Swinburne University of Technology

Using FXRuby and Ruby Sequel gem to write a to-do list application

Shubin Zhong
102872405
COS60006 Introduction to Programming
Tutor Mathew Wakefield
13 June 2020

Introduction

FXRuby is one of the libraries for developing graphical user interface (GUI) for ruby application. It's based on the FOX Toolkit and it's an open-source C++ library developed by Jeroen van der Zijp (Kanis, 2018). "Ruby Sequel gem is a simple, flexible, and powerful SQL database access toolkit for ruby" (Evans, 2020). These two libraries will be used in this tutorial.

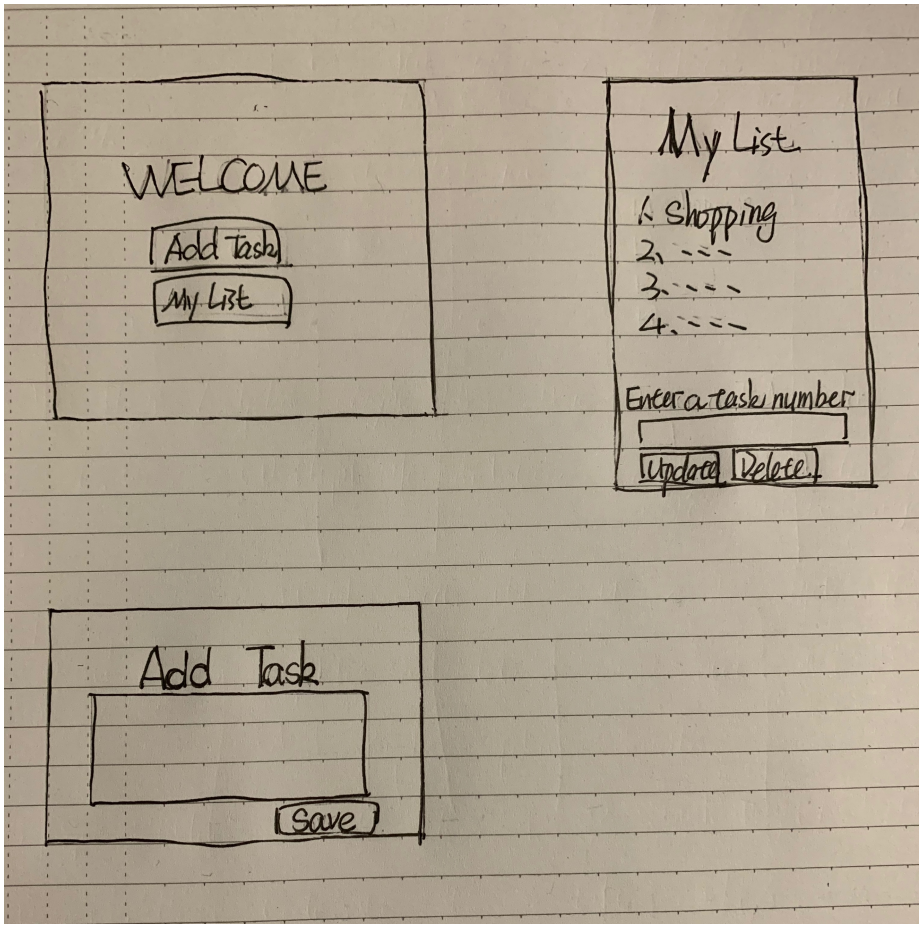
In this tutorial, I'm going to demonstrate using FXRuby and Ruby Sequel gem to write a to-do list application. Using the FXRuby to write a GUI for the to-do list application. And using a Ruby Sequel gem as a connector to connect this ruby application with a database. In this case, I will store the content of each task on the to-do list in the database.

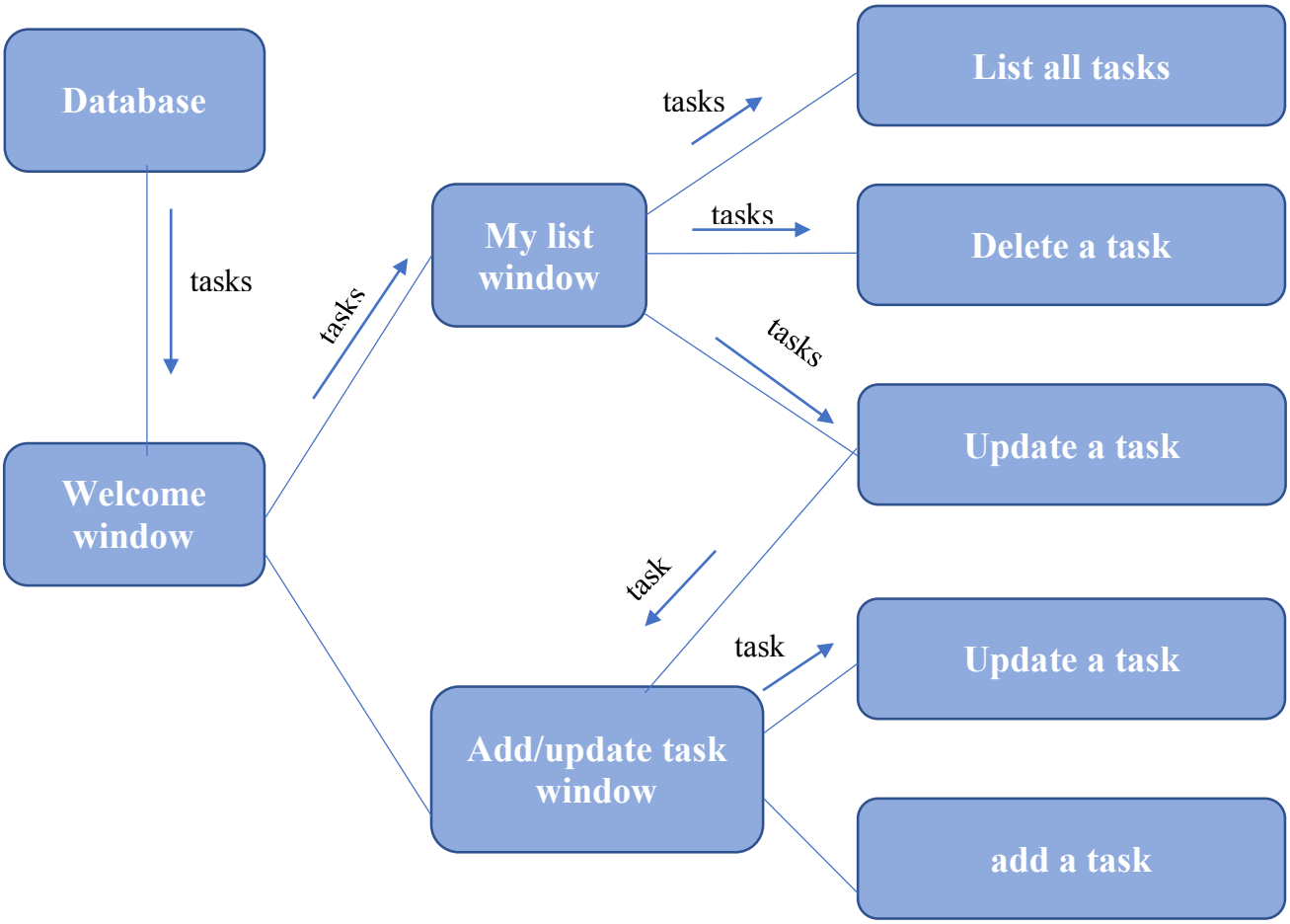
Installation

FXRuby has requirements for building it on your computer. For Linux, Windows and OS-X operating system, FXRuby runs with Ruby 2.2 version or newer than Ruby 2.2 version. Johnson (2010) provided more information on installing FXRuby on GitHub.

Before we installing the Sequel gem, we should have a database server. Then we install Ruby Sequel gem. It is easier than installing FXRuby. Entering this command line “gem install sequel” on the Terminal.

Structure chart of to-do list program





First, we draw a draft of our application. We create a tasks table in our database to store to do tasks. We design our to-do list will have 3 windows, one is a welcome window and there are 2 buttons on this window. These 2 buttons will link to their window respectively. On my list window, we will list all tasks on the window. We get all tasks from the tasks table and display it on the application. We also allow users to update or delete a task by entering the task number. This task number matches its id in the tasks table. If a user wants to update a task, we pass this task data to the update window. On add task window, we will allow users to create a task and store it in the database.

Creating a new ruby file in your computer and name it “todolist.rb”. Before we using FXRuby and Sequel, we should “require” them.

```
require "fox16"
require "sequel"
include Fox
```

- We start our program by requiring the “fox16” library. All the code we use to write GUI is related to the “fox16” library
- We will store our task contents in a “tasks” table in the database. So, we require the “sequel” library to connect our program with database.
- All FXRuby classes are defined under the FOX module, we normally use FXRuby classes by referring to their full name, for example, “FOX::FXApp”. “include FOX” works as “FOX::”, so we don’t need to precede every FXRuby class with “FOX::” prefix.

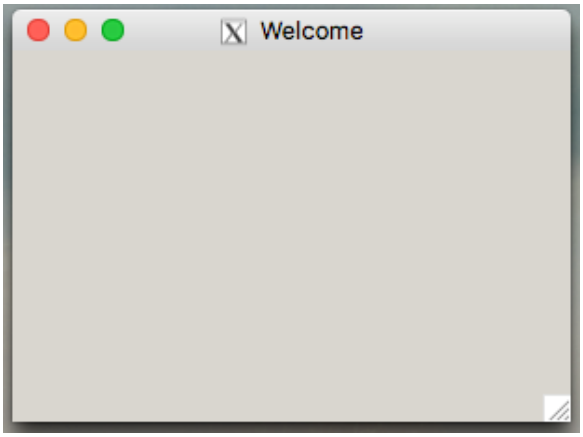
After requiring these libraries in our program, now we can write our application.

```ruby
class WelcomeWindow < FXMainWindow
    def initialize(app)
        super(app, "Welcome", :width => 300, :height => 200)
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end

end

if __FILE__ == $0
    FXApp.new do |app|
        WelcomeWindow.new(app)
        app.create
        app.run
    end
end
```

- We create an instance of FXApp class by **FXApp.new**, this instance is the core of our program.
- We need to create a main window for a Ruby application. In this tutorial, we create a custom window as a subclass of FXMainWindow class. And we initialize it by passing it an app parameter which we created on the last step. We also pass title, width and height these 3 arguments to it.

- We define a create method to make sure our window is created. Normally, the window we create is invisible until we place it on the screen. So, we use **show()** to place this window on the screen, and **PLACEMENT_SCREEN** means centring the window on the screen.
- We use **app.run** method to start this program.

When you run the above code, you will see this window on your screen.



Now we can add some widgets for our application. As we designed before, on the welcome window, there are 2 buttons on it. We use layout managers to arrange our widgets' positions and sizes. All these layout managers' first argument is their parent window, such as "self".

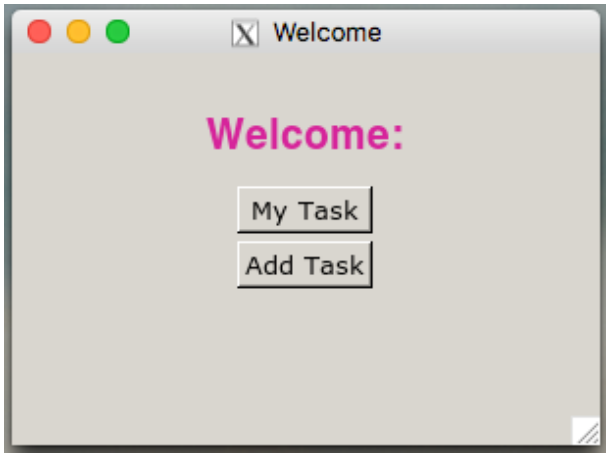
```
# draw welcome text
titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 30)
titleLabel = FXLabel.new(titleFrame, "Welcome:")
titleLabel.textColor = FXRGB(208,  32, 144)
titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

#draw selected button
buttonFrame = FXVerticalFrame.new(self, :opts => PACK_UNIFORM_WIDTH | LAYOUT_CENTER_X)
myTaskButton = FXButton.new(buttonFrame, "My Task")
addTaskButton = FXButton.new(buttonFrame, "Add Task")
```

- **FXHorizontalFrame** will arrange its child classes horizontally. In this case, its child class is titleLabel. We also give a hint LAYOUT_CENTER_X to it. It means putting this frame in the middle
- **FXLabel** to create a label which we can display some text on the window. We also can use its attributes ".textColor" and ".font" to change label text colour and font format
- **FXFont** to create a font format for label text. We define a labelFont function with an argument font weight. We pass this argument when we call this function

```
def labelFont(weight)
    font = FXFontDesc.new()
    font.encoding = FONTENCODING_DEFAULT
    font.face = "Helvetica"
    font.flags = 0
    font.setwidth = FONTSETWIDTH_WIDE
    font.size = 150
    font.slant = FONTSLANT_REGULAR
    font.weight = weight
    return(font)
end
```

- **FXVerticalFrame** will arrange its child classes vertically. In this case, its child class is the button. We give a display option to buttonFrame which is PACK_UNIFORM_WIDTH. This hint will set buttons in the same width and height. It looks like all the buttons "wear" the same uniform
- **:opts** is used to choose a layout hint for a class

Now, we finish writing our welcome window GUI. It looks like this:

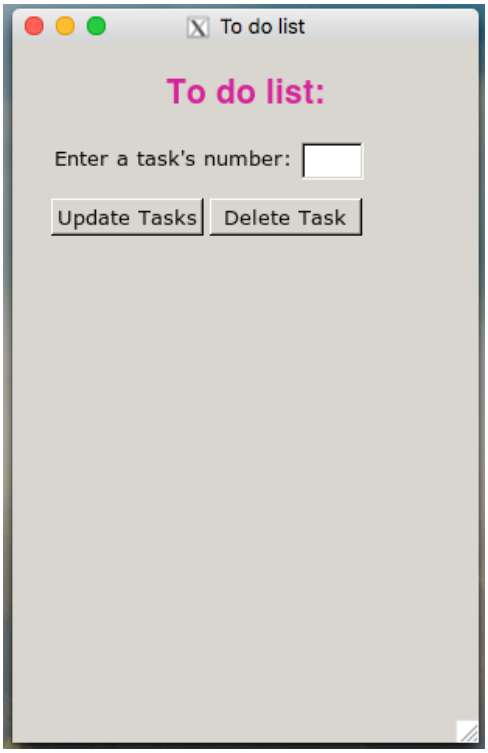


As we designed before, when users click my task button will go to to-do list window. So, we create to-do list window now. It is similar to create the welcome window.

- **FXTextField** to create a text field that users can enter some data

```
class ToDoListWindow < FXMainWindow
    def initialize(app)
        super(app, "To do list", :width => 300, :height => 450)
        titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 20)
        titleLabel = FXLabel.new(titleFrame, "To do list:")#JUSTIFY_CENTER_X
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        #choose a task
        chooseFrame = FXHorizontalFrame.new(self, :padTop => 10, :padLeft => 25)
        chooseLabel = FXLabel.new(chooseFrame, "Enter a task's number:")
        chooseTextField = FXTextField.new(chooseFrame, 4)
        enterID = chooseTextField.text
        puts("enter ID: " + enterID)

        buttonFrame = FXHorizontalFrame.new(self, :opts => PACK_UNIFORM_WIDTH, :padLeft => 25)
        updateTaskButton = FXButton.new(buttonFrame, "Update Tasks")
        deleteButton = FXButton.new(buttonFrame, "Delete Task", :opts => BUTTON_NORMAL)
    end


    def create
        super
        show(PLACEMENT_SCREEN)
    end

end
```

Because we haven't connected these two windows, we should change this block of codes from **WelcomeWindow.new(app)** to **ToDoListWindow.new(app)**. You can see the to-do list window will display on the screen like this:



Now we can connect these two windows. In FXRuby, we use connect method to connect user actions with a function or procedure in our program. In this case, we will connect a button click action with a change window procedure.

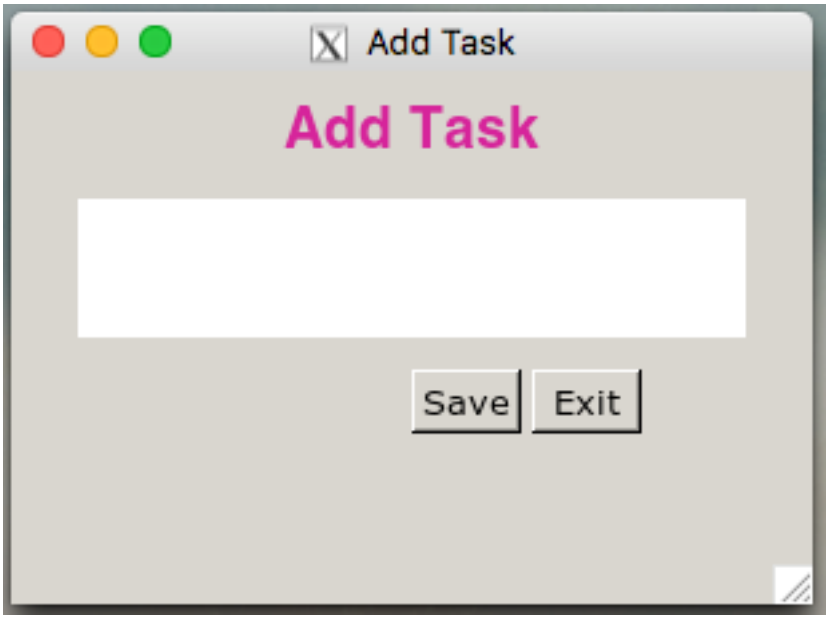
FXButton will send a SEL_COMMAND message to its target when it is clicked to its target. In this case, its target is its parent window and ask its parent window to close. Then open the to-do list window.

```
class WelcomeWindow < FXMainWindow
    def initialize(app)
        super(app, "Welcome", :width => 300, :height => 200)
        # draw welcome text
        titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 30)
        titleLabel = FXLabel.new(titleFrame, "Welcome:")
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        #draw selected button
        buttonFrame = FXVerticalFrame.new(self, :opts => PACK_UNIFORM_WIDTH | LAYOUT_CENTER_X)
        myTaskButton = FXButton.new(buttonFrame, "My Task")
        addTaskButton = FXButton.new(buttonFrame, "Add Task")

        myTaskButton.connect(SEL_COMMAND) {
            self.destroy
            toDoListWindow = ToDoListWindow.new(app)
            toDoListWindow.create
        }
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end

end
```

Now we can create the add task window. And connect this window with the welcome window. There is a slight difference between FXTextField and FXText. FXText supports multiple lines of text while FXTextField is single-line text entry.

```ruby
class AddTaskWindow < FXMainWindow
    def initialize(app)
        super(app, "Add Task", :width => 300, :height => 200)

        # draw a text field title
        titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 10)
        titleLabel = FXLabel.new(titleFrame, "Add Task")
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        # #draw a text field
        addTaskFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_FILL_X, :padLeft => 25, :padRight => 25)
        taskTextField = FXText.new(addTaskFrame, :opts => LAYOUT_FILL_X | LAYOUT_FILL_COLUMN)

        #draw button
        buttonFrame = FXHorizontalFrame.new(self, :opts => PACK_UNIFORM_WIDTH, :padLeft => 150)
        saveButton = FXButton.new(buttonFrame, "Save", :opts => BUTTON_NORMAL)
        exitButton = FXButton.new(buttonFrame, "Exit", :opts => BUTTON_NORMAL)
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end
end
```

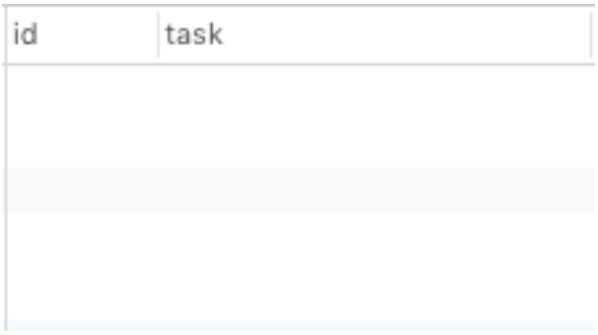
When you run this program, you will see this window:



So far, we have done the GUI for our to-do list application. What more, we have connected these three windows. As we designed before, we will store tasks in a tasks table in the database. Now, we contact our database using Ruby Sequel gem.

First, we need to connect our database and create a tasks table in the database. In this case, I create a table with an id as a primary key and a task column. I define a model class Tasks to wrap the dataset. This was introduced by Castello (2020).

```ruby
#connect to a database
DB = Sequel.mysql2(host: "localhost", user: "root", password: "123456", database: "todolist")

class Tasks < Sequel::Model(DB[:tasks])
end
```

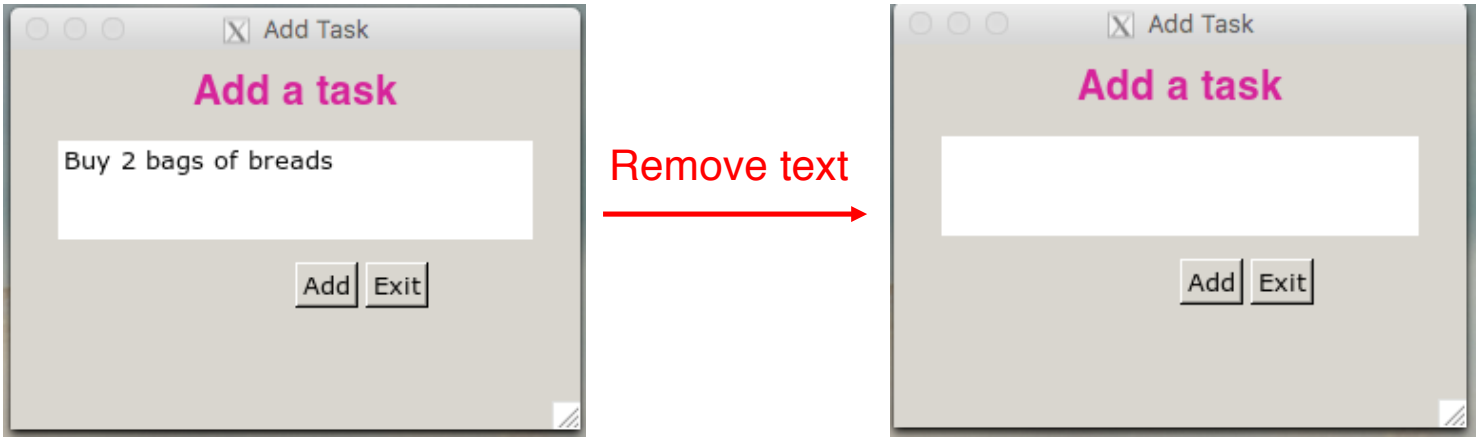


the tasks table

After we connect to the database, we can create a task and store it to the tasks table. Adding these codes into add task window.

```
#save task
saveButton.connect(SEL_COMMAND) {
    taskContent = Tasks.new
    taskContent[:task] = taskTextField.text
    taskContent.save
    taskTextField.removeText(0, taskTextField.length)
}
```

When the user clicks the save button, the program will create an instance of Tasks, get the user input and assign it to the taskContent[:task]. And taskContent.save will save this data to the tasks table. The removeText attribute used to remove the user input on the text entry field.



id	task
32	Buy 2 bags of breads

Task stored in the tasks table

Now we can add tasks and store them in the tasks table. The next step is printing these tasks on the to-do list window. Add these codes in the ToDoList window. Using a while loop to print out all tasks. For more information on how to using Ruby Sequel gem, Evans (2020) is an expert on it.

```
#display tasks list
length = Tasks.dataset.all.size
index = 0
while index < length
    id = Tasks.dataset.all[index][:id]
    task = Tasks.dataset.all[index][:task]
    taskText = id.to_s + ". " + task
    taskFrame = FXHorizontalFrame.new(self, :padLeft => 25)
    task = FXLabel.new(taskFrame, taskText)
    index += 1
end
```

The last thing we need to do is switching the add task window to the update window. As we provide an update function for users and the update task window share the same window with add task window. We pass the updated task data to update task window. We add these codes in the to-do list window.

```
deleteButton.connect(SEL_COMMAND){
    deleteTask = Tasks.first(id: chooseTextField.text)
    # puts deleteTask[:task]
    deleteTask.delete
    toDoListWindow = ToDoListWindow.new(app)
    toDoListWindow.create
}

updateTaskButton.connect(SEL_COMMAND){
    taskContent = Tasks.first(id: chooseTextField.text)
    puts taskContent
    self.destroy
    saveTaskWindow = SaveTaskWindow.new(app, "update", taskContent)
    saveTaskWindow.create
}
```

As we pass the display option and a task data to the add task window when we initialize this window. Adding these codes in the add task window. The display option used to determine the current window. The task data passed from the to-do list window used for updating a task so that we can prefill the task content in the text entry field. I also changed the save procedure a little bit. If the current window is the update window, we just store the updated task content while the add task window will create a new instance of Tasks.

```
if (displayOption == "update")
    taskTextField.text = taskContent[:task]
end
#draw button
buttonFrame = FXHorizontalFrame.new(self, :opts => PACK_UNIFORM_WIDTH, :padLeft => 150)
saveButton = FXButton.new(buttonFrame, saveBtnText, :opts => BUTTON_NORMAL)
exitButton = FXButton.new(buttonFrame, "Exit", :opts => BUTTON_NORMAL)

#save task
saveButton.connect(SEL_COMMAND) {
    if (displayOption == "update")
        taskContent[:task] = taskTextField.text
        taskContent.save
    else
        taskContent = Tasks.new
        taskContent[:task] = taskTextField.text
        taskContent.save
    end
    taskTextField.removeText(0, taskTextField.length)
}
```

Here is the final code.

```ruby
require "fox16"
require "sequel"
include Fox
#connect to a database
DB = Sequel.mysql2(host: "localhost", user: "root", password: "123456", database: "todolist")

class Tasks < Sequel::Model(DB[:tasks])
end

class WelcomeWindow < FXMainWindow
    def initialize(app)
        super(app, "Welcome", :width => 300, :height => 200)
        packer = FXPacker.new(self, :opts => LAYOUT_FILL)
        # draw welcome text
        titleFrame = FXHorizontalFrame.new(packer, :opts => LAYOUT_CENTER_X, :padTop => 30)
        titleLabel = FXLabel.new(titleFrame, "Welcome:")
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        #draw selected button
        buttonFrame = FXVerticalFrame.new(packer, :opts => PACK_UNIFORM_WIDTH | LAYOUT_CENTER_X)
        myTaskButton = FXButton.new(buttonFrame, "My Task")
        addTaskButton = FXButton.new(buttonFrame, "Add Task")

        #save task
        myTaskButton.connect(SEL_COMMAND) {
            self.destroy
            toDoListWindow = ToDoListWindow.new(app)
            toDoListWindow.create
        }

        addTaskButton.connect(SEL_COMMAND) {
            self.destroy
            saveTaskWindow = SaveTaskWindow.new(app)
            saveTaskWindow.create
        }
    end

    def create
        super
        show(PLACEMENT_SCREEN)
    end

end
```

```ruby
class ToDoListWindow < FXMainWindow
    def initialize(app)
        super(app, "To do list", :width => 300, :height => 450)
        self.connect(SEL_CLOSE, method(:on_close))

        #list tasks
        titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 20)
        titleLabel = FXLabel.new(titleFrame, "To do list:")#JUSTIFY_CENTER_X
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        length = Tasks.dataset.all.size
        index = 0
        while index < length
            id = Tasks.dataset.all[index][:id]
            task = Tasks.dataset.all[index][:task]
            taskText = id.to_s + ". " + task
            taskFrame = FXHorizontalFrame.new(self, :padLeft => 25)
            task = FXLabel.new(taskFrame, taskText)
            index += 1
        end

        #choose a task
        chooseFrame = FXHorizontalFrame.new(self, :padTop => 10, :padLeft => 25)
        chooseLabel = FXLabel.new(chooseFrame, "Enter a task's number:")
        chooseTextField = FXTextField.new(chooseFrame, 4)
        enterID = chooseTextField.text
        puts("enter ID: " + enterID)

        buttonFrame = FXHorizontalFrame.new(self, :opts => PACK_UNIFORM_WIDTH, :padLeft => 25)
        updateTaskButton = FXButton.new(buttonFrame, "Update Tasks")
        deleteButton = FXButton.new(buttonFrame, "Delete Task", :opts => BUTTON_NORMAL)

        deleteButton.connect(SEL_COMMAND){
            deleteTask = Tasks.first(id: chooseTextField.text)
            # puts deleteTask[:task]
            deleteTask.delete
            toDoListWindow = ToDoListWindow.new(app)
            toDoListWindow.create
        }

        updateTaskButton.connect(SEL_COMMAND){
            taskContent = Tasks.first(id: chooseTextField.text)
            puts taskContent
            self.destroy
            saveTaskWindow = SaveTaskWindow.new(app, "update", taskContent)
            saveTaskWindow.create
        }
    end
```

```
    def create
        super
        show(PLACEMENT_SCREEN)
    end

    def on_close(sender, sel, event)
        q = FXMessageBox.question(app, MBOX_YES_NO, "Exit", "Are you sure to exit?")
        if (q == MBOX_CLICKED_YES)
          getApp().exit(0)
        end
    end
end

def labelFont(weight)
    font = FXFontDesc.new()
    font.encoding = FONTENCODING_DEFAULT
    font.face = "Helvetica"
    font.flags = 0
    font.setwidth = FONTSETWIDTH_WIDE
    font.size = 150
    font.slant = FONTSLANT_REGULAR
    font.weight = weight
    return(font)
end
```

```ruby
class SaveTaskWindow < FXMainWindow
    def initialize(app, displayOption = "add", taskContent = nil)
        if (displayOption == "add")
            appTitle = "Add Task"
            saveBtnText = "Add"
            title = "Add a task"
        else
            appTitle = "Update Task"
            saveBtnText = "Update"
            title = "Update a task"
        end
        super(app, appTitle, :width => 300, :height => 200)

        # draw a text field title
        titleFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_CENTER_X, :padTop => 10)
        titleLabel = FXLabel.new(titleFrame, title)
        titleLabel.textColor = FXRGB(208,  32, 144)
        titleLabel.font = FXFont.new(getApp(), labelFont(FONTWEIGHT_BOLD))

        # #draw a text field
        addTaskFrame = FXHorizontalFrame.new(self, :opts => LAYOUT_FILL_X, :padLeft => 25, :padRight => 25)
        taskTextField = FXText.new(addTaskFrame, :opts => LAYOUT_FILL_X | LAYOUT_FILL_COLUMN)

        if (displayOption == "update")
            taskTextField.text = taskContent[:task]
        end
        #draw button
        buttonFrame = FXHorizontalFrame.new(self, :opts => PACK_UNIFORM_WIDTH, :padLeft => 150)
        saveButton = FXButton.new(buttonFrame, saveBtnText, :opts => BUTTON_NORMAL)
        exitButton = FXButton.new(buttonFrame, "Exit", :opts => BUTTON_NORMAL)

        #save task
        saveButton.connect(SEL_COMMAND) {
            if (displayOption == "update")
                taskContent[:task] = taskTextField.text
                taskContent.save
            else
                taskContent = Tasks.new
                taskContent[:task] = taskTextField.text
                taskContent.save
            end
            taskTextField.removeText(0, taskTextField.length)
        }

        exitButton.connect(SEL_COMMAND) {
            self.destroy
            toDoListWindow = ToDoListWindow.new(app)
            toDoListWindow.create
        }
    end
```

```
    def create
        super
        show(PLACEMENT_SCREEN)
    end

end

if __FILE__ == $0
    FXApp.new do |app|
        WelcomeWindow.new(app)
        app.create
        app.run
    end
end
```

Reference:

Castello, 2020, *How to Use The Ruby Sequel Gem (With Examples)*, RubyGuides, viewed 13 June 2020, <https://www.rubyguides.com/2019/06/ruby-sequel-orm/>

Evans, 2020, *sequel/README.rdoc*, GitHub, viewed 12 June 2020, <https://github.com/jeremyevans/sequel/blob/master/README.rdoc>

Evans, 2020, *Sequel: The Database Toolkit for Ruby*, Sequel, viewed 13 June 2020, <http://sequel.jeremyevans.net/rdoc/>

Hibbard, 2012, *An Introduction to FXRuby*, Sitepoint, viewed 13 June 2020, <https://www.sitepoint.com/an-introduction-to-fxruby/>

Johnson, 2010, *Home*, GitHub, viewed 13 June 2020, <https://github.com/lylejohnson/fxruby/wiki>

Kanis, 2018, *fxruby/README.rdoc*, GitHub, viewed 12 June 2020, <https://github.com/larskanis/fxruby/blob/1.6/README.rdoc>