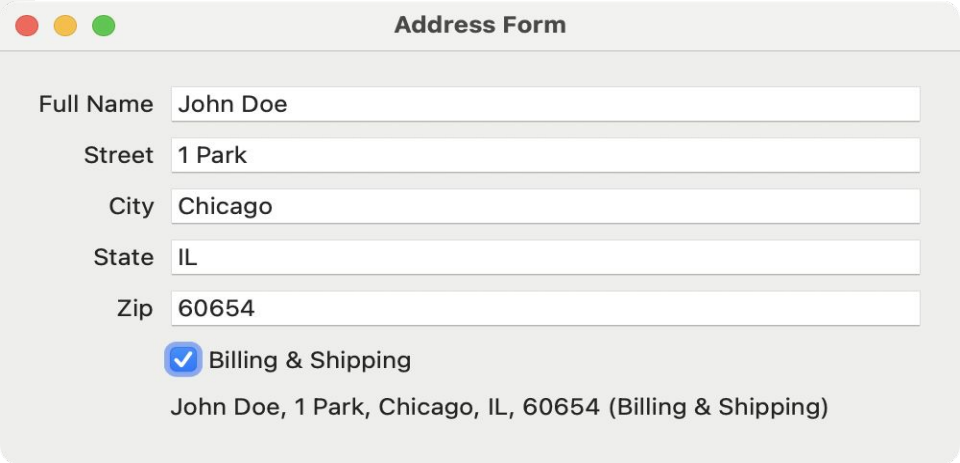
- Implement an address form with the following fields:
  - Full Name (`entry`)
  - Street (`entry`)
  - City (`entry`)
  - State (`entry`)
  - Zip (`entry`)
  - Billing & Shipping (`checkbox`)



Section 1 Test 1

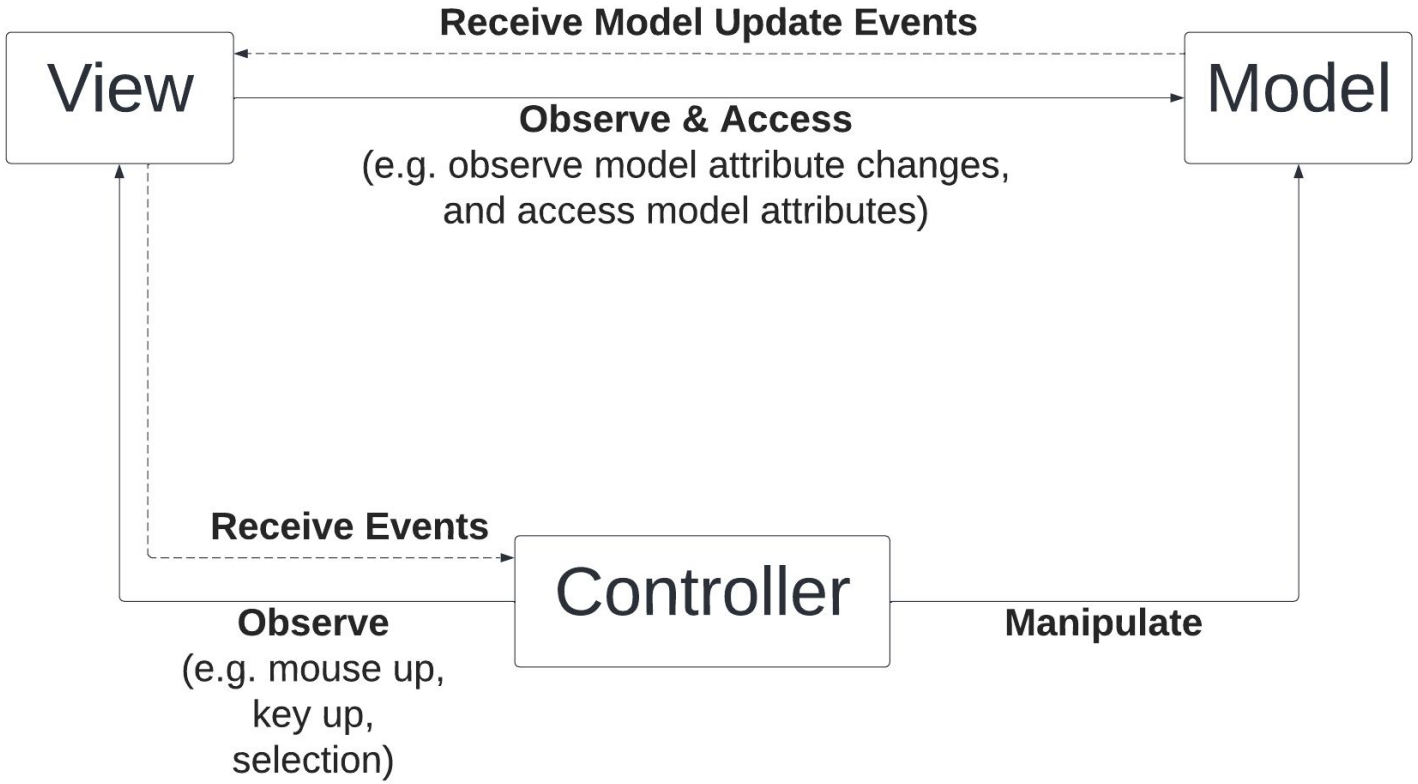
- Add a summary label at the bottom that is updated when these listeners are triggered:
  - **`on_changed`** for entry fields
  - **`on_toggled`** for checkbox field
- Address summary format examples:
  - Unchecked Billing & Shipping: "John Doe, 1 Park, Chicago, IL, 60654"
  - Checked Billing & Shipping: "John Doe, 1 Park, Chicago, IL, 60654 (Billing & Shipping)"





Section 2
MVC Software Architecture

MVC Software Architecture



Observer Pattern

- Observer Design Pattern is foundational in MVC Architectural Pattern
- In MVC (Model-View-Controller):
  - Controller observes View for user actions (e.g. mouse/keyboard) and trigger changes in Model
  - View observes Model for domain data changes and updates itself
- In Glimmer GUI DSL (Graphical User Interface Domain Specific Language):
  - Controller can rely on Listeners to observe View actions and trigger changes in Model
  - View can rely on **`observe(model, attribute)`** method to observe a Model attribute for changes and update its controls accordingly. It relies on Ruby Meta-Programming to auto-enhance a Model into an **`ObservableModel`**

Glimmer::DataBinding::Observer

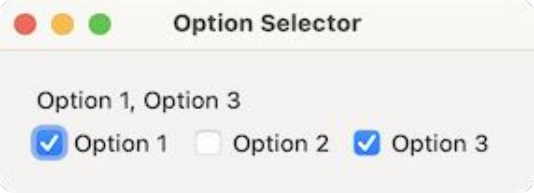
Behind the scenes, the **`observe`** keyword, which requires **`include Glimmer`**:

```
observe(model, attr) {
    ...
}
```

is equivalent to the following code WITHOUT **`include Glimmer`**, which is useful in Models that need to observe other Models without mixing the Glimmer DSL:

```
Glimmer::DataBinding::Observer.proc {
    ...
}.observe(model, attr)
```

MVC Example - Explicit Controller



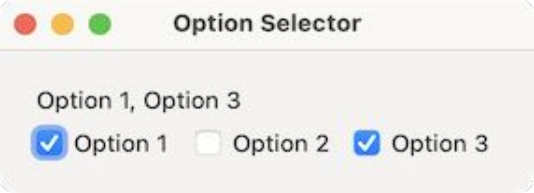
View observed by Explicit Controller to indirectly toggle option in Model

```
on_toggled do
  @option_selector_controller.toggle_option(option_number)
end
```

Model observed by View to indirectly update View label text when Model changes

```
observe(@option_selector_model, :selected_options) do
  @selected_options_label.text = @option_selector_model.summary
end
```

MVC Example - Implicit Controller



View observed by Implicit Controller (Listener) to directly toggle option in Model

```
on_toggled do
  @option_selector_model.toggle_option(option_number)
end
```

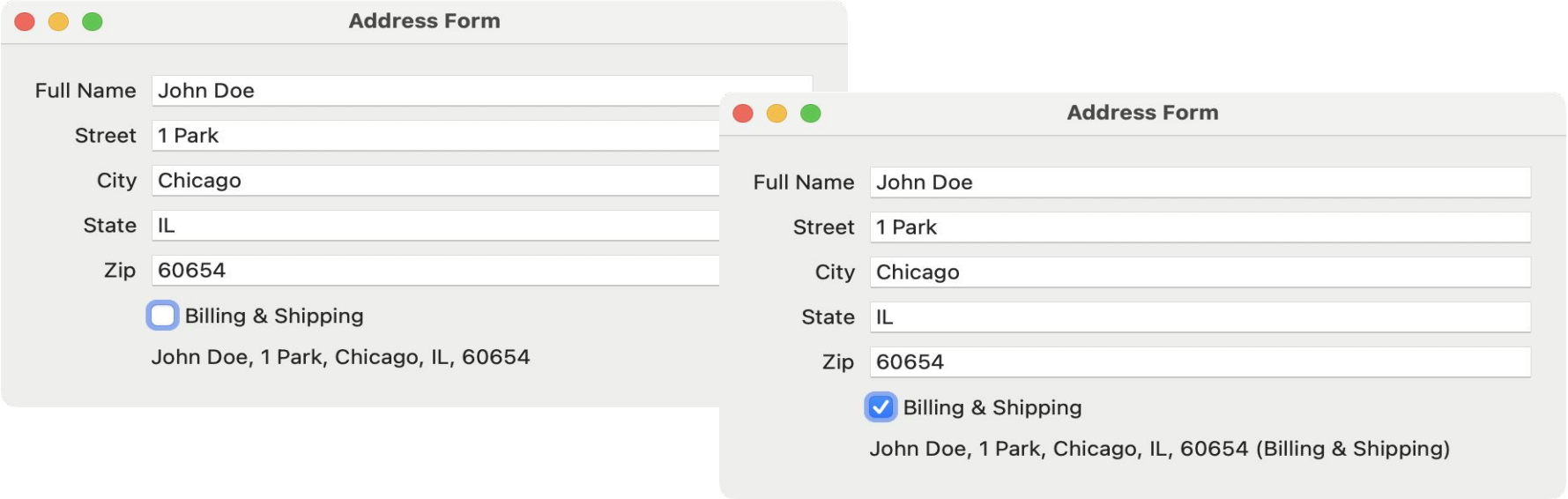
Model observed by View to indirectly update View label text when Model changes

```
observe(@option_selector_model, :selected_options) do
  @selected_options_label.text = @option_selector_model.summary
end
```

Section 2
MVC Software Architecture Exercises
(Follow on GitHub)

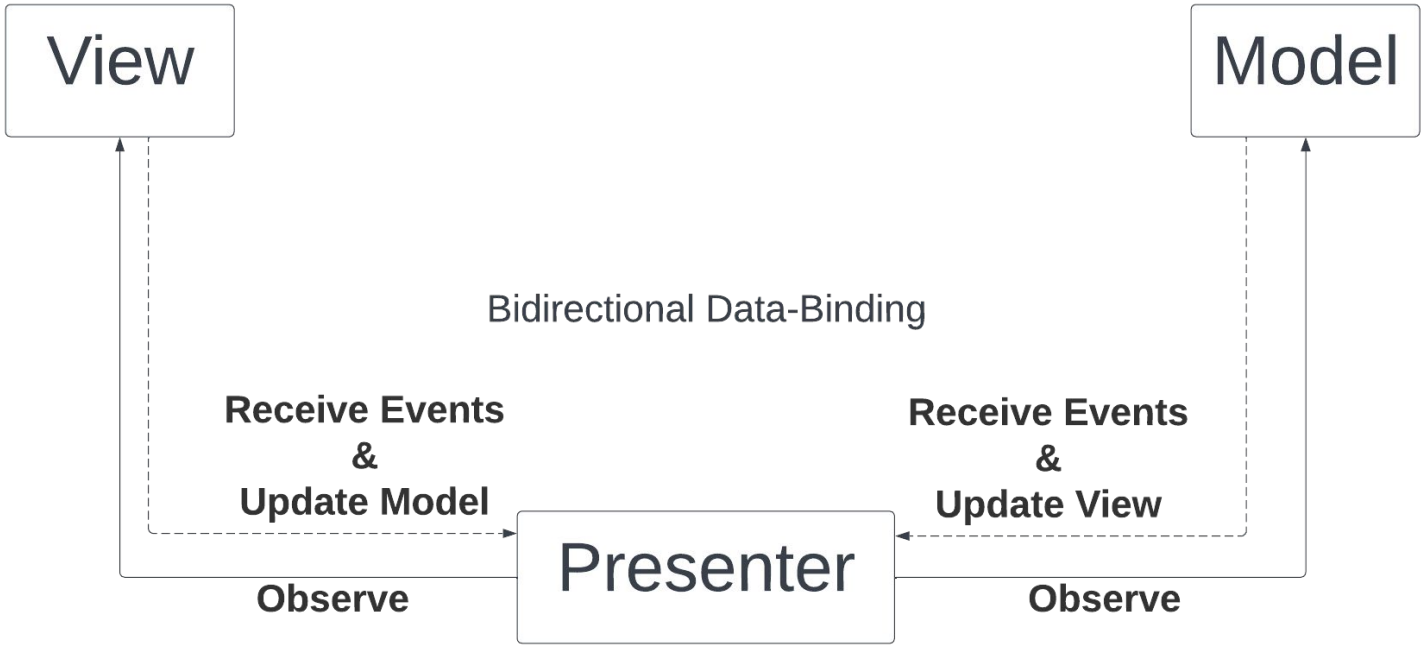
Section 2 Test 1

- Refactor address form app from Section 1 Test to follow MVC with implicit Controller:
  - **`Address`** class (Model): stores text/boolean attributes and computes summary
  - **`AddressFormView`** class (View): wires listeners to update Model attributes & observes Model attributes to update address summary



Section 3
MVP & Data-Binding

MVP & Data-Binding Software Architecture



Property Data-Binding

- Property Data-Binding declaratively & automatically synchronizes View properties w/ Model attributes bidirectionally or unidirectionally
- Bidirectional (two-way) with **<=>** for View read/write properties

```
entry {
  text <=> [@user, :first_name]
}

date_time_picker {
  time <=> [@reservation, :event_time]
}
```

Property Data-Binding

- Unidirectional (one-way) with **<=** for View read-only properties

```
button {
  text <= [@counter, :count, on_read: :to_s]
  ...
}
label {
  text <= [@order_presenter, :total_with_currency]
}
```

MVC+MVP Data-Binding Example

```
entry {
  text <=> [@counter, :count, on_read: :to_s, on_write: :to_i]
}

button {
  text <= [@counter, :count, on_read: ->(value) { "Click To Increment: #{value}" }]

  on_clicked do
    @counter.count += 1
  end
}
```

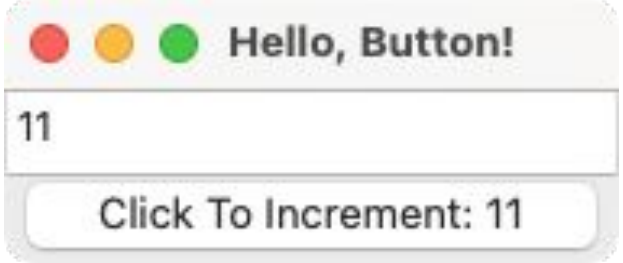


Table Data-Binding

- **`table cell_rows`** can be data-bound to an array of models/hashes or Enumerator/Enumerator::Lazy for lazy-loading
- Table data-binding automatically detects model attributes from column names by convention
- Setting editable true enables editing table & bidirectional data-binding

```
table {
  text_column('Name')
  text_column('Email')
  text_column('Phone')
  text_column('City')
  text_column('State')

  editable true
  cell_rows <=> [user, :contacts]
}
```

Table Data-Binding

- **`table cell_rows`** column mapping may still be customized manually if needed by passing in **`column_attributes`** option

```
cell_rows <= [country, :animals,
              column_attributes: {
                'Animal' => :name_color,
                'Sound' => :sound_color,
                'Description' => :mammal_description_color,
                'GUI' => :image_description_color
              }]
```

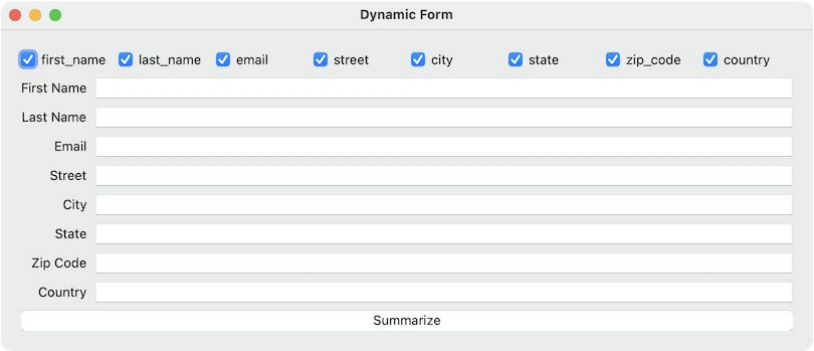
Content Data-Binding

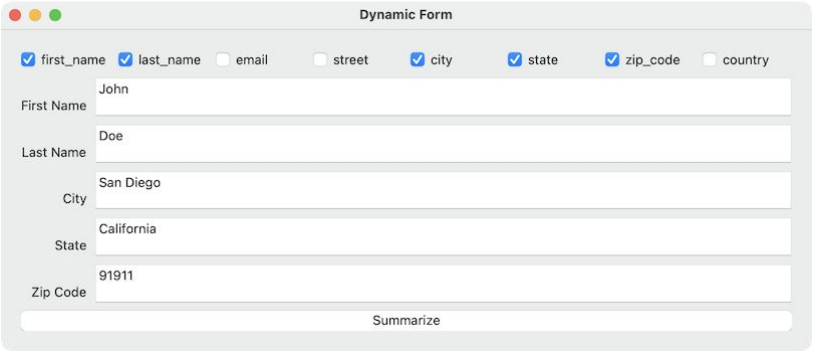
- Content Data-Binding declaratively & automatically generates (re-renders) View content in response to Model attribute changes
- Under a control, nest dynamically generated content inside an explicit **`content`** block that specifies the model and attribute to observe:

```
content(model, attribute) {
    … # controls to generate dynamically
}
```

Content Data-Binding Example

```
form {
  content(@user, :customizable_attributes) {
    @user.customizable_attributes.each do |attribute|
      entry {
        label attribute.to_s.split('_').map(&:capitalize).join(' ')
        text <=> [@user, attribute]
      }
    end
  }
}
```

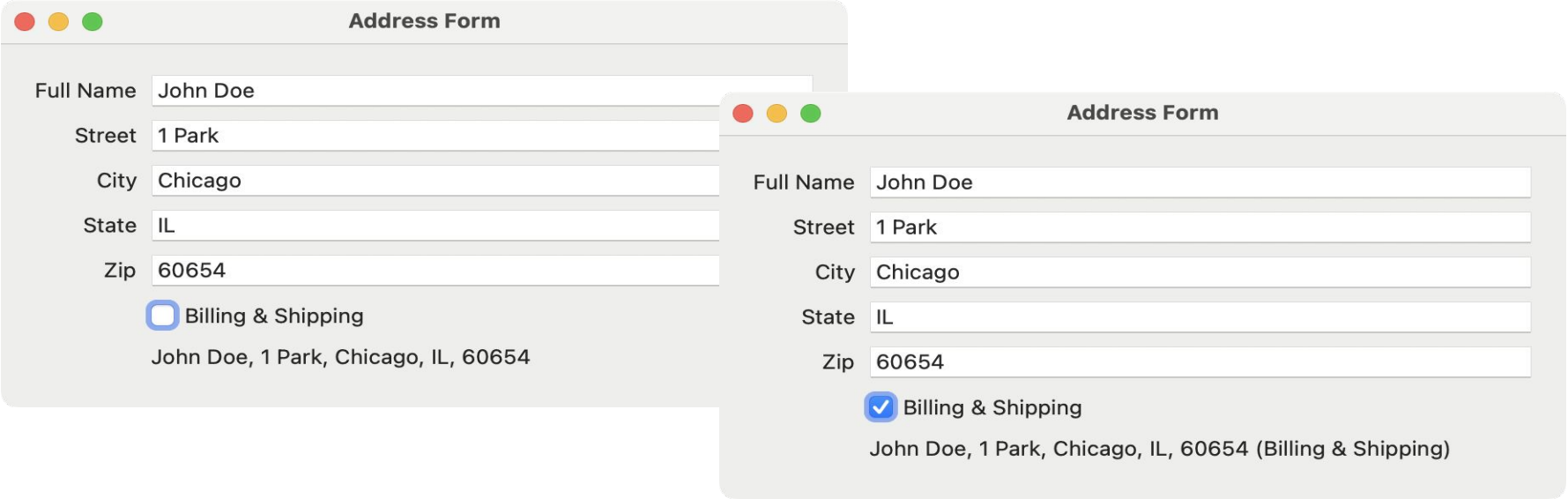




Section 3
MVP & Data-Binding Exercises
(Follow on GitHub)

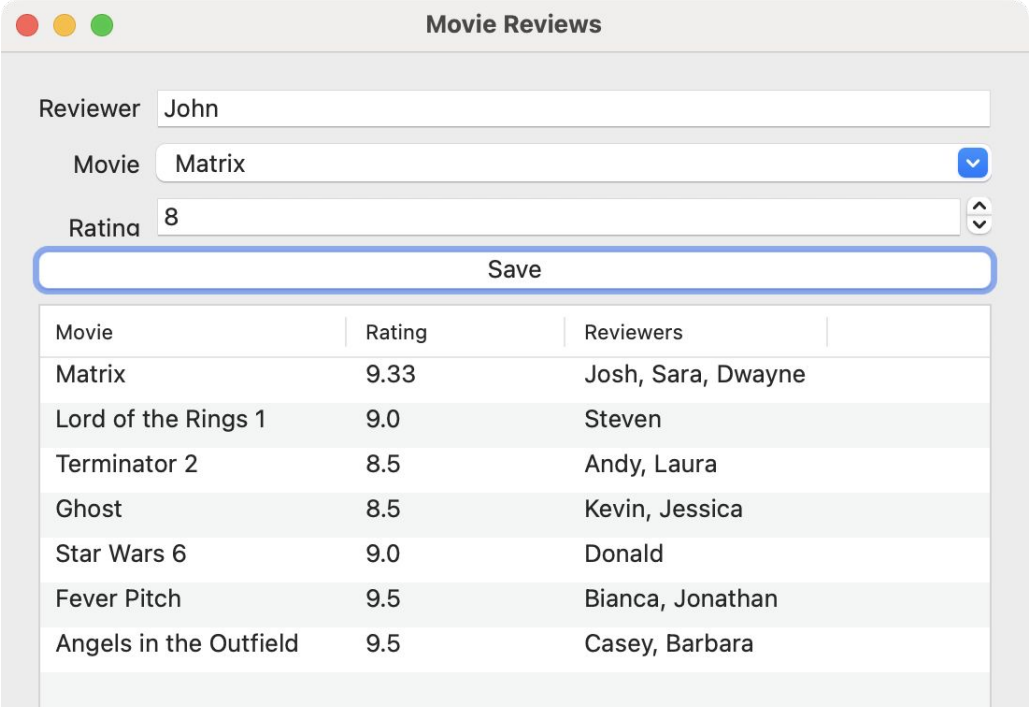
Section 3 Test 1

- Refactor address form app from Section 2 Test to follow MVP with Data-Binding:
  - **`Address`** class (Model): stores text/boolean attributes and computes summary
  - **`AddressFormView`** class (View): data-binds fields to Model attributes using the **`computed_by`** option for data-binding the Model summary attribute



Section 3 Test 2

- Movie Reviews App that stores movie reviews by reviewers and then calculates the average rating for each movie while listing movie ratings in a table.



Section 3 Test 2

- Movie Reviews App that follow MVP with Table Data-Binding & Attribute Data-Binding:
  - **`MovieReview`** class (Model): stores/resets attributes (can be **`Struct`**):
    - **`[attribute] reviewer(String)`**
    - **`[attribute] movie(String)`**
    - **`[attribute] rating(Float)`**
  - **`MovieRating`** class (Model): stores attributes and formats rating (can be **`Struct`**):
    - **`[attribute] movie(String)`**
    - **`[attribute] average_rating (Float)`**
    - **`[attribute] reviewers (String)`**: comma-separated list of reviewers
    - **`[method] rating`** formats **`average_rating`** as 2 decimal **`String`**

Section 3 Test 2

- Movie Reviews App that follow MVP with Table Data-Binding & Attribute Data-Binding:
  - **`MovieReviewsPresenter`** class (Presenter): has the following methods:
    - **`[method] new_movie_review`**: object to data-bind **`MovieReview`** form in View
    - **`[method] movie_reviews`**: stores **`MovieReview Array`**
    - **`[method] movie_ratings`**: calculated from **`movie_reviews`** (average rating per movie)
      - **`require 'facets/array/average'`** to calculate movie average rating
    - **`[method] save:`** saves new movie review
  - **`MovieReviewsView`** class (View)
    - New movie review form and save button to save new movie reviews by reviewers:
      - reviewer **`entry`**
      - movie **`entry`**
      - rating **`spinbox(1, 10)`**
    - Movie rating table

Section 4
Advanced Data-Binding

Advanced Data-Binding

- Data-Binding Converters
- Data-Binding Hooks
- Computed Data-Binding
- Nested Data-Binding
- Indexed Data-Binding
- Keyed Data-Binding

Details at: Glimmer DSL for LibUI Data-Binding API

Data-Binding Converters

- Data-Binding Converters enable:
  - Transforming Model attributes (on read) before displaying in View
  - Transforming View properties before storing in Model (on write)
- **`on_read: -> (val) {...}`** : convert on read from Model to View
- **`on_write: -> (val) {...}`** : convert on write to Model from View

```
text <= [@counter, :count, on_read: ->(value) { "Click To Increment: #{value}" }]
```

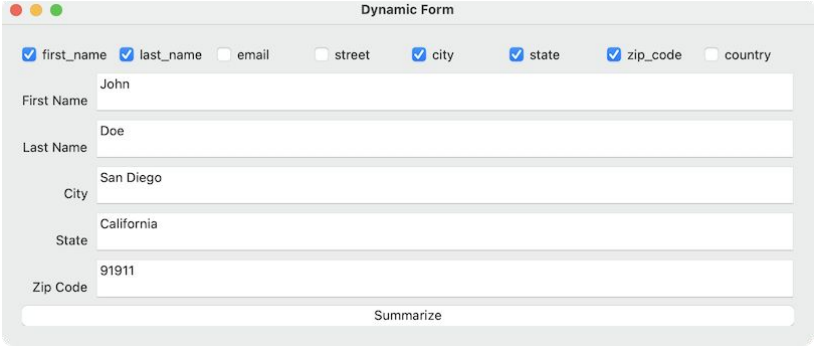
Click To Increment: 11

```
selected <=> [model, :selected_file,
              on_read: ->(f) {model.midi_files.index(f)},
              on_write: ->(i) {model.midi_files[i]}]
```

Data-Binding Converter Example

- Dynamic Form

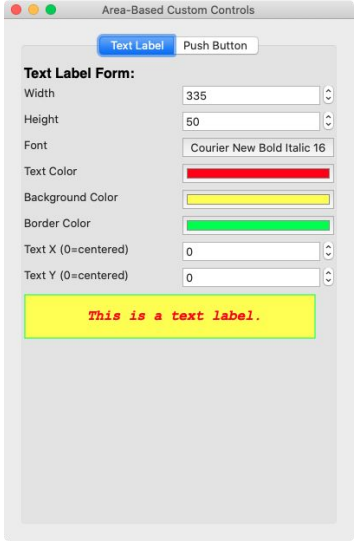
```
User::ALL_ATTRIBUTES.each do |attribute|
  checkbox(attribute.to_s) {
    checked <=> [@user, :customizable_attributes,
                 on_read: -> (attributes) { @user.customizable_attributes.include?(attribute) },
                 on_write: -> (checked_value) { @user.select_customizable_attribute(attribute, checked_value) }
                ]
  }
end
```



Data-Binding Hooks

- Data-Binding Hooks perform actions before and/or after copying data between View properties and Model attributes
- **`before_read: -> (val) {...}`** : do work before reading data from Model to View and before **`on_read`** converter is called
- **`after_read: -> (val) {...}`** : do work after reading data from Model to View and after **`on_read`** converter is called
- **`before_write: -> (val) {...}`** : do work before writing data to Model from View and before **`on_write`** converter is called
- **`after_write: -> (val) {...}`** : do work after writing data to Model from View and after **`on_write`** converter is called

Data-Binding Hook Examples




```
value <=> [self, :label_width, after_write: method(:rebuild_text_label)]

value <=> [self, :label_height, after_write: method(:rebuild_text_label)]

font <=> [self, :label_font_descriptor, after_write: method(:rebuild_text_label)]

color <=> [self, :label_text_color, after_write: method(:rebuild_text_label)]

color <=> [self, :label_background_fill, after_write: method(:rebuild_text_label)]

color <=> [self, :label_border_stroke, after_write: method(:rebuild_text_label)]
```

Computed Data-Binding

- Computed Data-Binding enables observing extra Model attributes that are dependencies of a computed Model attribute so that when any of the extra attributes change, the View is updated from the computed attribute accordingly.
- Examples:

```
text <= [order, :total, computed_by: :order_lines]

text <= [user, :salary, computed_by: [:dob, :position, :state]]
```

Nested/Indexed/Keyed Data-Binding

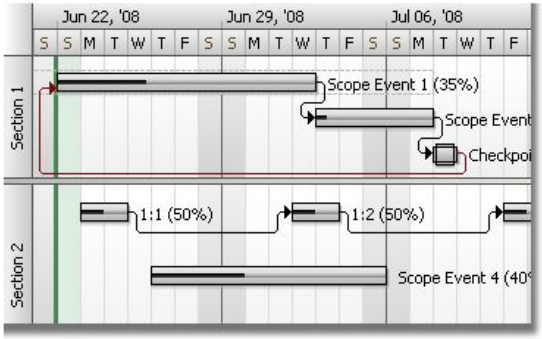
- Nested data-binding enables data-binding a View property to a nested Model attribute (e.g. **`user.address.street`**)
- Indexed data-binding enables data-binding a View property to an indexed Model attribute within an Array attribute (e.g. **`user.names[0]`**)
- Keyed data-binding enables data-binding a View property to a keyed Model attribute inside a Hash attribute (e.g. **`user.names[:nickname]`**)

Nested/Indexed/Keyed Data-Binding Examples

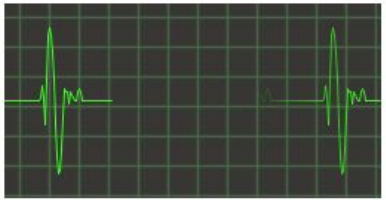
```
text <= [user, 'address.street']

text <= [user, 'names[0]']

text <= [user, 'addresses[0].city']

text <= [user, 'names[:nickname]']

text <= [user, 'addresses[:billing].city']
```

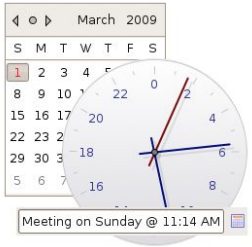
Section 5
Custom Components
(Bonus)

Custom Controls

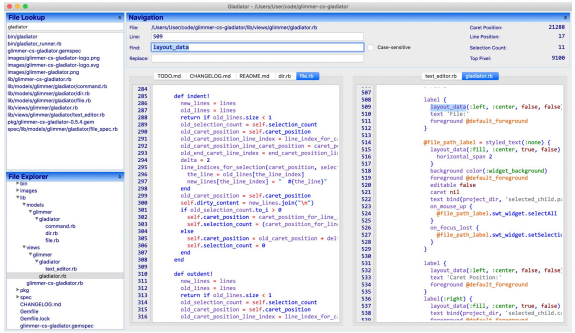


- Custom Controls
  - **`include Glimmer::LibUI::CustomControl`**
  - Accept **`options`** that can be passed as **`Hash`** arguments
  - Define **`body`** block with Glimmer GUI DSL
  - **`before_body`** block hook to setup variables
  - **`after_body`** block hook to register listeners
  - **`body_root`** API method returns root control
  - Useful for:
    - Specializing Controls
    - Aggregating Controls
    - Build brand new non-native controls with Area graphics

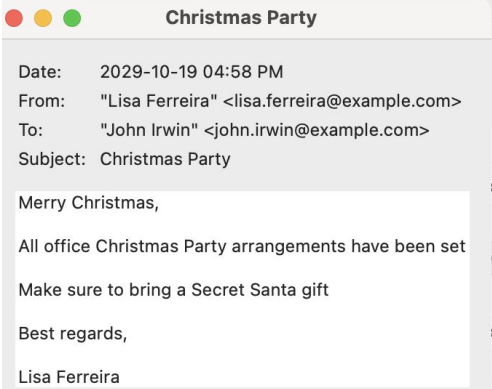




Custom Windows



- Custom Windows (aka Applications)
  - **`include Glimmer::LibUI::CustomWindow`**
  - Or **`include Glimmer::LibUI::Application`**
  - Custom Windows are Custom Controls with **`window`** as body root
  - Represent apps or reusable custom windows
  - Automatic implementation of **`.launch`** method



Section 5
Custom Components
(Bonus)
Exercises
(Follow on GitHub)

Section 8
Scaffolding

Glimmer Scaffolding

- **`glimmer`** command enables scaffolding apps, custom components (custom controls, custom windows, and custom shapes), and packaging gems:
  - **`glimmer scaffold[app_name]`**
  - **`glimmer scaffold:customcontrol[name,namespace]`**
  - **`glimmer scaffold:customshape[name,namespace]`**
  - **`glimmer scaffold:customwindow[name,namespace]`**
  - **`glimmer scaffold:gem:customcontrol[name,namespace]`**
  - **`glimmer scaffold:gem:customshape[name,namespace]`**
  - **`glimmer scaffold:gem:customwindow[name,namespace]`**

Application Scaffolding

- Application scaffolding enables automatically generating the directories/files of a new desktop GUI application that follows the MVC architecture and can be packaged as a Ruby gem that includes an executable script for running the app conveniently.
- Scaffolding ensures that Software Engineers follow the recommended Glimmer DSL for LibUI conventions and best practices.
- Application Scaffolding greatly improves Software Engineering productivity when building desktop applications.

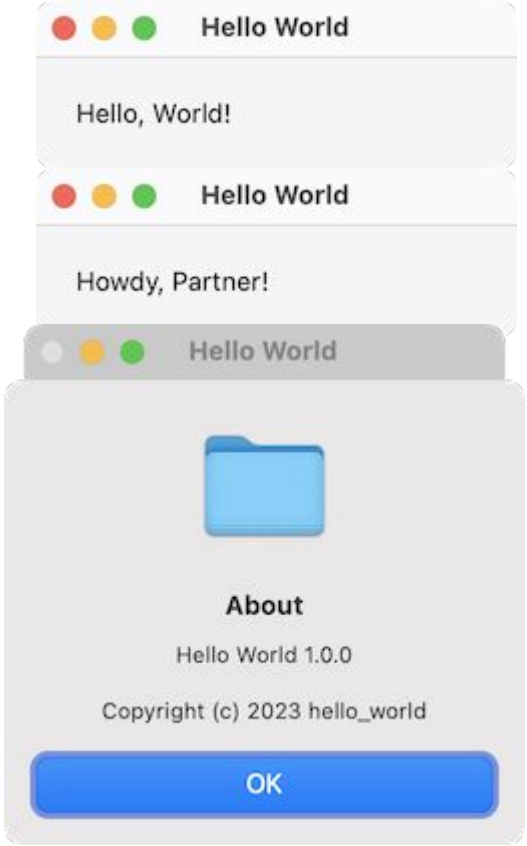
Application Scaffolding Instructions

- Run: **`glimmer "scaffold[app_name]"`**
- **`bin/app_name`** runs the scaffolded app (or **`glimmer run`**)
- App namespace is **`app/app_name.rb`**
- App entry point is **`app/app_name/launch.rb`**
- Views live under **`app/app_name/view`**
- Models live under **`app/app_name/model`**
- **`Rakefile`** includes tasks to:
  - Auto-generate gemspec from **`Gemfile`** (no need for manual **`gemspec`**)
  - Build application Ruby gem to distribute in Ruby ecosystem

Application Scaffolding Example

- **`glimmer "scaffold[hello_world]"`**
- **`bin/hello_world`** runs app

```
Created hello_world/.gitignore
Created hello_world/.ruby-version
Created hello_world/.ruby-gemset
Created hello_world/VERSION
Created hello_world/LICENSE.txt
Created hello_world/Gemfile
Created hello_world/Rakefile
Created hello_world/app/hello_world.rb
Created hello_world/app/hello_world/view/hello_world.rb
Created hello_world/app/hello_world/model/greeting.rb
Created hello_world/icons/windows/Hello World.ico
Created hello_world/icons/macosx/Hello World.icns
Created hello_world/icons/linux/Hello World.png
Created hello_world/app/hello_world/launch.rb
Created hello_world/bin/hello_world
```



Section 8
Scaffolding Exercises
(Follow on GitHub)

Summary

1. GUI Basics
2. MVC Software Architecture
3. MVP & Data-Binding
4. Advanced Data-Binding
5. Custom Components (Bonus)
6. Scaffolding (Bonus)

Hack Day at Main Stage - Ballroom South

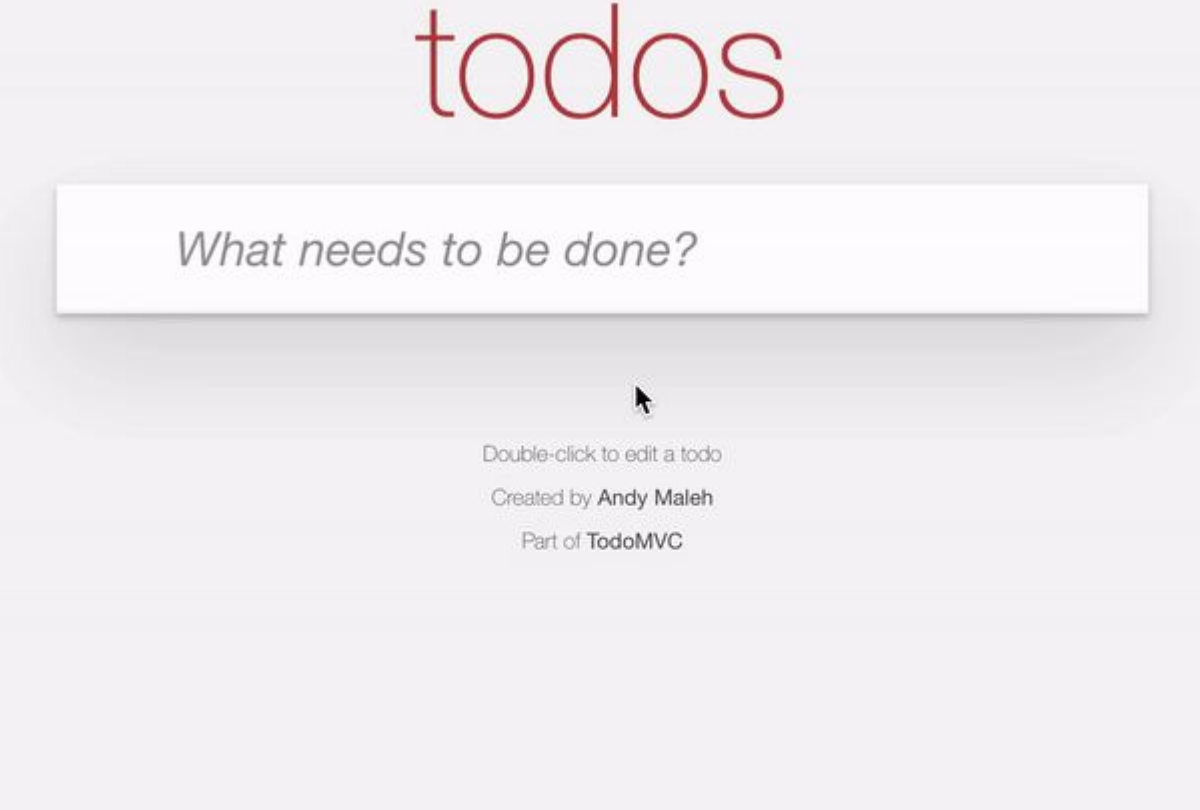
- Glimmer hacking & continued learning at Main Stage - Ballroom South 3:45p-5:45p (2h).
- Examples of things to hack on:
  - Build app that stores data in memory (in Ruby variables)
  - Build app that stores data in flat files (e.g. CSV files)
  - Build app that stores data with a relational database (e.g. SQLite)
  - Do workshop exercises if you missed workshop
  - Finish workshop exercises if you did not finish all exercises
  - Do additional exercises in RubyConf 2023 longer version of workshop
  - Build a board game or 2D game (e.g. Hangman).

Next Steps

- Check out other Glimmer GUI DSLs:
  - Glimmer DSL for SWT (100% feature complete and runs on JRuby supporting native Mac/Windows/Linux APP/PKG/DMG/EXE/MSI/DEB/RPM packaging options)
  - Glimmer DSL for WX (new DSL that runs on the 100% feature complete wxWidgets toolkit, which is better than LibUI for more serious app development; Glimmer DSL is not final yet)
  - Glimmer DSL for Web (all the awesomeness of Glimmer on the Web 🤯! Build Rails Frontend Web UIs with Glimmer Ruby code)
  - Other Glimmer DSLs at https://github.com/AndyObtiva/glimmer

Glimmer DSL for Web (Future of Rails Frontend Development 🤯)

https://sample-glimmer-dsl-web-rails7-app-black-sound-6793.fly.dev/



Glimmer DSL for Web (Future of Rails Frontend Development 🤯)

- Enables writing Ruby code in the Frontend of Rails applications
- Simplifies Frontend Development significantly over all JavaScript libraries like React, Angular, Ember, Vue, and Svelte due to the unique characteristics of Ruby
- Improves productivity/maintainability & cuts down delivery time and cost
- Enables writing JavaScript, HTML, & CSS in one language: Ruby (using Ruby-to-JavaScript-Transpiler, Opal, Fukuoka Ruby 2023 Winner)
- Supports the Rails vision of the One Person Framework by enabling the same person who builds the Rails Backend in Ruby to also build the Frontend.
- Supports Ruby code reuse from Backend
- Cuts down hiring by half as it enables all Ruby devs to be Fullstack devs.
- Fast enough (faster than React in a webpage I rewrote in my work app)

```
markup {
  div {
    section(class: 'todoapp') {
      new_todo_form(presenter: @presenter)

      todo_list(presenter: @presenter)

      todo_filters(presenter: @presenter)
    }

    todo_mvc_footer

    on_remove do
      @presenter.unsetup_filter_routes
    end
  }
}
```

todos

What needs to be done?

Double-click to edit a todo

Created by Andy Maleh

Part of TodoMVC

```
style {
  r('body, button, html') {
    margin 0
    padding 0
  }

  r('button') {
    _webkit_font_smoothing :antialiased
    _webkit_appearance :none
    appearance :none
    background :none
    border 0
    color :inherit
    font_family :inherit
    font_size 100.%
    font_weight :inherit
    vertical_align :baseline
  }

  r('.todoapp') {
    background '#fff'
    margin '130px 0 40px 0'
    position :relative
    box_shadow '0 2px 4px 0 rgba(0, 0, 0, 0.2)
  }
```

```ruby
class NewTodoInput < TodoInput
  option :presenter

  markup {
    input(placeholder: "What needs to be done?", autofocus: "") {
      value <=> [presenter.new_todo, :task]

      onkeyup do |event|
        presenter.create_todo if event.key == 'Enter' || event.keyCode == "\r"
      end
    }
  }

  style {
    todo_input_styles

    r(component_element_selector) {
      padding '16px 16px 16px 60px'
      height 65
      border :none
      background 'rgba(0, 0, 0, 0.003)'
      box_shadow 'inset 0 -2px 1px rgba(0,0,0,0.03)'
    }

    r("#{component_element_selector}::placeholder") {
      font_style :italic
      font_weight '400'
      color 'rgba(0, 0, 0, 0.4)'
    }
  }
end
```

Resources

- https://github.com/AndyObtiva/glimmer-dsl-libui
- https://github.com/AndyObtiva/glimmer-dsl-libui#supported-keywords
- https://github.com/AndyObtiva/how-to-build-desktop-applications-in-ruby
- https://bit.ly/rubyconf2024desktop
- https://bit.ly/rubyconf2023desktop
- https://github.com/AndyObtiva
- https://andymaleh.blogspot.com
- https://twitter.com/AndyObtiva
- https://www.youtube.com/@glimmer-dsl
- https://www.youtube.com/@montreal-rb
- https://github.com/AndyObtiva/glimmer-dsl-swt
- https://github.com/AndyObtiva/glimmer-dsl-wx
- https://github.com/AndyObtiva/glimmer-dsl-web

bit.ly/rubyconf2024desktop

Presentation Slides for How To Build Basic Desktop Applications in Ruby