XojoPi

Programming the Raspberry Pi with Xojo

Friday, March 24, 2017

XojoPi

Table of Contents

Programming the Raspberry Pi with Xojo

In this book you'll learn how to set up your Raspberry Pi so you can make your own apps for it using Xojo. With Xojo you can easily make fun text, GUI and web apps for the Raspberry Pi.

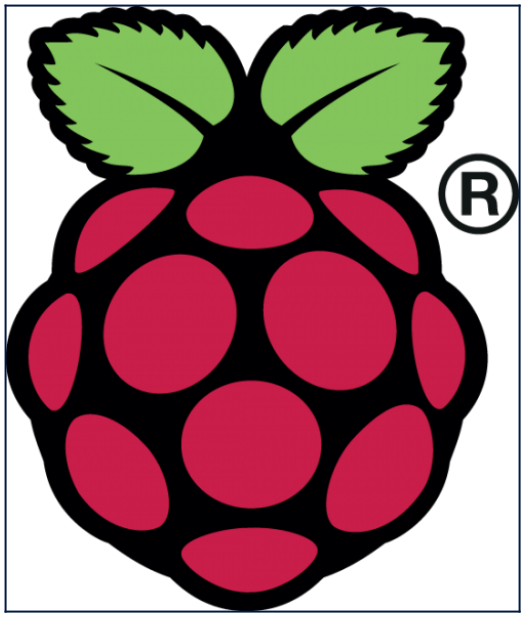
Xojo is an integrated development tool, but it is also a programming language. Xojo builds on languages such as Visual Basic and Java to provide the fastest, easiest and most fun way for you to make your own apps for the Raspberry Pi. Xojo uses safe programming patterns (strong data typing, for example), is object-oriented and has modern programming features such as introspection, extension methods and delegates. Programming with Xojo is fast, easy and most importantly, fun!

Although maybe you haven't heard of it before, Xojo has been around for many years. Since 1998, in fact. Xojo was originally called REALbasic, then eventually Real Studio, but the programming language remains largely the same.

Xojo feels familiar to programmers who have used other languages such as Visual Basic and Java because it uses a similar object-oriented programming model, with similar data types and constructs. Xojo is also friendly to new programmers. A big problem with most programming languages is that they are overcomplicated and overwhelm those new to them. Xojo has a powerful, integrated code editor with auto-complete that makes it easy to learn the language. The Xojo IDE is also incredibly easy to use, making experimentation (one of the best ways to learn) fast and fun.

Xojo is a great programming language for creating all types of apps for your Pi, from simple "hello, world" that teach you how to program to apps that can control hardware connected to the Raspberry Pi. We can't wait to see what you create with Xojo and your Raspberry Pi!

Table of Contents



About Raspberry Pi

First introduced in 2012, the Raspberry Pi is a tiny computer about the size of a deck of playing cards. It was created by the Raspberry Pi Foundation with the goal of making computing more accessible to everyone.

The Raspberry Pi itself is sometimes referred to as a "single-board" computer and although it is truly tiny, it packs quite a lot of power. There are currently several models of Raspberry Pi, all of which use some variant of the ARM CPU that is often seen in smartphones:

- Raspberry Pi (not compatible with Xojo)
- **Raspberry Pi 2** (compatible with Xojo)
- **Raspberry Pi 3** (compatible with Xojo)
- Raspberry Pi Zero (not compatible with Xojo)

The original Raspberry Pi is no longer available. Its relatively slow and underpowered CPU make it unsuitable for use with Xojo.

The Raspberry Pi 2 is a much faster version of the original Pi that shares the same ports and general configuration. It sells for around $35 (USD).

In 2016, the Rasperry Pi 3 was introduced as a faster version of the Pi 2 with built-in wifi and Bluetooth. It also sells for $35 (USD).

The Raspberry Pi Zero is an even tinier computer (about the size of a stick of gum) that is approximately the speed of the original Raspberry Pi. It's weaker CPU makes it unsuitable for use with Xojo so will not be discussed in this guide.

As of March 2017, the Raspberry Pi Foundation has said that over 12.5 million of the various models have been sold.

What Can You Do With It?

The Raspberry Pi is a real computer, so you can do whatever you want with it! Most people typically install a version of Linux on it (Raspbian is most common). After doing so you can run and install lots of apps and even create your own apps for it. You'll learn about how you can use Xojo to do this in a bit.

In addition to creating apps, the Raspberry Pi makes it easy to interface hardware devices to it. In particular, the Raspberry Pi has a General Purpose Input/Output (GPIO) port that makes it possible to directly wire your own hardware to it.

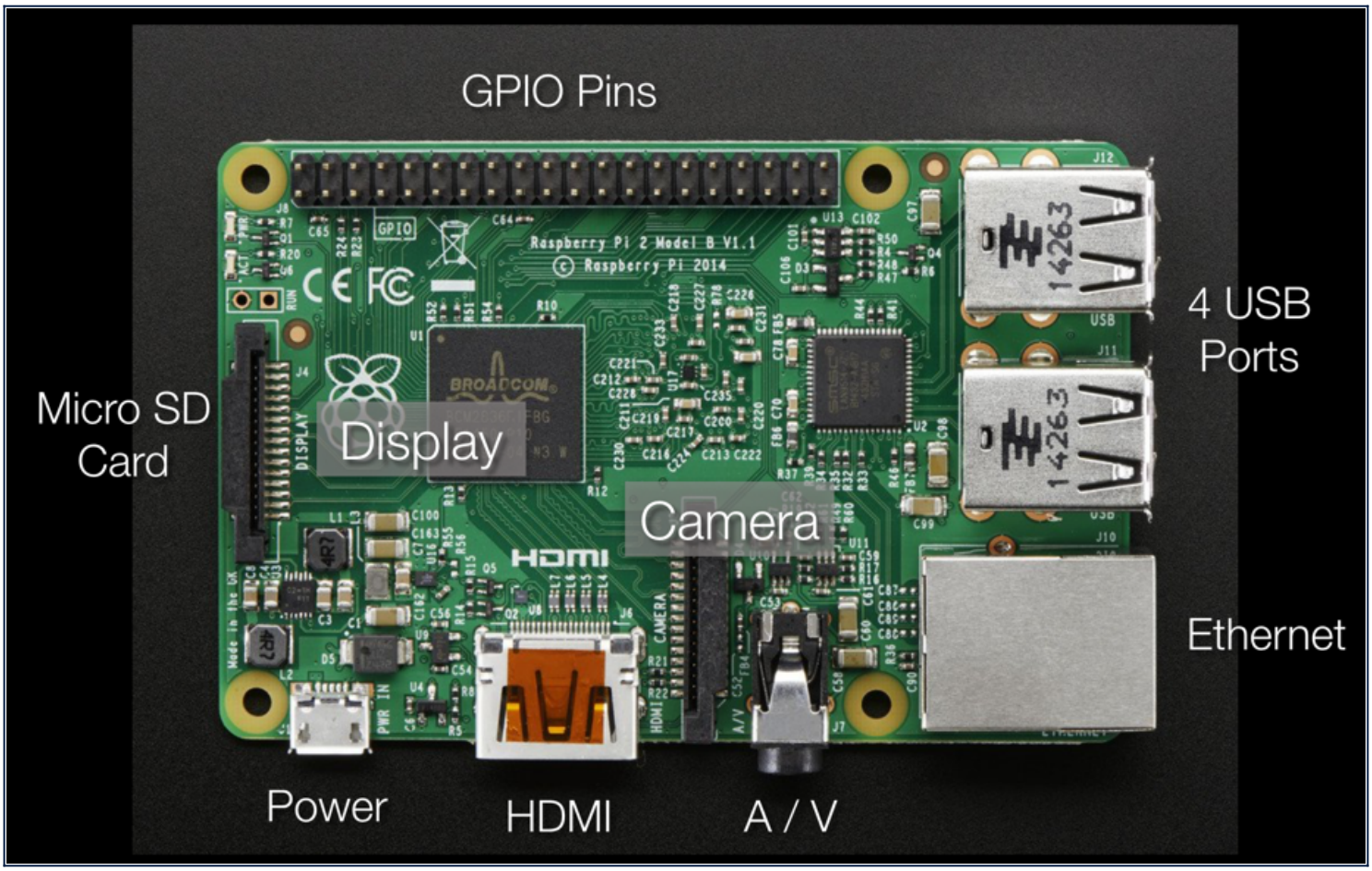
The Raspberry Pi also has relatively low power requirements, so it can be run on batteries or be left on 24/7 without worrying about excessive electricity usage.

The Pi has a few limitations over a standard computer that are helpful to keep in mind. First, it uses an SD card for file storage in place of a hard drive or SSD. This means that file access can be noticably slower than a typical computer. In addition, the Pi has a relatively small amount of RAM that cannot be upgraded (1GB), which somewhat limits the size and scope of apps that you will want to run on it.

The Hardware

Xojo is compatible with the Raspberry Pi 2 and Pi3 models.

Both the Raspberry Pi 2 and Raspberry Pi 3 models share similar hardware. Below is a picture of the Raspberry Pi 2 hardware with the various ports and interfaces labeled:



GPIO Pins

The Raspberry Pi has a 40-pin connector that is referred to as the GPIO (General Purpose Input/Output). With these pins you can connect hardware devices that you build directly to the Pi so you can control them using software that you make with Xojo.

The GPIO pins and their usage are described in the upcoming chapter: Interfacing Hardware with GPIO.

USB Ports

The Raspberry Pi has four USB ports that you can use to connect things such as a mouse, keyboard, wifi dongle (if necessary), external drive, etc.

Ethernet Port

The Ethernet port is used to directly connect the Pi to a wired network. This can provide faster network access than

using wifi.

Audio/Visual Port

The AV port is a 1/8" (3.5mm) jack that can toggle between outputting composite video or stereo audio.

HDMI Port

The HDMI port can be used to connect your Pi to a display or television. It also outputs audio.

Power Port

The power port is a micro-USB port that is only used to provide power to the Pi.

Micro SD Card Slot

The microSD card slot is actually underneath the Pi board. The Pi boots from the microSD card using the OS you have installed on it.

Display Port

The display port is used to connect Pi-specific display devices such as LCD displays or touchscreen displays.

Camera Port

The camera port is used to connect Pi-specific camera devices, such as webcams.

Where to Purchase

Xojo recommends you purchase your Raspberry Pi as a kit through out partner, CanaKit. With The Complete Starter Kit you get the Pi itself, a case, power supply, heat sink and HDMI cable. But more importantly you also get a coupon code for a free license of the Raspberry Pi edition of Xojo. This edition lets you create console (text based apps) for your Raspberry Pi.

Setting up Your Raspberry Pi

Installing Raspbian

For use with Xojo, you will want your Pi to use the Raspbian operating system. Raspian is a version of Debian Linux that has been optimized for the Pi. If you purchased a Pi kit, then it may have already come with an SD card that has Raspbian pre-installed. If so, you can just insert the SD card.

If you are setting things up yourself, then you'll want to download Raspbian and install it onto your SD card. A 32GB card is a reasonable size that gives you plenty of space to install other apps.

You can download the the latest version of Raspbian from the official Raspberry Pi site:

https://www.raspberrypi.org/downloads/raspbian/

You should download the "RASPBIAN JESSIE WITH PIXEL" file.

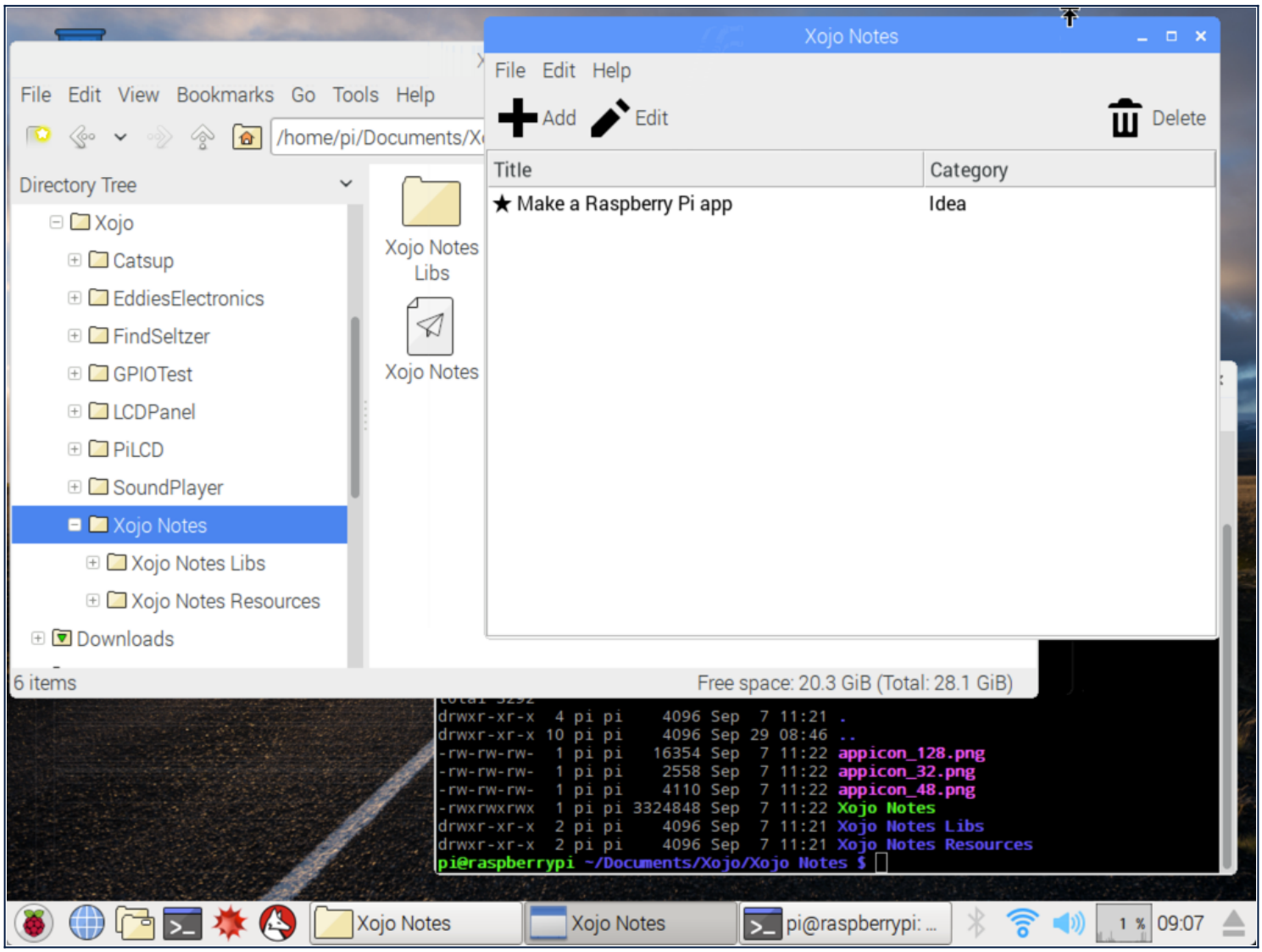
Installing the operating system onto the SD card varies by the operating system of the computer you are currently using. For specific steps you should again refer to the official Raspberry Pi docs for your OS:

- Windows
- Mac
  - You might also want to try the simpler ApplePi Baker app to help make SD cards.
- Linux

Once your card is all set up, you can insert it into the Raspberry Pi. At this point you should also connect a display (an HDMI TV works well), USB keyboard and USB mouse. Now you can connect the power to boot the Pi.

The Pi does not have a power switch. It turns on automatically when connected to power.

Follow the on-screen instructions to install Raspbian. When it is finished, your Pi will boot to the Pixel desktop. This screenshot shows the Pixel desktop with the status panel moved from its default at the top of the screen to the bottom:



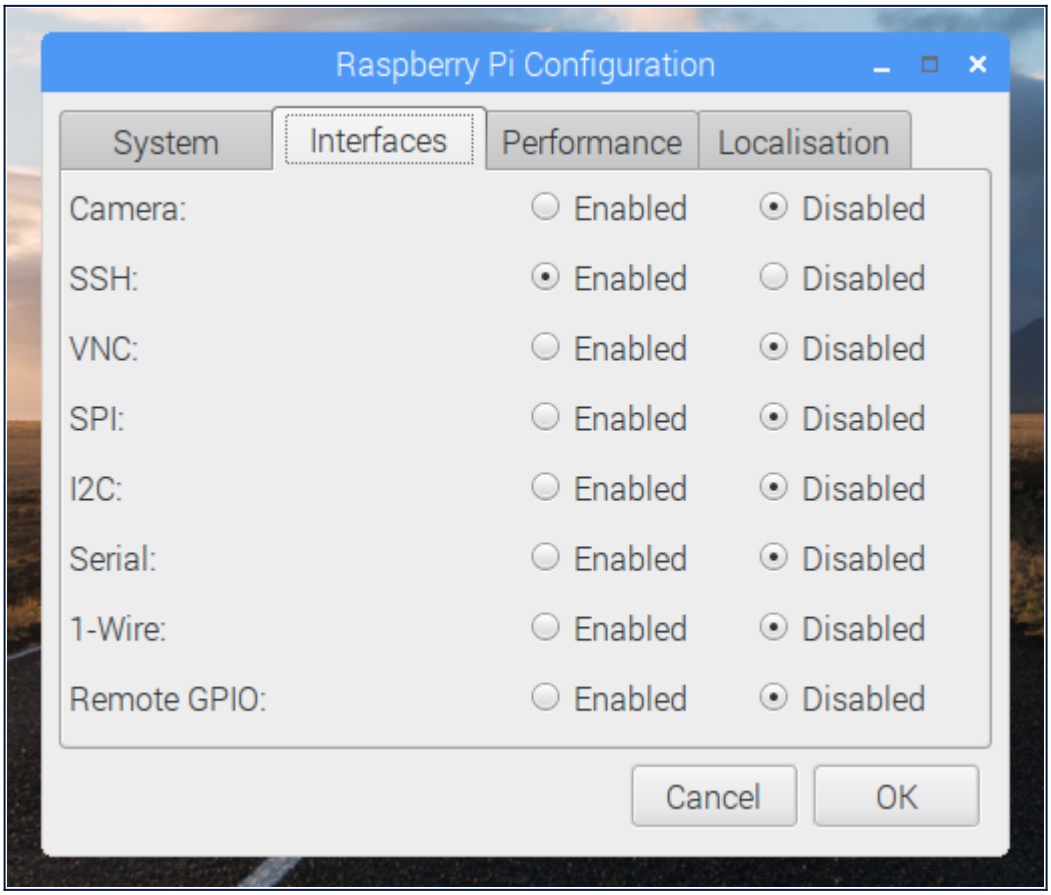
Be sure to play around with the Pi to get a feel for how it works. Be sure to try out the pre-installed apps.

Connect to Network

At this point, you should make sure your Pi is connected to your network (usually wi-fi). If you were not prompted to connect to wi-fi, click the wi-fi icon in the main status bar and enter your network connection information. Once you are connected to the network, make a note of the IP address. This is an example of an IP address you might see on your home network: 10.0.1.11.

Keep in mind that this IP address could change when your Pi reboots depending on how your wi-fi router hands out IP addresses. Unless your Pi is offline for a long period of time, it will probably keep the same IP address. If you need to find the IP address, you can always boot up again while connected to a display or use a LAN scanning tool to find your Pi on the network after it auto-connects.

Next you should go to the Pi Configuration (Pi menu->Preferences->Raspberry Pi Configuration) to enable connections using both SSH and VNC. This will allow you to remotely connect to the Pi desktop.



Connecting Remotely

To be able to use your Pi with Xojo, you'll want to be able to connect to it remotely from your main computer. This allows you to transfer your apps to it so that you can run and test them. You'll want to make sure your Pi is powered on an connected to the network. You do not need to leave the display, keyboard or mouse connected.

SSH

The first method you can use to connect remote is something called "ssh". This standards for "secure shell" and provides a text-based way for you to connect to the Pi. To connect using ssh, you simply start a terminal (or command shell) on your computer and type the command to connect using ssh. For example on macOS you would use this command:

```
ssh pi@10.0.1.11
```

You'll be prompted for the password for the "pi" user, which is "raspberry" if you haven't changed it.

When you are connected using ssh, the terminal commands now work as if you are running terminal directly on the Pi. You can use commands such as "ls" to list the files in a folder and the "cd" command to change to different folders.

You can run Xojo console apps that have been copied to the Pi by switching to the folder containing the app and typing its name like this:

```
./MyApp
```

SFTP

The next thing you'll want to be able to do is to copy files to your Pi from your main computer. This is how you get your finished apps that you make with Xojo onto the Pi.

To do this, you will use somethingn called SFTP or "Secure File Transfer Protocol". There are lots of free and paid SFTP apps that you can use for this. Common ones include FileZilla and CyberDuck.

With your SFTP app, add a connection for your Pi and use the same credentials that you used to connect using ssh. Once you are connected you'll be able to see the files on the Pi. By using the SFTP app you can select (or drag) files from your main computer and they will be copied to the Pi. Keep in mind that copying files is not instantaneous and it may take several seconds for files to copy depending on both their sizes and the speed of the SD card you are using on the Pi. In most cases files should transfer in under 30 seconds.

VNC Screen Sharing

The final way to connect to your Pi is to use VNC screen sharing. This lets you interact with the Pi desktop from within a window on your main computer. Any VNC client works, such as RealVNC. On macOS you can use the built-in screen sharing that is available from the Finder (Select Go->Connect to Server).

When prompted, enter the connection information. The URL is typically:

```
vnc://10.0.1.11:5901
```

Because you are connecting to the UI over the network, it won't be as fast as when you have the Pi directly connected to an HDMI display, but this does make it easier to use the Pi without having to keep it connected to its own keyboard, mouse and display.

Now that your Pi is set up and you know how to connect to it, you are ready to start learning about how you can use Xojo to make your own Pi apps.

Xojo Overview

The sections in this guide will give you an overview of Xojo, its programming language and some useful framework features. To learn about all the features that Xojo has to offer, be sure to read the complete User Guide. If you have never done any programming, you may also find the Introduction to Programming with Xojo book useful.

As mentioned earlier, Xojo is an integrated tool, but it is also a programming language. It can make many types of apps for Raspberry Pi (but also for Windows, MacOS, Linux, iOS, and web).

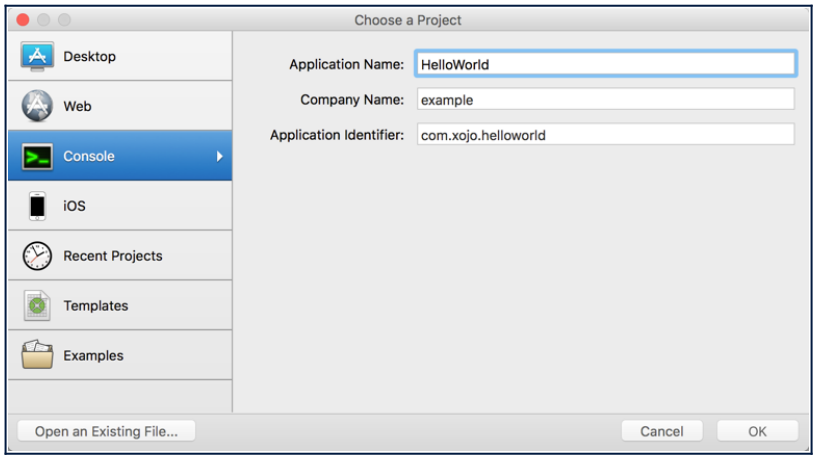
You you haven't already done so, you should first download and install Xojo on your primary computer:

http://www.xojo.com/download

Xojo IDE

Xojo is an integrated development environment, which means that you write your code and design your user interface using a single, integrated tool.

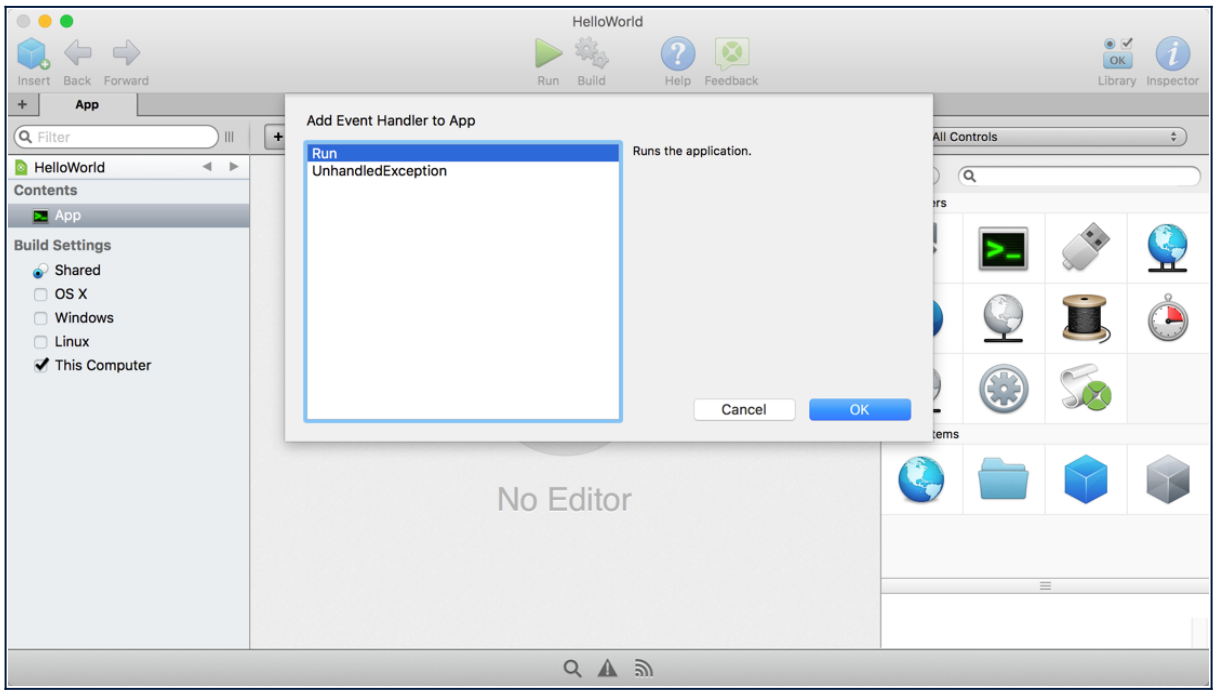
When you first start Xojo the Project Chooser is displayed. Here you can choose the type of project you want to create. Depending on your license, you can create Console, Desktop or Web projects for the Raspberry Pi. The simplest type of project to start with is the Console project, so click on "Console" in the Chooser and enter the name "HelloWorld" for the Application Name.



Click OK to open the main Xojo workspace window.

You are now going to make a simple app that will display "Hello, World!" when the app is run. Xojo programming is event based and the code you write runs when certain events occur. In the case of a console app, the Run event is called when the app starts. So the first thing you'll want to do is add the Run event to your app.

Click on App in the Navigator (the area on the left). In the Command Bar you'll see a button with a "+" appear. Click it to show the Add menu and select "Event Handler". In the list, choose the "Run" event and click OK.



Now you'll see the Code Editor where you can type your Xojo code. Since you don't really know any Xojo code yet, this will be kept as simple as possible. Enter this code:

```
Print("Hello, World!")
```

The Print command is used to display text to the terminal window. This command takes a parameter that is the text to display. The above code passes the text "Hello, World!". Text in Xojo always needs to be enclosed with quotation marks.

You should now save your project (File->Save) with the name "HelloWorld".

After saving the project, you'll want to build it for the Raspberry Pi. To do so you need to go to the Build Settings that are shown in the Navigator on the left. Select the item that says "Linux".

> Why select Linux? Because The Raspbian OS used by the Raspberry Pi is a Pi-specific version of Linux.

In the Inspector that appears (on the right-hand side), change the value in the Architecture property from "x86 32-bit" to "ARM 32-bit", which is the CPU used by the Raspberry Pi and Raspian.

Now you can click the "Build" button on the toolbar to build an app that you can transfer to the Pi. You may also want to save again.

As shown in the previous section, you can now use SFTP to transfer the app to your Pi. Start your SFTP client, connect to the Pi and select the HelloWorld folder containing the app you just built and copy it to the Pi at a location you can easily find. Usually you'll want to put it in the main user's Documents folder.

> Be sure you copy the entire folder containing both the app itself and the Libs folder!

After the app has transferred, you can now use ssh to connect to the Pi. In the terminal window, navigate to the location where you copied the HelloWorld app folder. For example, if you copied it to the user's documents folder, then you can use the CD command like this:

```
cd ~/Documents/HelloWorld
```

You can use the "ls" command to list the files to verify that they were copied over correctly. To run the app, use this command:

```
./HelloWorld
```

The "./" that precede the app name are important because it tells the terminal that the app is in the current folder.

You should see "Hello, World!" appear in the terminal.

Congratulations! You've made your first Raspberry Pi app using Xojo!

Remote Debugging

In the "Hello World" example, you manually transferred the built app to the Pi in order to run and test it. That process works fine and is necessary for when you want to install finished versions of your app on the Pi. But for testing, it can get a bit tedious to have to manually build and copy to the Pi each time you want to run and test.

Fortunately, Xojo has a built-in feature that can make this much easier: the Remote Debugger.

The Remote Debugger is a small program you run on the Pi that Xojo communicates with. When you want to Run your project on the Pi, you click Run Remotely from within Xojo to have your app automatically built for and transferred to the Pi in one quick and easy step.

The Remote Debugger is described in full in the User Guide, but this brief overview should allow you to get the Remote Debugger up and running on the Pi quickly.

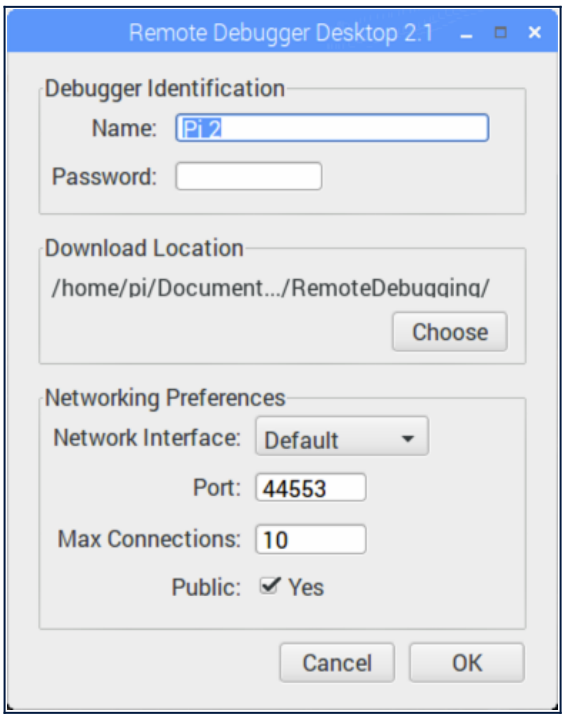
Currently you can Remote Debug to the Raspberry Pi from the Xojo IDE running on MacOS or Linux. Windows support will be added in a future version of Xojo.

Set up the Remote Debugger on the Pi

The Remote Debugger is included in your Xojo installation. Go the the installation folder and then find this file:

Extras/Remote Debugger Desktop/Linux ARM.zip

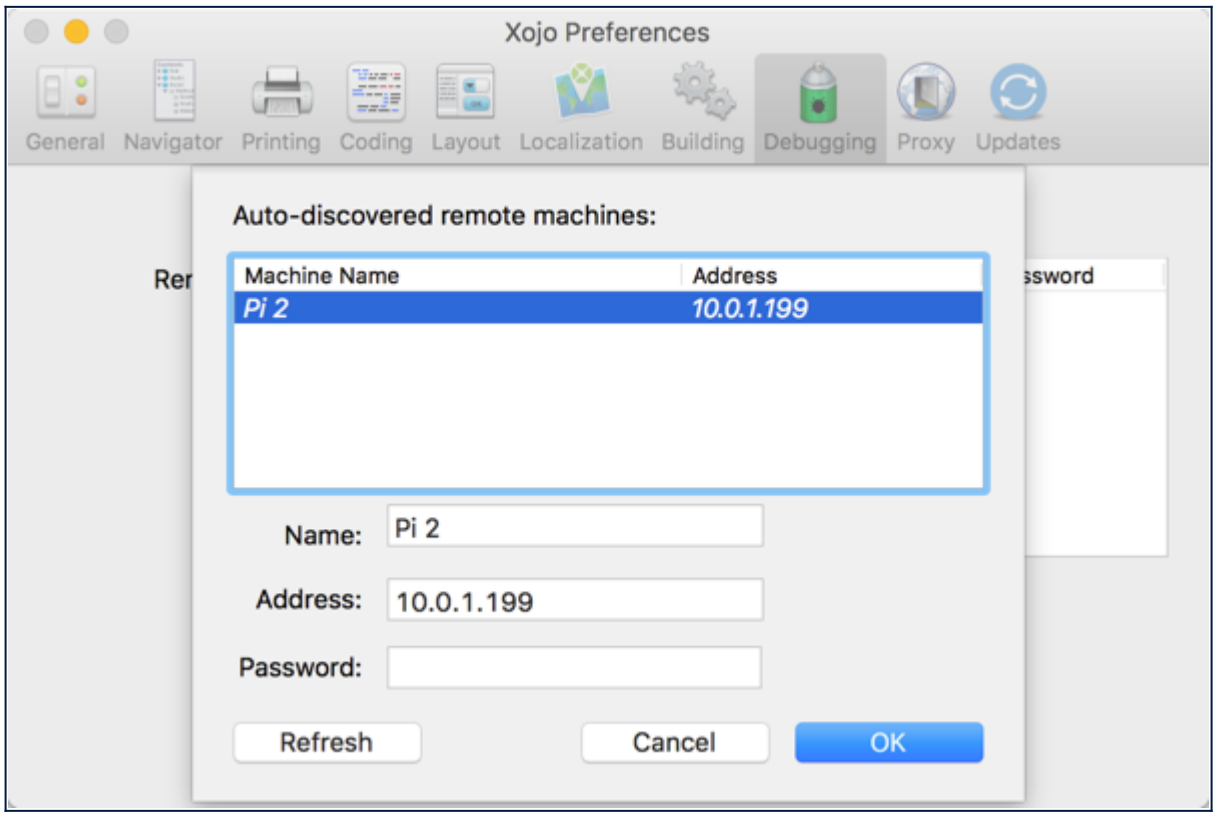
1. Copy this file to your Raspberry Pi and unzip it. Use SFTP to copy the file as described in Setting up Your Raspberry Pi.
2. On the Pi, find the ZIP file you copied over, right-click on it in the File Manager and select "Extract Here".
3. This creates a folder called "Remote Debugger Desktop". Open this folder and then double click the "Remote Debugger Desktop" file to start the Remote Debugger.
4. In the Remote Debugger, go to the Edit menu and select Options. Enter a name for your Pi in the "Name:" field.
5. In the Download Location section click Choose and choose a location for your debug apps to be copied. Using the Desktop works for most people.
6. Click OK to save these changes.



Tell Xojo your Pi is a Remote Debugging Host

Now you can start Xojo on your development machine and add the Pi as a Remote Debugging host.

1. Open the Xojo Preferences and click the Debugging panel.
2. In the Remote Debugging Hosts section, click the Add button to add your Pi. It's likely your Pi shows up as an auto-discovered remote machine. If it does, just select it and click OK.
3. If it does not show up you'll need to manually add it by entering a name and the IP address of the Pi (which is displayed in the Remote Debugger running on the Pi if you need to go back to find it).
4. Click OK to save the changes.



Run an App on the Pi

It's now time to try running a Xojo app on the Pi.

1. Open the "Hello World" project you created in Xojo Overview.
2. In the Build Settings section of the Navigator, click on Linux. Change the Architecture in the Inspector to "ARM 32-bit".
3. Now click the Project menu, go to "Run Remotely" and choose your Pi that is listed there. This builds your app for Pi and transfers it to the Pi.
4. A desktop app will automatically run, but you have to manually start a console app (such as this HelloWorld example). To manually start a console app, go to the Pi (using ssh, VNC or directly use it), open the Terminal and navigate to the location containing the transferred app. This will be the "Download Location" that you selected in the Remote Debugger Stub on the Pi. For example, if you chose the Desktop, then you can navigate to the folder by using: cd ~/Desktop/DebugHelloWOrld. Once you are in the directory, you can run the console app like this:

```
./DebugHelloWorld
```

The prefix "Debug" is added to the beginning of your app name when debugging.

While your app is running on the Pi, you can use Xojo to debug it. You app will stop at breakpoints allowing you to

view variables and other information just as you can when you are debugging normally.

Xojo Programming

This section covers some of the Xojo programming language data types that you'll want to know to get started programming your Pi.

To write code you use the Code Editor, which is essentially a text editor specifically designed for writing and editing Xojo code. Your code consists of a series of lines, with each line containing Xojo language commands. Like with any text editor, the text you write in the Code Editor does not "wrap" when you reach the end of the line. Instead the Code Editor scrolls. You create new lines by pressing Return.

The Xojo language consists of the series of commands that you write. With Xojo you generally have one command per line of code.

To make things easier, Xojo does not provide a big blank text editor for you to write your code. Code belongs to a project item method or a control on a layout. If you recall from the simple "HelloWorld" example from earlier, the code was placed in the Run event handler for the app. Event Handlers are a common place to put your code, but you can also create your own methods to contain code. Methods can be added to nearly any project item, including the main App object, windows, modules and classes.

You can add Event Handlers to a control in your layout by selecting the control and choosing "Event Handler" from the Add button menu on the Layout Editor commandbar, the Insert button on the main toolbar or the Insert menu.

You can add your own methods (which can be called by other code in your app) by choosing "Method" from the Add or Insert options.

Variables

A variable is the simplest way to store values that can be changed (or vary) by your code. All variables are defined in your code using the Dim statement and each variable must be defined before it is used. When you define a variable you tell it the type of data that it can contain.

A variable definition consists of four parts: the Dim keyword, the variable name, the **As** keyword and the data type. This is also often referred to as a "variable declaration". Here is an example variable declaration:

```
Dim age As Integer
```

There are several simple, built-in data types: Text (String is also used), Integer, Double, Currency, Boolean, and Color. Data Types are covered in the Data Types section below. Or you can refer to Using Data Types in the User Guide.

Once you have declared a variable, you can assign it a value:

```
Dim age As Integer
age = 42
```

Variables can be declared anywhere within a method (or event), as long as the declaration precedes its first usage.

A variable can be accessed only within the method (or event) in which it was declared. When the method is finished, the memory that was used to store the variable's value becomes available for other uses. This means that another method in the application cannot access the variable value.

The term **scope** is used to describe where something, such as a variable, can be accessed. Variables and constants declared within methods have what is called a local scope because they are locally available only to the method. Thus these are also referred to as local variables.

Data Types

Xojo is a strongly-typed programming language. This means that you must specify the type of every variable you create. Strongly-typed programming languages are easier and safer to use because the compiler can inform you of programming mistakes in variable usage before they make it into your app, reducing the chance of bugs.

Boolean

Boolean is one of the simplest data types. It can only contain one of two values: True or False (the default). This is how you declare a Boolean variable:

```
Dim b As Boolean
```

The special keywords True and False are used to change the value of a Boolean variable. If you want to declare a Boolean variable and assign it to True, you can do so in one step:

```
Dim b As Boolean = True
```

Boolean variables can also be assigned the result of a Boolean expression. A Boolean expression is an expression that evaluates to True or False. For example, this code sets a variable to True or False depending on whether an Integer value is greater than 5:

```
Dim i As Integer = 10
Dim b As Boolean
b = (i > 5) ' b gets set to True
```

Text

Text stores its text as Unicode.

The maximum size of Text is limited only by available memory. To declare a variable containing Text:

```
Dim t As Text
```

The default value of text is simply an empty text, often written as "". You can also check if text is empty using the Empty method:

```
Dim t As Text
If t.Empty Then
  ' Do something if text is empty
End If
```

You can directly assign text to the variable as part of its declaration:

```
Dim t As Text = "Hello, World!"
```

Basically, any type of characters can be stored as Text, such as: "Lucas", "12/30/2015", "42.15".

Yes, even though those last two values look like a date and a number, they are actually Text because they are a series of characters within quotes. If you want to treat them as if they are an actual date or a number then you need to use a specific data type for that, such as Date or Double.

Combining Text

You can combine multiple text values together by using the addition (+) operator. This concatenates the two values together. For example:

```
Dim name As Text = "Lucas" ' no space at end
Dim age As Text = "12"
Dim combined As Text = name + age ' result is "Lucas11"
```

Remember, that text values combined in this manner are only ever concatenated, regardless of what value they contain. For example, even if it looks like the text contains numbers, adding them will not calculate their sum:

```
Dim value1 As Text = "42"
Dim value2 As Text = "10"
Dim value3 As Text = value1 + value2 ' result is the text "4210", not 52
```

Text Management

Text has many methods that can be used to manage its contents. For example, you may want to check if a Text variable contains another text value or you may want to remove whitespace from a text value.

To learn more about the methods that are available with Text variables, refer to the Text data type in the Reference Guide.

String

In most cases you will want to use the Text data type, but String is sometimes useful when working with UI controls that do not use the Text data type.

String stores the text in memory as bytes with an optional encoding (usually UTF-8). A String without an encoding may or may not contain text (it could actually be binary data), which can lead to all kinds of hard-to-find bugs when trying to display the String as text.

If you have a String variable or property, you can convert it to Text by calling the ToText method:

```
Dim s As String = "Hello"
Dim t As Text = s.ToText
Dim caption As Text = OKButton.Caption.ToText
```

Converting a String to a Text works only when the String has a known encoding. If the encoding of the String is not known (because it was loaded from an outside source such as a file, a database or memory), then you will have to first set the encoding using the DefineEncoding method.

Conversely, you can always convert a Text variable to a String because the text is Unicode and can always be correctly converted.

Numbers

Xojo has separate data types depending on the type of number you need. The Integer type is used for whole numbers (positive or negative and 0). If you need a number that contains a decimal, you can use either Double or Currency.

Because these data types contain actual numbers, you can perform mathematical computations on them.

Integer

An Integer contains whole numbers and is declared like so (default value is 0):

```
Dim value As Integer
```

You can assign it a value when you declare it:

```
Dim value As Integer = 42
```

If you try to assign a value with a decimal to an Integer, the decimal value is truncated (not rounded) before it is assigned. For example:

```
Dim value As Integer
value = 42.65 ' value will contain 42
```

Double

A Double is a number that can contain a floating-point decimal value. In other languages, Double may be referred to as a double precision real number. Because Doubles are numbers, you can perform mathematical calculations on them.

⚠ You should avoid using Double to store monetary values. As an IEEE floating-point number, there are some decimal values that cannot be represented as a Double and this could cause rounding issues. In these situations, use Currency instead.

The default value of a Double is 0.0:

```
Dim value As Double
```

You can also assign a value when you declare it:

```
Dim value As Double = 3.14
```

Converting Between Numbers

You can convert between numbers without any special commands, but you have to keep in mind that the numbers may be rounded or truncated. For example, converting a Double to an Integer does not round the value:

```
Dim d As Double = 4.6
Dim i As Integer = d ' i = 4
```

Use the Round method to round a value.

You can also directly assign an Integer to a Double:

```
Dim i As Integer = 42
Dim d As Double = i
```

Program Structure

This chapter covers some of the Xojo programming language commands that you'll want to know to get started programming your Pi.

The methods you write execute one line at a time from top to bottom, left to right. There will be times when you want your application to execute some of its code based on certain conditions (using comparisons or boolean expressions). When your application's logic needs to make decisions it's called branching. This allows you to control what code gets executed and when. You may also want your code to execute some of its code repeatedly. This is called looping.

If...Then...End If

The If...Then...End If statement is used when your code needs to test a boolean (True or False) expression and then execute code based on its result. If the expression you are testing is True, then the lines of code you place between the If...Then line and the End If line are executed, otherwise they are skipped.

```
If condition Then
  ' [Your code goes here]
End If
```

Say you want to test the Integer variable month and if its value is 1, run your code:

```
If month = 1 Then
  ' [Your code goes here]
End If
```

The part "month = 1" is a boolean expression; it's either True or False. The variable month is either 1 or it is not 1.

Suppose you have a Button that performs an additional task if a particular CheckBox is checked. The value property of a CheckBox is Boolean so you can test it in an If statement easily:

```
If CheckBox1.Value Then
  ' [Your code goes here]
End If
```

Remember that you can declare local variables using the Dim statement inside an If statement. However, such variables go out of scope after the End If statement. For example:

```
If error = -123 Then
  Dim a As Text
  a = "Oops! An error occurred."
End If
```

```
Label1.Text = a ' out of scope!
```

If you need the variable after the End If statement, you should declare it local to the entire method, not within the If...End If statement.

While...Wend

A While loop executes one or more lines of code between the While and the Wend statements. The code between these statements is executed repeatedly, provided that the condition passed to the While statement continues to evaluate to True.

Consider the following example:

```
Dim i As Integer
While i < 10
  i = i + 1
Wend
```

The variable “i” will be zero by default when it is created by the Dim statement. Because zero is less than ten, execution will move inside the While...Wend loop. The variable i is incremented by one and the loop returns to the top where the While statement checks to see if the condition is still True and if it is, then the code inside the loop executes again. This continues until the condition is no longer True. If the variable i was not less than ten in the first place, the contents fo the loop are skipped entirely and execution would continue at the line of code after the Wend statement.

Do...Loop

Do loops are similar to While loops but a bit more flexible. Do loops continue to execute all lines of code between the Do and Loop statements until a particular condition is True. While loops on the other hand execute as long as the condition remains True. Do loops provide more flexibility than While loops because they allow you to test the condition at the beginning or end of the loop. The example below shows two loops; one testing the condition at the beginning and the other testing it at the end:

```
Dim i As Integer
Do Until i = 10
  i = i + 1
Loop

i = 0
Do
  i = i + 1
Loop Until i = 10
```

The difference between these two loops is this: In the first case, the loop will not execute if the variable i is already

equal to ten. The second loop executes at least once regardless of the value of i because the condition is not tested until the end of the loop.
It is possible to create a Do loop that does not test for any condition. Consider this loop:

```
Do
  i = i + 1
Loop
```

Because there is no test, this loop will run endlessly. You use the Exit command to force a loop to exit without testing for a condition. However, this is generally considered poor design because you have to read through the code to figure out what will cause the loop to end.

```
Do
  i = i + 1
  If i > 10 Then Exit
Loop
```

Now that you understand the various coding concepts, now it is time to learn more about the place where you write you code: methods.

Methods are the building blocks of your application. The code you write most often exists in a method. A method is one or more instructions that are performed to accomplish a specific task; an action of some sort.

There are many built-in methods. For example, the Quit method causes your application to quit. Most classes have built-in methods. For example, the ListBox class has a method called AddRow for adding rows to it (as the name implies). And you can of course, create your own methods.

For...Next

While and Do loops are perfect when the number of times the loop should execute cannot be determined because it is based on a condition. A For loop is for cases in which you can determine the number of times to execute the loop. For example, suppose you want to add the numbers one through ten to a List Box. Since you know exactly how many times the code should execute, a For loop is the right choice. For loops also differ from While and Do loops because For loops have a loop counter variable, a starting value for that variable and an ending value. The basic construction of a For loop is:

```
Dim Counter As Integer
For Counter = 0 To 100
  ' [your code goes here]
Next
```

Notice that the Dim statement declares the counter as an Integer. Although an Integer is the most common way to define the counter variable, you can also declare it as a Single or Double.

In this example, the counter variable was declared in the usual way, via the Dim statement. Since counter variables are rarely needed outside the For loop, you can also declare the counter variable right inside the For statement. In other words, you can redo this example like this:

```
For Counter As Integer = 0 To 100
  ' [your code goes here]
Next
```

Notice that the Dim statement has been removed from the example. If you declare the counter variable this way, you can use it only within the For loop. It goes out of scope after the For loop is finished. This is the recommended way to declare a counter variable. Of course, if you need to read or change the value of the counter variable outside the For loop, which is rarely necessary, you should use the Dim statement instead.

In the prior examples, the starting value and the ending value are specified as numbers. You can also use variables, as shown in this example:

```
Dim startingValue, endingValue As Integer
startingValue = 0
endingValue = 100

For counter As Integer = startingValue To endingValue
  ' [your code goes here]
Next
```

The first time through the loop, the counter variable will be set to StartingValue. When the loop reaches the Next statement, the counter variable will be incremented by one. When the Next statement is reached and the counter variable is equal to EndingValue, the counter will be incremented and the loop will end.

Look back at the example mentioned earlier. You want to add the numbers one through ten to a List Box. The following code accomplishes that:

```
For i As Integer = 1 To 10
  ListBox1.AddRow(Str(i))
Next
```

The counter variable (i in this case) is passed to the Str function to be converted to a string so that it can be passed to the AddRow method of ListBox1.

Note: The letter “i” is commonly used as the loop counter for historical reasons. In the old FORTRAN programming language, the letters I through N are integers by default. Therefore, FORTRAN programmers began the practice of using those letters as counters, and in the order they appear in the alphabet. That is, if a FORTRAN programmer needed to nest one loop in another, he would use j as the counter for the inner loop. This convention made it easy for FORTRAN programmers to follow the logic of code that processed multi-dimensional arrays.

By default, For loops increment the counter by one. You can specify another increment value using the Step statement. In this example, the Step statement is added to increment the counter variable by 5 instead of 1:

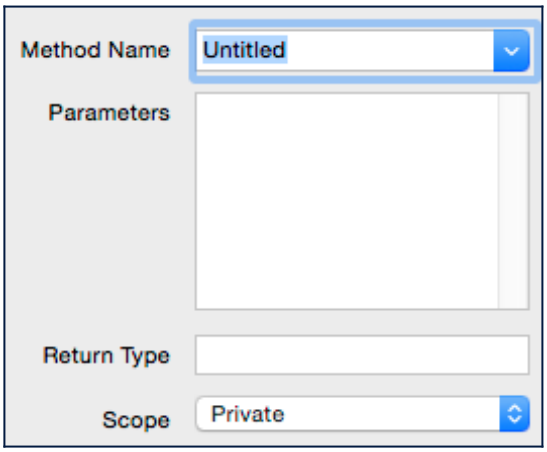
```
For i As Integer = 5 To 100 Step 5
  ListBox1.AddRow(Str(i))
Next
```

In this example, the For loop starts the counter at 100 and decrements by 5:

```
For i As Integer = 100 DownTo 1 Step 5
  ListBox1.AddRow(Str(i))
Next
```

Creating Your Own Methods

Methods can be added to project items, including classes (including Windows and Web Pages) and modules. To add a method, choose Insert ↠ Method from the menu or toolbar when the appropriate project item is selected. This adds the method, displays a blank code editor and displays the Inspector where you can set the properties for the method:



- **Method Name**
  The name of the method. Just like variables, methods are given names to describe them and the same naming rules apply.
- **Parameters**
  Parameters are values that you pass to the method that it can then use within the method as if they were variables. Parameters are separated by commas and are declared similarly to how you use the Dim statement:
  `value As Integer`
  `value As Integer, name As Text`
- **Return Type**
  Methods that do not specify a return type are called Subroutines (or Procedures in some other languages). Methods that specify a return type are called Functions. The Return Type can be any valid data type, including classes.
- **Scope**
  Scope indicates what parts of your code can call the method. Choices are Public, Protected and Private.
  - Public methods can be called from anywhere in your code with no restrictions.

- Protected methods have some restrictions, which vary depending on where the method is located (class or module).
- Private methods can only be called by the module or class that contains the method.

The method signature (its name, parameters and return type) also appears at the top of the Code Editor for reference.

Using Methods

To use a method you specify its name, optionally prefixed by the module or class instance name (depending on how the method is defined). Classes are discussed in the Object-Oriented Programming section coming up next and Modules are discussed in the Modules section of the User Guide.

An example of a method is the ToText method on the number data types that converts the number to text:

```
Dim i As Integer = 42
Dim t As Text = i.ToText
```

Passing Parameters to Methods

Some methods, such as the ToText method shown above, are called simply with their name. But some methods require additional information or values. This information that is passed to a method is called a parameter. A method can have any number of parameters.

A Dictionary is a built-in class that is commonly used to store information. It has several methods to add, remove or lookup values in it. This is how you can create a Dictionary:

```
Dim myDictionary As Dictionary
myDictionary.Value("Name") = "Red Sox"
```

This method call removes the key "Name" and its value from the Dictionary:

```
myDictionary.Remove("Name")
```

The value "Name" was passed as a parameter to the Remove method.

Methods define their parameters and specify the types for them. A method can have any number of parameters. When passing multiple parameters, separate each one with a comma.

The Lookup method on a Dictionary takes two parameters and can be used to get a value for a key and if not available, provide a default:

```
```

```
Dim value As Text = myDictionary.Lookup("Name", "Unknown")
```

In these examples, the parameters have been passed as text literals, but you can also pass variables and constants as parameters instead:

```
Dim key As Text = "Name"
Dim default As Text = "Unknown"
Dim value As Text = myDictionary.Lookup(key, default)
```

As with variable assignment, the parameters you pass to the method must match the types of the parameters as declared on the method.

Returning Values from Methods

Some methods return values. These methods are called Functions. When a method returns a value, the value is passed back from the method to the line of code that called the method. For example, the method Ticks returns the number of ticks (a tick is 1/60th of a second) that have passed since you turned on your computer. You can assign the value returned by a method the same way you assign a value to a variable. In the example below, the value returned by Ticks is assigned to the variable *elapsed*:

```
Dim elapsed As Double
elapsed = Ticks
```

Some methods require parameters and return a value. You saw this earlier with the Lookup method for a Dictionary:

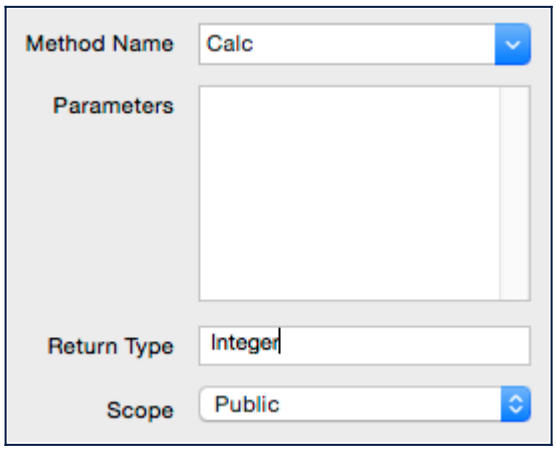
```
Dim value As Text = myDictionary.Lookup("Name", "Unknown")
```

The method takes two parameters and returns a value.

You can also directly pass methods that return values to other methods:

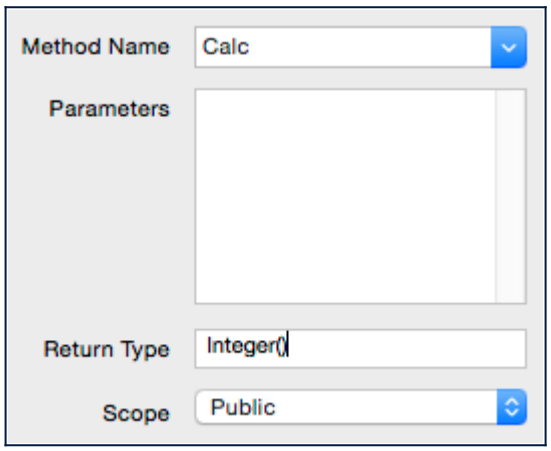
```
Dim t1 As Text = "Hello, how are you?"
Dim t2 As Text = "Hello, I am fine."

Dim b As Boolean
b = t1.BeginsWith(t2.Left(5)) ' True
```

For a method to return a value, you just have to specify a Return Type in the method properties of the Inspector. For example, this method definition returns an Integer:



A method can return any type, from the intrinsic data types to any class or other type you have created yourself.

Methods can also return arrays of any type. To indicate that the return type is an array, add the double parenthesis to it. This returns an Integer array:



Debugging

When you run your apps on the Pi using the Remote Debugger, you can use the built-in Xojo debugger to watch your code as it runs. To do this, you set breakpoints in your code. A breakpoint is an indicator that tells the debugger to activate itself when the line of code is reached. For example, you might want to set a breakpoint at the start of a method if you want to carefully review the code as it executes.

To set breakpoints, you click on the “dashes” that appear in the gutter of the Code Editor. Each dash indicates a line of code that can have a breakpoint set. You can also set a breakpoint using the Project ↠ Breakpoint ↠ Turn On menu (⌘+ on Mac, Ctrl- on Windows and Linux). The same command turns off a previously set breakpoint on the line.

If you want to turn off all breakpoints throughout your project, use the Project ↠ Breakpoint ↠ Clear All menu. To see all breakpoints in your project, use Project ↠ Breakpoint ↠ Show All menu to display the breakpoints in the Find panel.

When you run your project, code execution stops when a breakpoint is reached and the Xojo Debugger displays. In the debugger you'll see the current line of code highlighted and a list of variables in the current code block (method,

event, etc.) for you to watch.

Use the button on the command bar to control the debugger:

- **Pause/Resume**: Use Pause to pause a running app and activate the debugger. If you are in the debugger, the Resume button tells your app to continue running where it left off. You can also click the Run button on the main toolbar to Resume, select Project ↠ Resume from the menu or you can use the ⌘+R (Ctrl-R on Windows and Linux) shortcut.
- **Stop**: The Stop button immediately stops the running app. The app is quit immediately and no further code is run. You can also use the shortcut Shift+⌘+R (Shift-Ctrl-R on Windows and Linux).
- **Step**: The Step button is used to run the code one line at a time. Each time you click Step, the highlighted code is executed and you remain in the debugger. If you click Step while on a method call, the method is called and you move to the next line of code after the method call. Step is the command you use most often while debugging. In addition to clicking the button, you can use Project ↠ Step ↠ Step Over menu command or the shortcut Shift+⌘+O (Shift-Ctrl-O on Windows and Linux).
- **Step In**: The Step In button works like the Step button except when you reach a method call. Instead of calling the method and moving to the next line of code, Step In moves you to the first line of code in the method. In addition to clicking the button, you can use Project ↠ Step ↠ Step Into menu command or the shortcut Shift+⌘+I (Shift-Ctrl-I on Windows and Linux).
- **Step Out**: If you are in a method, clicking Step Out, runs the rest of the code in the method and then stops when the method returns. In addition to clicking the button, you can use Project ↠ Step ↠ Step Into menu command or the shortcut Shift+⌘+T (Shift-Ctrl-T on Windows and Linux).
- **Edit Code**: The Edit Code button allows you to jump to the Code Editor for the current method that is in the debugger. Here you can edit the code (which you can not do in the code display of the debugger). However, changes that you make to code in the Code Editor are not reflected in the currently running app. You'll need to quit and re-run the app to see the changes you made.

You can learn more about the Xojo Debugger in the Using the Debugger topic of the User Guide.

Object-Oriented Programming with Classes

In its simplest form, a class is a container of code and data much like a module. But unlike a module, a class provides better code reuse. Classes are the fundamental building blocks of object-oriented programming.

Understanding Classes, Instances and References

Before you can use a class in your project, it is important to understand the distinction between these three concepts: the **class** itself, the **instance** of the class and the **reference** to the class.

The Class	The Instance
Think of the class as a template for a container of information (code and data), much like a module. And like a module, each class exists in your project only once. But unlike a module, a class can have multiple instances that each contain different data.	Classes provide better code reuse because of a concept called instances. Unlike a module, which exists only once in your application, a class can have multiple instances. Each instance (also called an object) is a separate and independent copy of the class and all its methods and properties.

The Class

There are many built-in classes for user interface controls such as PushButton, WebButton, Label, WebLabel, TextField, WebTextField, ListBox, and WebListBox.

By themselves, control classes are not all that useful; they are just abstract templates. But each time you add a control class to a layout (a window or web page) you get an instance of the class. Because each button is a separate instance, this is what allows you to have multiple buttons on a window, with each button being completely independent of each other with its own settings and property values.

You create classes in Xojo by choosing Class from the Insert menu or toolbar button. This is an example of a Vehicle class that will be used in later examples:

```
Class Vehicle
  Property Brand As Text
  Property Model As Text
End Class
```

The Instance

These instances are what you interact with when writing code on the window and is what the user interacts with when they use your application.

For example, when you drag a TextArea from the Library to a Window, you create a usable instance of the TextArea on the Window. The new instance has all the properties and methods that were built into the TextArea class. You get all that for free — styled text, multiple lines, scroll bars, and all the rest of it. You customize the particular instance of the TextArea by modifying the values of the instance’s properties.

When you add a control to a window (or web page), the Layout Editor creates the reference for you automatically (it is the name of the control).

However, when writing code you create instances of classes using the New keyword. For example, this create a new

instance of a Vehicle class that is in your project:

```
Dim car As New Vehicle
```

The Reference

A reference is a variable or property that refers to an instance of a class. In the above code example, the car variable is a reference to an instance of the Vehicle class.

You interact with properties and methods of the class using dot notation like this:

```
Dim car As New Vehicle
car.Brand = "Ford"
car.Model = "Focus"
```

Here the Brand and Model were defined as properties of the Vehicle class and they are given values for the specific instance. Having an instance is important! If you try to access a property or method of a class without first having an instance, you will get a NilObjectException likely causing your app to quit. This is is one of the most common programming errors that people make and is typically caused by forgetting the New keyword. For example, this code might look OK at first glance:

```
Dim car As Vehicle
car.Brand = "Ford"
car.Model = "Focus"
```

But the 2nd line will result in a NilObjectException when you run it because car is not actually an instance. It is just a variable that is declared to eventually contain a Vehicle instance. But since an instance was not created, it gets the default value of Nil, which means it is empty or undefined.

If you are not sure if a variable contains an instance, you can check it before you use it:

```
If car <> Nil Then
  car.Brand = "Ford"
  car.Model = "Focus"
End If
```

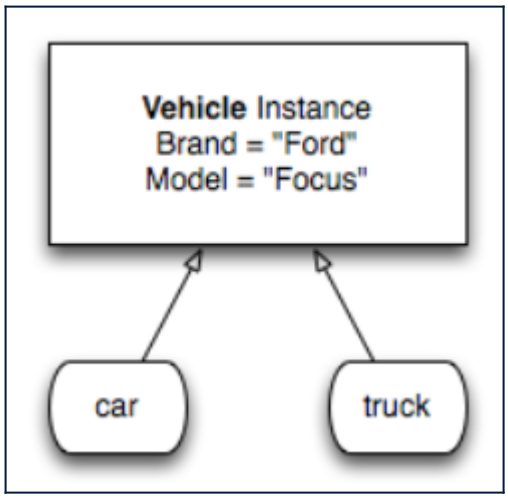
When the variable is not Nil, then it has a valid reference. And since it is a reference, it has important considerations when assigning the variable to another variable: When you do an assignment from one reference variable to another, the second variable points to the same reference as the first variable.

This means the second variable refers to the same instance of the class as the first variable and is not a copy of it. So if you change a property of the class with either variable, then the property is changed for both. An example might help:

```
Dim car As New Vehicle
```

```
car.Brand = "Ford"
car.Model = "Focus"
Dim truck As Vehicle
truck = car
' truck.Model is now "Focus"
car.Model = "Mustang"
' truck.Model is now also "Mustang"
truck.Model = "F-150"
' car.Model is now also "F-150"
```

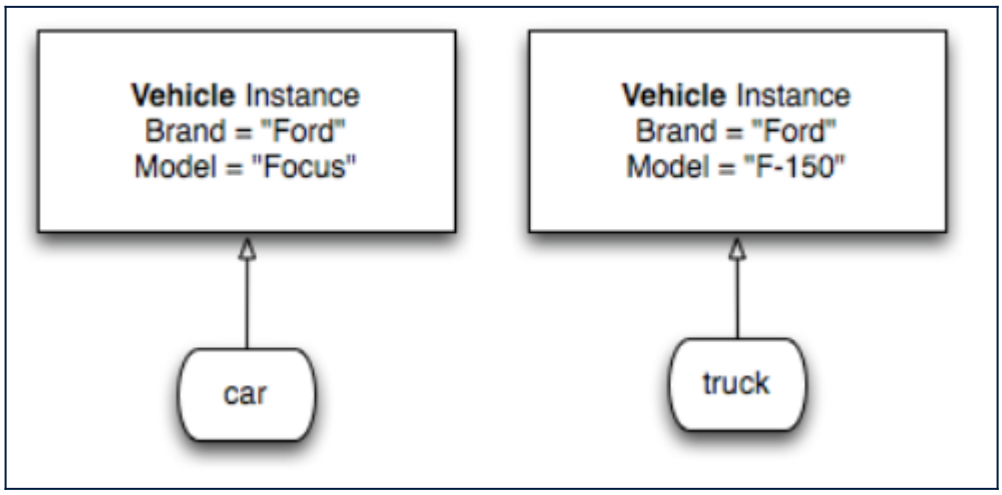
In this diagram you can see that the variables for both car and truck point to the same instance of Vehicle. So changing either one effectively changes both.



If you want to create a copy of a class, you need to instead create a new instance (using the New keyword) and then copy over its individual properties as shown below:

```
Dim car As New Vehicle
car.Brand = "Ford"
car.Model = "Focus"
Dim truck As New Vehicle
truck.Brand = car.Brand
trunk.Model = car.Model
' truck.Model is now also "Focus"
truck.Model = "F-150"
' car.Model remains "Focus"
```

When you do it this way, you get two separate instances. Changes to one do not affect the other as you can see in these diagrams:



Adding Classes to Your Projects

Adding a class to a project is easy. To add a new class, click the Insert button on the toolbar and choose Class or select Class from the Insert menu. This adds a new class to the Navigator with the default name (Class1 for the first class).

Use the Inspector to change the name of the class. Like modules, classes primarily contain properties and methods. But they can also contain many other thing such as constants, enumerations, events and structures.

Adding Properties to Classes

Properties are variables that belong to an entire class instance rather than just a single method. To add a property to a class, use the Add button on the Code Editor toolbar, Insert ↠ Property from the menu, the contextual menu or the keyboard shortcut (Option-Command-P on OS X or Ctrl+Shift+P on Windows and Linux). You can set the property name, type, default value and scope using the Inspector.

To quickly create a property, you can enter both its name and type on one line in the Name field like this: PropertyName As DataType. When you leave the field, the type will be set in the Type field.

Properties added in this manner are sometimes called Instance Properties because they can only be used with an instance of the class. You can also add properties that can be accessed through the class itself without using an instance. These are called Shared Properties.

Shared Properties

A shared property (sometimes called a Class Property) is like a “regular” property, except it belongs to the class, not an instance of the class. A shared property is global and can be accessed from anywhere its scope allows. In many ways, it works like a module property.

It is important to understand that if you change the value of a shared property, the change is available to every usage of the shared property.

Generally speaking, shared properties are an advanced feature that you only need in special cases. For example, if

you are using an instance of a class to keep track of items (e.g., persons, merchandise, sales transactions, and so forth) you can use a shared property as a counter. Each time you create or destroy an instance of the class, you can increment the value of the shared property in its constructor and decrement it in its destructor. (For information about constructors and destructors, see the section Constructors and Destructors.) When you access it, it gives you the current number of instances of the class.

Adding Methods to Classes

To add a method to a class, use the Add button on the Code Editor toolbar, Insert ↠ Method from the menu, the contextual
menu or the keyboard shortcut (Option-Command-M on OS X or Ctrl+Shift+M on Windows and Linux). You can set the method name, parameters, return type and scope using the Inspector.

When typing the method name, the field will autocomplete with the names of any methods on its super classes.

Methods added in this manner are called Instance Methods because they can only be used with an instance of the class.
You can also add methods that can be accessed through the class itself. These are called Shared Methods.

Project - Find Seltzer

One of the best ways to learn how to make apps is by making apps. Now that you have an understanding of Xojo and its programming language, you are going to make a simple game called "Find Seltzer". Find Seltzer is a simple console text adventure game where the goal is to move through the rooms of a house and find where Seltzer the cat is hiding. You play the game by typing commands to move from room to room where you examine the items in the room.

To get started, create a new Console project and name it "FindSeltzer".

This project uses two classes to track the room and the stuff contained in the rooms. The classes are **Room** and **Stuff**.

Stuff Class

First you'll want to create the Stuff class since it is used by the Room class. In Xojo, select Insert ↠ Class and name the class "Stuff". This class has two properties, which you can add by choosing Add ↠ Property from the command bar for the class:

- SeltzerIsHere As Boolean
- StuffName As String

Make sure the scope of these properties is set to Public. The SeltzerIsHere property is set to True for the stuff that seltzer is hiding under. It is randonly set to something when a new game starts. StuffName is simply the name of the stuff, such as "Table" or "Bed". That's it for the Stuff class.

Room Class

Now you can add the Room class. This class is a bit more complicated because it has properties to track information about the room and methods for moving and looking at what is in the room. Start by adding these properties to Room (make sure their scope is set to Public):

- MoveableDirections As Dictionary
- RoomName As String
- RoomStuff() As Stuff

MoveableDirections is a Dictionary that contains the directions that you can move to. For example, if you are in the Kitchen you may want to allow the user to move west to get to the playroom. This dictionary is populated as part of the game setup.

RoomName is simply the name of the room, such as "Kitchen" or "Playroom".

RoomStuff is an array of Stuff classes. Because it is an array it allows a room to have multiple things in it.

Now on to the three methods used by Room. The first method is a Constructor. Add it by selecting Add->Method from the command bar for the class. In the Inspector, you can choose "Constructor" from the popupmenu or just type it. The Constructor is called each time a new Room is created and its purpose is to simply initialize the Dictionary so it's code is this:

```
MovableDirections = New Dictionary
```

The next method is LookAt. This method takes the name of something to look at (a Stuff). If the item is in the room, it checks if Selzter is hiding there. If Seltzer is there text is displayed and the method returns True, which means you won. If Seltzer is not there, text is displayed and it returns False. Here is the code:

```
Function LookAt(noun As String) As Boolean
  For Each s As Stuff In RoomStuff
    If s.StuffName = noun Then
      app.score = app.score + 1
      If s.SeltzerIsHere Then
        Print "You found Seltzer!"
        Return True
      Else
        Print "There is nothing to see."
        Return False
      End If
    End If
  Next

  App.Sorry(noun)

  Return False
End Function
```

⚠ Remember, when you add methods in Xojo, you do not type the method header. That's the part that starts with "Function". Instead you add a method to the class and specify its name and parameters in the Inspector.

The final method is the Move method, which looks up if you can move the the specified room. If so, then the new room is returned.

```
Function Move(direction As String) As Room
  If MovableDirections.HasKey(direction) Then
    Return MovableDirections.Value(direction)
  Else
    Return Nil
  End If
End Function
```

That's it for the Room class.

Game Code

The rest of the code is what runs the game and it is on the App object. First, there are three properties for tracking the state of the game:

- CurrentRoom As Room
- Score As Integer

CurrentRoom is the current room the user is in. Score is the count of moves it takes to find Seltzer.

There are four methods in the class that are used to play the game. The simplest method is called Sorry and it is called when a command is not recognized:

```
Sub Sorry(command As String)
  Print "Sorry, I don't understand '" + command + "'."
End Sub
```

The ShowCommands method is called when the user types "?" or "help" and it displays the commands that the game recognizes:

```
Sub ShowCommands()
  Print "Available commands:"
  Print "  Go N/E/W/S/U/P : Move north, east, west, south, up down"
  Print "  Look noun: Look an item in a room"
  Print "  Examine noun: Look an item in a room"
  Print "  Quit: Quit the game"
  Print "  ? or Help: Show this text"
  Print ""
End Sub
```

The DisplayRoom method displays the details for the current room and is called when the user types "look":

```
Sub DisplayRoom()
  Print "You are at the " + CurrentRoom.RoomName + "."
  Print ""

  If CurrentRoom.RoomStuff.Ubound >= 0 Then
    For Each s As Stuff In CurrentRoom.RoomStuff
      Print "There is a " + s.StuffName + "."
    Next
    Print ""
  End If

  Dim move As String = "You can move "
  For Each d As String In CurrentRoom.MovableDirections.Keys
    move = move + d + " "
  Next
  Print move.Trim + "."
End Sub
```

The largest method in the game is the Setup method. It creates all the stuff, the rooms and the map for the house. It then randomly assigns Seltzer to be in one of the items in a room:

```
Sub Setup()
  Dim rooms() As Room

  ' Front Porch
  Dim door As New Stuff
  door.StuffName = "Door"

  Dim frontPorch As New Room
  frontPorch.RoomName = "Front Porch"
  frontPorch.RoomStuff.Append(door)
  rooms.Append(frontPorch)

  ' Start at front door
  CurrentRoom = frontPorch

  ' Downstairs hallway
  Dim hallway As New Room
  hallway.RoomName = "Downstairs Hallway"
  rooms.Append(hallway)

  ' Kitchen
  Dim table As New Stuff
  table.StuffName = "Kitchen Table"

  Dim kitchen As New Room
  kitchen.RoomName = "Kitchen"
  kitchen.RoomStuff.Append(table)
  rooms.Append(kitchen)

  ' Playroom
  Dim condo As New Stuff
  condo.StuffName = "Cat Condo"

  Dim playRoom As New Room
  playroom.RoomName = "Playroom"
  playroom.RoomStuff.Append(condo)
  rooms.Append(playRoom)

  ' Living room
  Dim couch As New Stuff
  couch.StuffName = "Couch"

  Dim livingRoom As New Room
  livingRoom.RoomName = "Living Room"
  livingRoom.RoomStuff.Append(couch)
  rooms.Append(livingRoom)

  ' Upstairs hallway
```

```
Dim upstairHallway As New Room
upstairHallway.RoomName = "Upstairs Hallway"
rooms.Append(upstairHallway)

' Mom&Dad's Bedroom
Dim gate As New Stuff
gate.StuffName = "Gate"

Dim MFBedroom As New Room
MFBedroom.RoomName = "Mother & Father Bedroom"
MFBedroom.RoomStuff.Append(gate)
rooms.Append(MFBedroom)

' Office
Dim desk As New Stuff
desk.StuffName = "Desk"

Dim office As New Room
office.RoomName = "Office"
office.RoomStuff.Append(desk)
rooms.Append(office)

' Boy's Bedroom
Dim bed As New Stuff
bed.StuffName = "Bed"

Dim boyBedroom As New Room
boyBedroom.RoomName = "Boy's Bedroom"
boyBedroom.RoomStuff.Append(bed)
rooms.Append(boyBedroom)

' Girl's Bedroom
Dim sax As New Stuff
sax.StuffName = "Saxophone"

Dim girlBedroom As New Room
girlBedroom.RoomName = "Girl's Bedroom"
girlBedroom.RoomStuff.Append(sax)
rooms.Append(girlBedroom)

' House Map
frontPorch.MovableDirections.Value("N") = hallway
hallway.MovableDirections.Value("N") = kitchen
hallway.MovableDirections.Value("S") = frontPorch
kitchen.MovableDirections.Value("S") = hallway
PlayRoom.MovableDirections.Value ("E") = kitchen
PlayRoom.MovableDirections.Value ("S") = livingRoom
LivingRoom.MovableDirections.Value ("E") = hallway
```

```
  LivingRoom.MovableDirections.Value ("N") = playRoom
  kitchen.MovableDirections.Value("W") = playRoom
  hallway.MovableDirections.Value("W") = livingRoom
  hallway.MovableDirections.Value("U") = upstairHallway
  UpstairHallway.MovableDirections.Value("D") = hallway
  UpstairHallway.MovableDirections.Value("E") = MFBedroom
  MFBedroom.MovableDirections.Value("W") = upstairHallway
  UpstairHallway.MovableDirections.Value("N") = office
  Office.MovableDirections.Value("S") = upstairHallway
  UpstairHallway.MovableDirections.Value("W") = boyBedroom
  boyBedroom.MovableDirections.Value("E") = upstairHallway
  girlBedroom.MovableDirections.Value("N") = upstairHallway
  UpstairHallway.MovableDirections.Value("S") = girlBedroom

  ' Loop through the rooms
  ' Randomly add Seltzer to an item in a room
  Dim stuffList() As Stuff
  For Each r As Room In Rooms
    For Each s As Stuff In r.RoomStuff
      If s.StuffName = "Door" Or s.StuffName = "Gate" Then Continue
      stuffList.Append(s)
    Next
  Next

  stuffList.Shuffle
  stuffList(0).SeltzerIsHere = True
End Sub
```

The last bit of code is the code that starts the game and processes your commands. This code is in the Run event handler for the App object:

```
  Setup

  Print "Find Seltzer!"
  Print ""

  DisplayRoom

  Dim command As String

  Do
    Print ""
    Print "What would you like to do?"
    command = Input

    Select Case command
    Case "quit", "die"
```

```
    Quit
  Case "look"
    DisplayRoom
    Continue
  Case "?", "help"
    ShowCommands
    Continue
  End Select

  Dim commands() As String = command.Split(" ")

  If commands.Ubound >= 1 Then

    Dim verb As String = commands(0)
    Dim noun As String
    For i As Integer = 1 To commands.Ubound
      noun = noun + commands(i) + " "
    Next
    noun = noun.Trim

    Select Case verb
    Case "Quit"
      Quit
    Case "Go", "Move"
      ' Direction
      Dim direction As String = noun.Left(1)

      Dim newRoom As Room = CurrentRoom.Move(direction)
      If newRoom <> Nil Then
        CurrentRoom = newRoom
        DisplayRoom
        score = score + 1
      Else
        Print "You cannot move in that direction."
      End If

    Case "Look", "Examine"
      If CurrentRoom.LookAt(noun) Then
        Print "Game Over!"
        Print "Your Score is " + Score.ToText
        Quit
      End If
    Case Else
      Sorry(command)

    End Select

  Else
```

```
    Sorry(command)
  End If
Loop Until False
```

You can test the game by running it on your main computer. You can use the Remote Debugger to run the game on your Raspberry Pi.

Remember, because this is a console app you will have to manually start the app on the Pi from its terminal. For example, if the "Download Location" in your Remote Debugger Stub is set to the Desktop, you can use these commands in the Pi Terminal to navigate to the directory and run the app:

```
cd ~/Desktop/DebugFindSeltzer
```

```
./FindSeltzer
```

Files

This chapter covers the parts of the Xojo framework that you can use to load and save files to access data on the Internet.

Files

All file access is done using a class called FolderItem. A FolderItem is anything that can be stored on a drive such as volumes, folders, files, applications, and documents.

Using the FolderItem class, you can get a reference to any such items on your drives. To read from a file, you need a FolderItem for it. To write to a file, you need a FolderItem. When you ask users to select a file using one of the file selectors, you get a FolderItem referring to the file they selected.

Once you have a FolderItem, you can refer to its properties (such as Name or path) and perform actions on it such as deleting or copying it.

There are actually two separate FolderItem classes that are available for use. There is the classic FolderItem class which works with Desktop, Web and Console projects (but not iOS). There is also the Xojo.IO.FolderItem class that is part of the Xojo framework. This class is available for all project types, including iOS.

Although you cannot use these two classes interchangeably, it is easy to switch between them. To create a Xojo.IO.FolderItem from a classic FolderItem, you use the Constructor:

```
Dim userFile As FolderItem = GetOpenFolderItem("") ' classic FolderItem
Dim newFile As New Xojo.IO.FolderItem(userFile.NativePath.ToText)
```

And you do a similar thing to create a classic FolderItem from a Xojo.IO.FolderItem:

```
Dim newFile As New Xojo.IO.FolderItem("test.txt")
Dim classicFile As New FolderItem(newFile.Path, FolderItem.PathTypeNative)
```

You will only need to worry about switching between FolderItems when you are working with projects that use a mixture of the classic and Xojo frameworks.

Reading from a Text File

Once you have a FolderItem that represents an existing text file you wish to open, you open the file using the Open shared method of the TextInputStream class. This method is a function that returns a “stream” that carries the text from the text file to your application. The stream is called a TextInputStream. This is a special class of object designed specifically for reading text from text files. You then use ReadAll or ReadLine methods of the TextInputStream to get the text from the text file. The TextInputStream keeps track of the last position in the file you read from.

The TextInputStream.ReadAll method returns all the text from the file (via the TextInputStream) as Text. The ReadLine method returns the next line of text (the text after the last character read but before the next end of line character). As you read text, you can determine if you have reached the end of the file by checking the TextInputStream's EOF (end of file) property. This property will be True when the end of the file has been reached. When you are finished reading text from the file, call the TextInputStream's Close method to close the stream to the file, making the file available to be opened again.

This example lets the user choose a text file using the Open-file dialog box and displays the text in a TextArea:

```
Dim f FolderItem
f = GetOpenFolderItem("")
If f <> Nil Then
  Dim file As New Xojo.IO.FolderItem(f.NativePath)
  Dim stream As Xojo.IO.TextInputStream
  stream = TextInputStream.Open(file, Xojo.Core.TextEncoding.UTF8)
  TextArea1.Text = stream.ReadAll
  stream.Close
End If
```

Because ReadAll reads all of the text in the file, the resulting string will be as large as the file. Keep this in mind because reading a large file could require more memory than the system has available for the app.

This example reads the lines of text from a file stored in the Desktop folder and puts each line as a row in a ListBox:

```
Dim f As Xojo.IO.FolderItem
f = SpecialFolder.Desktop.Child("SampleFile.txt")
If f <> Nil And f.Exists Then
  Dim stream As Xojo.IO.TextInputStream
  stream = TextInputStream.Open(f, Xojo.Core.TextEncoding.UTF8)
  While Not stream.EOF
    ListBox1.AddRow(stream.ReadLine)
  Wend
  stream.Close
End If
```

Writing to a Text File

Once you have a FolderItem that represents the text file you wish to open and write to, you open the file using the Append shared method of the TextOutputStream class. If you are creating a new text file or overwriting an existing text file, use the Create shared method of the TextOutputStream class. These methods are functions that return a "stream" that carries the text from your application to the text file. The stream is called a TextOutputStream. This is a special class of object designed specifically for writing text to text files. You then use the WriteLine method of the TextOutputStream class to write the text to the text file.

The WriteLine method, by default, adds a carriage return to the end of each line.

When you are finished writing text to the file, call the TextOutputStream's Close method to close the stream to the file making the file available to be opened again. This prompts the user to select an existing text file and then adds the contents of three TextFields to the end of the text file and closes the stream:

```
Dim f FolderItem
f = GetOpenFolderItem("")
If f <> Nil Then
  Dim file As New Xojo.IO.FolderItem(f.NativePath)
  Dim stream As Xojo.IO.TextOutputStream
  stream = TextOutputStream.Append(file, Xojo.Core.TextEncoding.UTF8)
  stream.WriteLine(NameField.Text)
  stream.WriteLine(AddressField.Text)
  stream.WriteLine(PhoneField.Text)
  stream.Close
End If
```

If you want to create a new text file, then call TextOutputStream.Create instead. This example passes a default filename for the new text file:

```
Dim f As FolderItem
f = GetSaveFolderItem("", "CreateExample.txt")
If f <> Nil Then
  Dim file As New Xojo.IO.FolderItem(f.NativePath)
  Dim stream As TextOutputStream
  stream = TextOutputStream.Create(file, Xojo.Core.TextEncoding.UTF8)
  stream.WriteLine(NameField.Text)
  stream.WriteLine(AddressField.Text)
  stream.WriteLine(PhoneField.Text)
  stream.Close
End If
```

Graphical User Interfaces

A graphical user interface (GUI) is what you typically see when you use an app: the layout, with buttons and other controls. In order for your app to have a GUI, you'll need to create either a Desktop or Web app.

After starting Xojo and choosing either Desktop or Web to create your project, you'll see the Layout Editor with a list of controls in the Library on the right side of the main workspace window. You can drag controls from the Library onto the layout, positioning as you want.

The types of controls vary between desktop and web projects, but there are many similarities, such as Button, Listboxes, PopupMenus, etc. To work with a control, you implement its events. For example, a button has an Action event that is called when the button is clicked.

To create a simple "Hello, World" GUI app, drag a button onto the layout. With the button selected, click the "+" button in the command bar and select "Event Handler". Choose "Action" and click OK.

In the code editor, put this code to display a simple dialog:

```
MsgBox("Hello, World!")
```

Run the project and click the OK button to see the dialog.

You can learn more about the many controls available to you by reading the appropriate section of the User Guide:

- Desktop Controls
- Web Controls

Canvas

One of the most useful controls is the Canvas control, which you can use to draw just about anything on the screen. This can be text, pictures or graphics you draw yourself. You put the drawing code in the Paint event handler.

As a simple example, drag a Canvas onto a layout, resize it to use most of the layout area and add its Paint event handler using the same steps shown above to add the Action event handler to the button. In the Paint event handler, add this code to draw some simple shapes and text:

```
g.ForeColor = &cff0000 ' Red
g.FillRect(10, 10, 100, 50)

g.ForeColor = &c00ff00 ' Green
g.FillOval(100, 100, 100, 50)

g.ForeColor = &c0000ff ' Blue
g.DrawString("Hello, World!", 10, 100)
```

GUI Project - Music Player

In this project you will create a GUI Music Player that can play mp3 and aac (m4a) files from a folder.

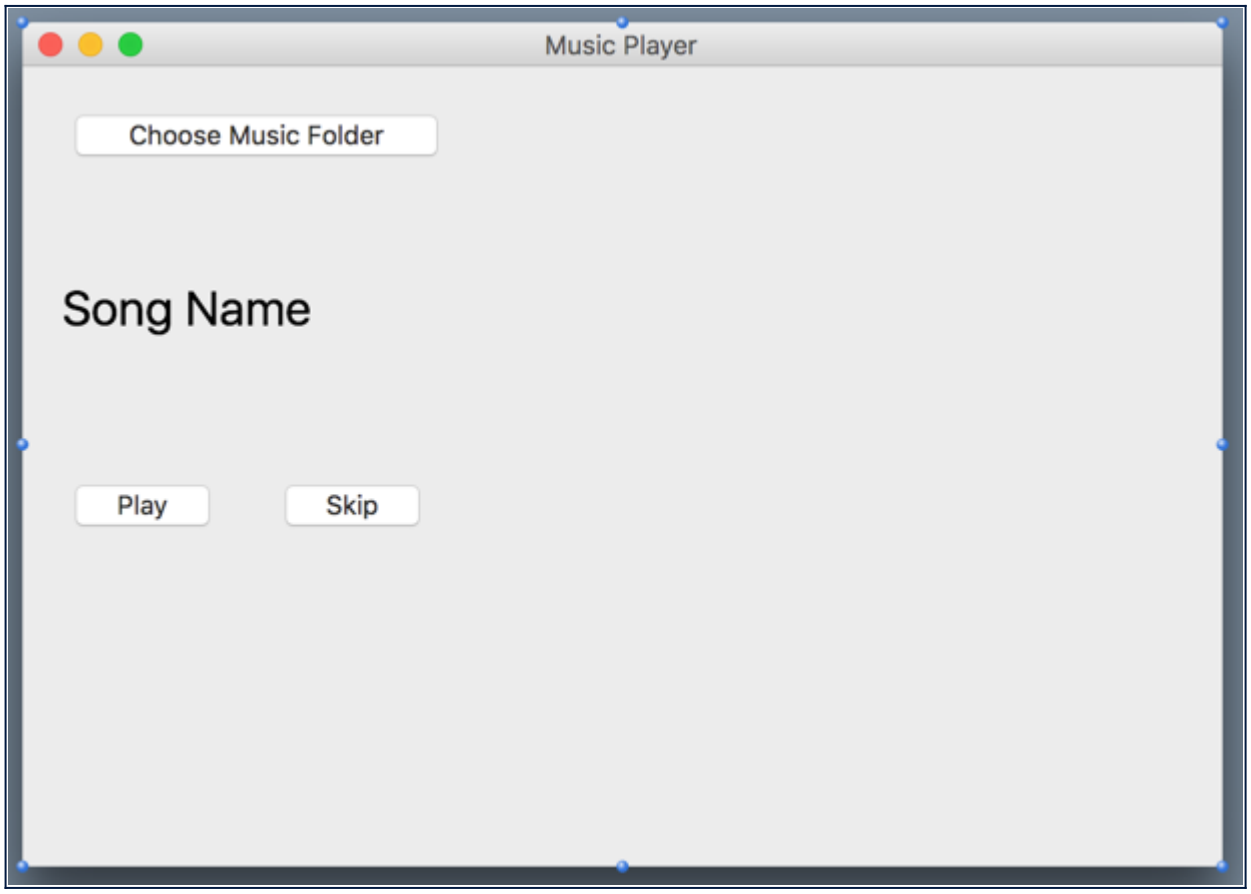
By default Raspberry Pi sound output is sent to the HDMI port. There is also a headphone port that can be used for sound output. To switch between the sound outputs, you use this command at the Terminal:

```
sudo amixer cset numid=3 1
```

Other values you can use are: 0=auto, 1=headphones, 2=HDMI.

Create the Project

Create a Desktop project and call it MP3Player. For controls, you will add three buttons and a label to Window1. Arrange them to look like this:



Change the name of the label from "Label1" to "NowPlayingLabel" and make it as wide as the window.

Now add these properties to keep track of the song files to play and the currently playing song sound:

- CurrentSong As Sound
- CurrentSongIndex As Integer
- Songs() As FolderItem

The music files are loaded as Sound objects and the CurrentSong property is the song that is currently playing. The

Songs() array is an array of the MP3 and AAC files that were loaded from the selected folder. And the CurrentSongIndex is the position of the current file (in the array) to play.

With this setup out of the way, you can start adding code. The first thing to do is to load the music files from the selected folder. Add the Action event to the "Choose Music Folder" button and enter this code:

```
' Choose a folder, grab all music files in the folder
' and save in array
Dim musicFolder As FolderItem = SelectFolder
If musicFolder <> Nil Then
  CurrentSongIndex = -1
  ReDim Songs(-1)

  Dim count As Integer = musicFolder.Count
  For i As Integer = 1 To count
    Dim f As FolderItem = musicFolder.Item(i)
    If f <> Nil And (f.Name.Right(3) = "mp3" Or f.Name.Right(3) = "m4a") Then
      Songs.Append(f)
    End If
  Next
End If

PlayNextSong
```

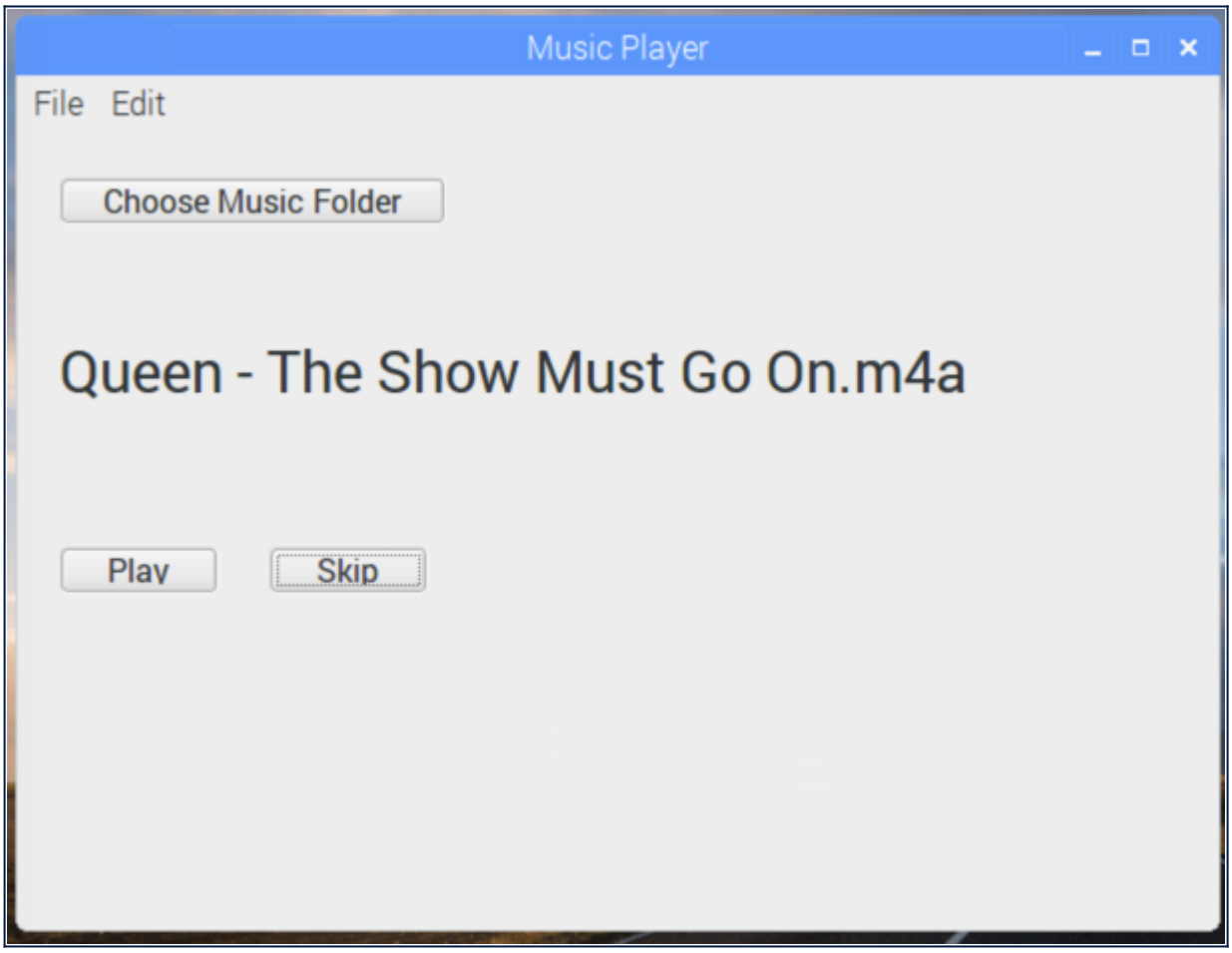
The last line calls a method, PlayNextSong, that starts playing the next song in the array. Here is the code for the PlayNextSong method:

```
Sub PlayNextSong()
  If CurrentSong <> Nil Then CurrentSong.Stop

  CurrentSongIndex = CurrentSongIndex + 1
  If CurrentSongIndex > Songs.Ubound Then CurrentSongIndex = 0

  If CurrentSongIndex <= Songs.Ubound Then
    CurrentSong = Songs(CurrentSongIndex).OpenAsSound
    CurrentSong.Play
    NowPlayingLabel.Text = Songs(CurrentSongIndex).Name
  End If

End Sub
```

Remember, do not type the Sub/End Sub part of the method in the code editor; instead enter that in the Inspector.

This code stops any currently playing song, increases the current song index by 1 (looping back to the start if the end is reached) and then loads the sound file from the array so it can be played. Lastly it updates the label with the name of the song file.

You can do a quick test with just this code in place. Run the project (using the Remote Debugger) on the Pi and choose a folder containing music files. The first song in the folder should start playing.



Adding Features

There are still two buttons on the window that don't yet do anything. The Play button will be used to Play and Stop the current song. Add the Action event to the Play button and enter this code:

```
If CurrentSong.IsPlaying Then
  CurrentSong.Stop
Else
  CurrentSong.Play
End If
```

The Skip button skips to the next song. Add the Action event and have it call the PlayNextSong method:

```
PlayNextSong
```

You can now again run the project on the Pi to test out these new features. When music is playing, click the Play button to stop and play. Click the Skip button to skip to and play the next song.

If you've let a song play until it finishes, you'll notice that the next song does not start playing. This is because there is no code to check if the current song has finished. If you recall the code in the Play button it checks a property called "IsPlaying" to see if the song is currently playing. That property could be also be used by a Timer to check if the song has finished playing so it could start playing the next song. However this won't quite be enough. If the user manually stops the song, then IsPlaying will be false and the Timer will automatically start playing the next song, which is probably not the behavior you want. To track when a user has manually stopped the music, another property is needed. Add this to Window1:

- IsSongStopped As Boolean

Now you can drag a Timer on to Window1 and add the Timer's Action event handler with this code (you can leave the Timer with its default period of 1000 and Mode to Multiple):

```
If CurrentSong <> Nil Then
  If Not CurrentSong.IsPlaying And Not IsSongStopped Then
    ' The song is no longer playing and the user has not manually stopped
    ' so that must mean the song has finished on its own.
    PlayNextSong
  End If
End If
```

And lastly, you need to go back to the Play button Action event handler and change its code to set the value in IsSongStopped like this:

```
If CurrentSong.IsPlaying Then
  CurrentSong.Stop
  IsSongStopped = True
Else
  CurrentSong.Play
  IsSongStopped = False
End If
```

Run the project on the Pi and you'll see that the next song starts playing about 1 second after the current song finishes.

Improvements

Try improving the music player with these changes:

- Change the text of the Play button to switch between Play/Stop depending on the state of the music.
- Change the text of the Skip button to show the name of the song that will be played next.
- Use the MoviePlayer control to play the music (instead of the Sound property) so you can get pausing, duration and other features.

Internet Access

This chapter covers the parts of the Xojo framework that you can use to load ans save files to access data on the Internet.

HTTPSocket

An HTTPSocket is used to connect to make Internet connections to web pages and web services. To use an HTTPSocket, you can drag a Generic Object from the Library and change its Super in the Inspector to Xojo.Net.HTTPSocket.

Requesting Data

The Send method is used to send a request to a web page or to a web service. Two common types of request methods are GET and POST. After you send a request, the results are available in the PageReceived event handler of the HTTPSocket.

This simple code sends a GET request to the demo web service that returns the list of customers in a database:

```
MySocket.Send("GET", "http://demos.xojo.com/EEWS/index.cgi/api/GetAllCustomers")
```

The Send method is asynchronous, which means that after you call the Send method your code continues running. When the data is returned, the PageReceived event on the HTTPSocket is called with the results.

This code in the PageReceived event collects the returned data (in JSON format) into a Dictionary:

```
' Convert binary data to JSON text
Dim jsonText As Text = Xojo.Core.TextEncoding.UTF8.ConvertDataToText(Content)

' Convert JSON text to a Dictionary
Dim json As Xojo.Core.Dictionary
json = Xojo.Data.ParseJSON(jsonText)

Dim customers As Xojo.Core.Dictionary
customers = json.Value("GetAllCustomers")

' Loop through all the customers in the Dictionary and get the first name
For Each entry As Xojo.Core.DictionaryEntry In customers
  Dim custDict As Xojo.Core.Dictionary = entry.Value
  Dim firstName As Text = custDict.Value("FirstName") ' display or otherwise use
Next
```

Submitting Data

You also use the Send method to submit data, but you typically use the POST request method and supply the data using the SetRequestContent method. Using the demo web service from above, you can request all the information

for a specific customer:

```
Dim cust As New Xojo.Core.Dictionary
cust.Value("ID") = 10179 ' the specific customer

Dim json As Text
json = Xojo.Data.GenerateJSON(cust)

Dim data As Xojo.Core.MemoryBlock
data = Xojo.Core.TextEncoding.UTF8.ConvertTextToData(json)

MySocket.SetRequestContent(data, "application/x-www-form-urlencoded")
MySocket.Send("POST", "http://demos.xojo.com/EEWS/index.cgi/api/GetCustomer")
```

The resulting customer information is available in the PageReceived event and can be converted from JSON to a Dictionary for use in your project:

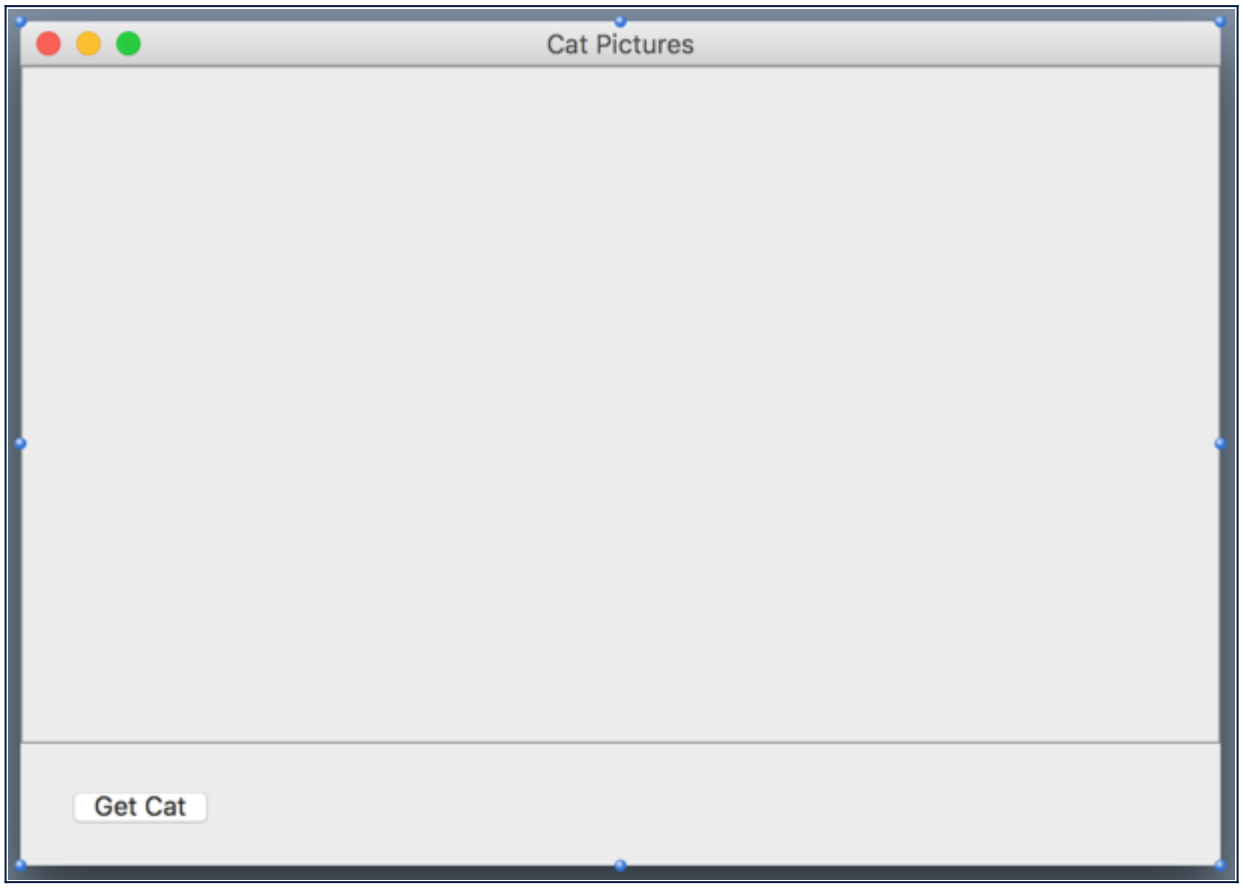
```
Dim jsonText As Text = Xojo.Core.TextEncoding.UTF8.ConvertDataToText(Content)

Dim json As Xojo.Core.Dictionary
json = Xojo.Data.ParseJSON(jsonText)

Dim custInfo As Xojo.Core.Dictionary
custInfo = json.Value("GetCustomer")

Dim id As Integer
Dim firstName, lastName As Text
For Each entry As Xojo.Core.DictionaryEntry In custInfo
  Select Case entry.Key
  Case "ID"
      id = Integer.FromText(entry.Value)
  Case "FirstName"
      firstName = entry.Value
  Case "LastName"
      lastName = entry.Value
  End Select
Next
```

Cat Pictures

In this project you will create an app that gets and displays cat pictures from the Internet. Your app will get pictures from the Internet using a web site called TheCatAPI. This website gives you a different cat picture each time you call it. The process for your app will be to call the web site (using a special URL for accessing its "web service") when a button is clicked and then display the picture in a Canvas control on the window after it is received.

Create the Project

Create a new desktop project and add a Canvas and button to Window1. Resize the Canvas so it uses up most of the window area so your layout looks like this:



Now add a property to store the cat picture after it is received so that it can be displayed:

- CatPicture As Picture

Next you need to add an HTTPSocket to connect to the Internet and receive the cat pictures. Drag the "Generic Object" from the Library onto the Window1 layout. This adds it at the bottom of the layout (called the Shelf). Click on it and display the Inspector. Change its name from "Object1" to "CatSocket" and its Super from "Object" to "HttpSocket".

It time to add the first bit of code that displays the picture that was received. Right-click on the CatSocket object in the Shelf and select "Add to CatConnector" in the menu, from the submenu select "Event Handler". This displays

the Event Handler window. In addition to being object-oriented, Xojo is event-based. Events are actions that occur when something happens in your app, such as the user tapping on the UI or data being received from the Internet.

In the list of event handlers, you want to click the PageReceived and then click the OK button to add the event handler to CatSocket. The PageReceived event handler is what is called when the picture is received from the web service. If you look back at the CatSocket in the Navigator on the left, you'll now see that "PageReceived" is displayed below it. This is where you will put the code to display the picture.

Click on "PageReceived" to display the Code Editor where you can enter this code:

```
' Convert Xojo.Core.MemoryBlock to MemoryBlock for use by Picture
Dim temp As MemoryBlock  = Content.Data
Dim mb As New MemoryBlock(Content.Size)
mb.StringValue(0, mb.Size) = temp.StringValue(0, mb.Size)

CatPicture = Picture.FromData(mb)
Canvas1.Invalidate(False)
```

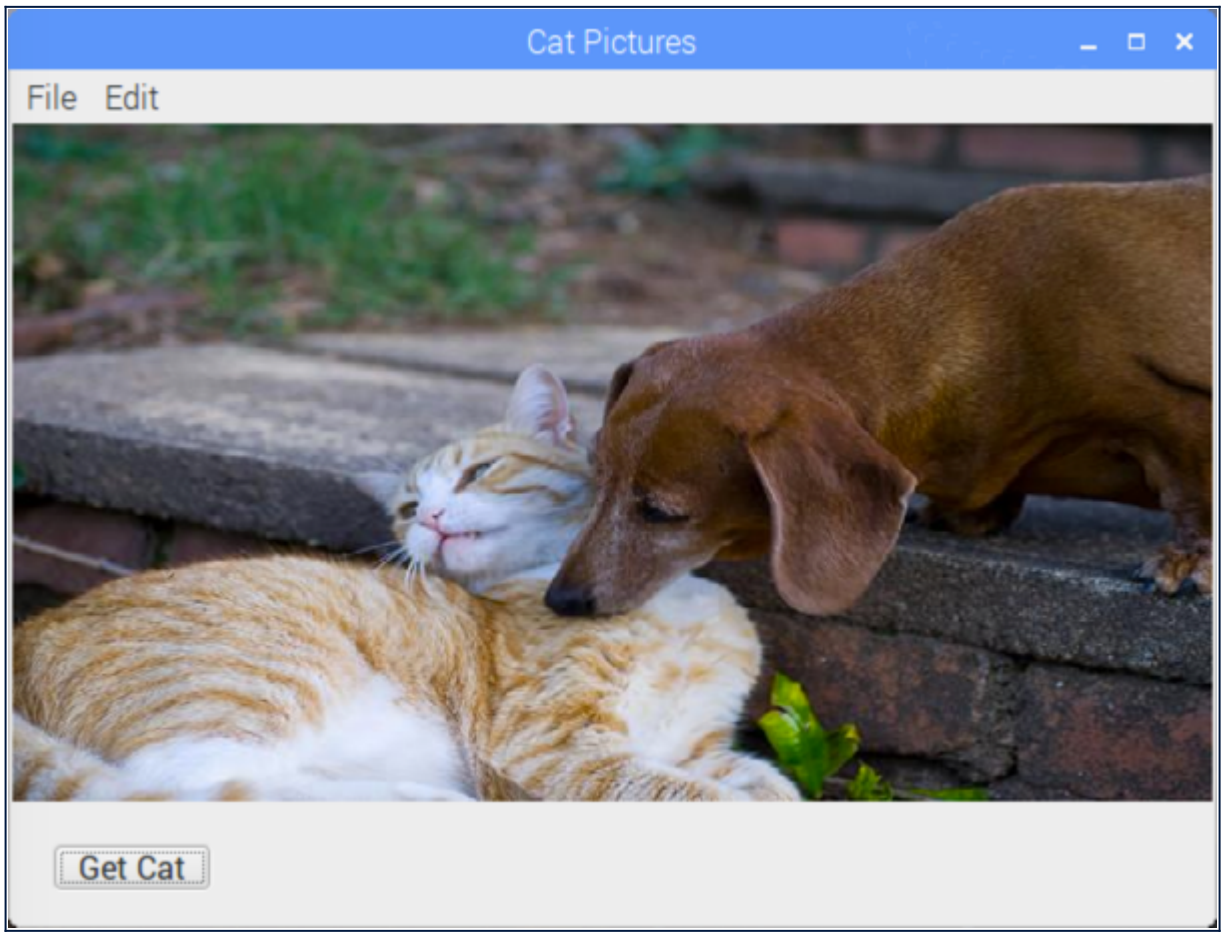
This code converts the data received by the CatConnector to a Picture, then tells the Canvas to refresh itself. Speaking of the Canvas, you need to tell it how to draw the picture. Add the Paint event to Canvas1 with this code:

```
If CatPicture <> Nil Then
  g.DrawPicture(CatPicture, 0, 0, g.Width, g.Height, 0, 0, CatPicture.Width,
CatPicture.Height)
End If
```

Lastly, add the Action event to the button with this code to request the picture from the web service:

```
CatSocket.Send("GET", "http://thecatapi.com/api/images/get")
```

You can test this on your main computer or run it on the Pi using the Remote Debugger. Click the Get Cat button to fetch and display a cat picture.



Improvements

Here are some suggestions for improvements to the Cat Pictures app:

- Set the locking for the Canvas so that the picture window (and thus cat picture) can be resized.
- Add the ability to save the picture to a file. Look at the Picture.Save method.

Making a Game

You can use the Canvas control described in the Graphic User Interface section and used by the Cat Pictures project to also make games. In addition to the Canvas control you typically use a Timer control to handle animation and redrawing of the Canvas.

As a quick example, in this sectionyou'll learn how to make a simple Catch Xojo game. In this game, Xojo logos will drop from the top of the screen and it is your job to catch has many as you can.

Game Setup

This game is a desktop app, so create a new desktop project and drag a Canvas to the layout, resizing it to fill the available area. Change its name (using the Inspector) to GameCanvas.

You also need to add the image that will drop. You can use any image you want or just drag the XojoLogoSmall.png file into your project.

The next things to add to the layout are two Timers. Drag the first timer to the layout and change its name to AddItemTimer and its Period to 1000. This Timer adds new XojoLogos to the screen at a rate of 1 per second (1000 milliseconds). Drag a second timer to the layout and change its name to DropTimer and its Period to 20. This Timer moves the XojoLogos down the screen.

The last thing to add to the project is a class that will represent these items that will appear on the screen for you to catch. Add a class to the project and name it "CatchItem".

Now you are ready to start adding code.

Adding Code

The first code to add will simply draw Xojo logos at the top of the screen and have them drop to the bottom.

To start, first you add two properties to the CatchItem class:

- X As Integer
- Y As Integer

These properties are used to track the position of the item on the screen.

Now you can add a new method to the CatchItem class. Name it "Constructor". This method is called when a new CatchItem is created and is used to initialize it. This is the code:

```
Sub Constructor(maxWidth As Integer)
  X = Xojo.Math.RandomInt(0, maxWidth - XojoLogo.Width)
  Y = 0
End Sub
```

This code chooses a random number for the X (horizontal) position of the item and starts it at the top (Y = 0) of the screen.

You can now add a Move method to move the item down the screen:

```
Sub Move()
  Y = Y + 2
End Sub
```

The last method to add is the Draw method which simply draws the item on the screen:

```
Sub Draw(g As Graphics)
  g.DrawPicture(XojoLogo, X, Y)
End Sub
```

You'll notice this method takes a parameter. This parameter will be supplied by code on the Canvas.

You now need to go back to Window1 to add the code to display and move your CatchItem. First, you'll want to add a couple properties to the window:

- ItemsToCatch() As CatchItem
- Score As Integer

ItemsToCatch is an array that contains all the CatchItems to display. Score will eventually be used to count the items that you catch.

Now select the AddItemTimer on the Window and add its Action event handler. As noted earlier, this Timer has a Period of 1000 milliseconds, which means that the Action event is called about once a second. Each time it is called, you can add a new item to display on screen with this code:

```
Dim item As New CatchItem(GameCanvas.Width)
ItemsToCatch.Append(item)
GameCanvas.Invalidate(False)
```

This code creates a new CatchItem (supplying the width to the constructor so that it knows where it can be positioned) and then adds the new item to the ItemToCatch array property. This does not display the item on the screen; it just adds it to the array of items to display on the screen. Displaying the item on the screen is done by the Canvas and you tell the Canvas to draw by calling the Invalidate method to indicate that its contents are out-of-date and should be redrawn. This calls the Canvas Paint event to get called and this is the code you'll add next.

Select the GameCanvas and add its Paint event handler with this code:

```
For i As Integer = ItemsToCatch.Ubound DownTo 0
  ItemsToCatch(i).Draw(g)
  If ItemsToCatch(i).Y > g.Height Then
    ' Remove items once they fall off the screen
    ItemsToCatch.Remove(i)
```

```
  End If
Next
g.DrawString("Score: " + Str(Score), 10, 10)
```

This code loops through the items in the ItemsToCatch array (last to first) and tells each one to draw by calling its Draw method. After drawing, it checks the Y position to see if it would be off the screen and if it is, then the item is removed from the array so that it no longer gets drawn. Lastly, the score (always 0 right now) is drawn in the top left of the screen.

You're not quite done, but you may want to run the project now to see what it does. You can actually just run it on whatever computer you are using; you don't have to transfer it to the Pi just yet. When you run it, you should see a bunch of Xojo logos appear at the top of the screen, but not move. That's because you haven't added the last bit of code to tell the items to move. Quit the app and back in Xojo, select the DropTimer and add this code:

```
For Each item As CatchItem In ItemsToCatch
  item.Move
Next
GameCanvas.Invalidate(False)
```

This code loops through all the items in the CatchItem array and calls each of their Move methods. Then it tells the Canvas to update itself, because all the items no have new positions and need to be redrawn.

Run the project again and you'll see items appear at the top of the screen and drop down to the bottom.

You are now nearly done! The next code to add is to detect when you "catch" an item. For this game, you'll catch an item by simply clicking on it. In order to determine if an item has been clicked, you'll want to add a new method to the CatchItem class:

```
Function IsCaught(mouseX As Integer, mouseY As Integer) as Boolean
  Dim itemRect As New Xojo.Core.Rect(X, Y, 50, 50)
  Dim mousePoint As New Xojo.Core.Point(mouseX, mouseY)

  Return itemRect.Contains(mousePoint)
End Function
```

This method takes as parameters the coordinates of where the mouse was clicked. It then creates a Rect for the current position of the item and a point for the mouse click position. If the Rect contains the mouse point, then IsCaught returns True.

Now you need to determine when the mouse was clicked.

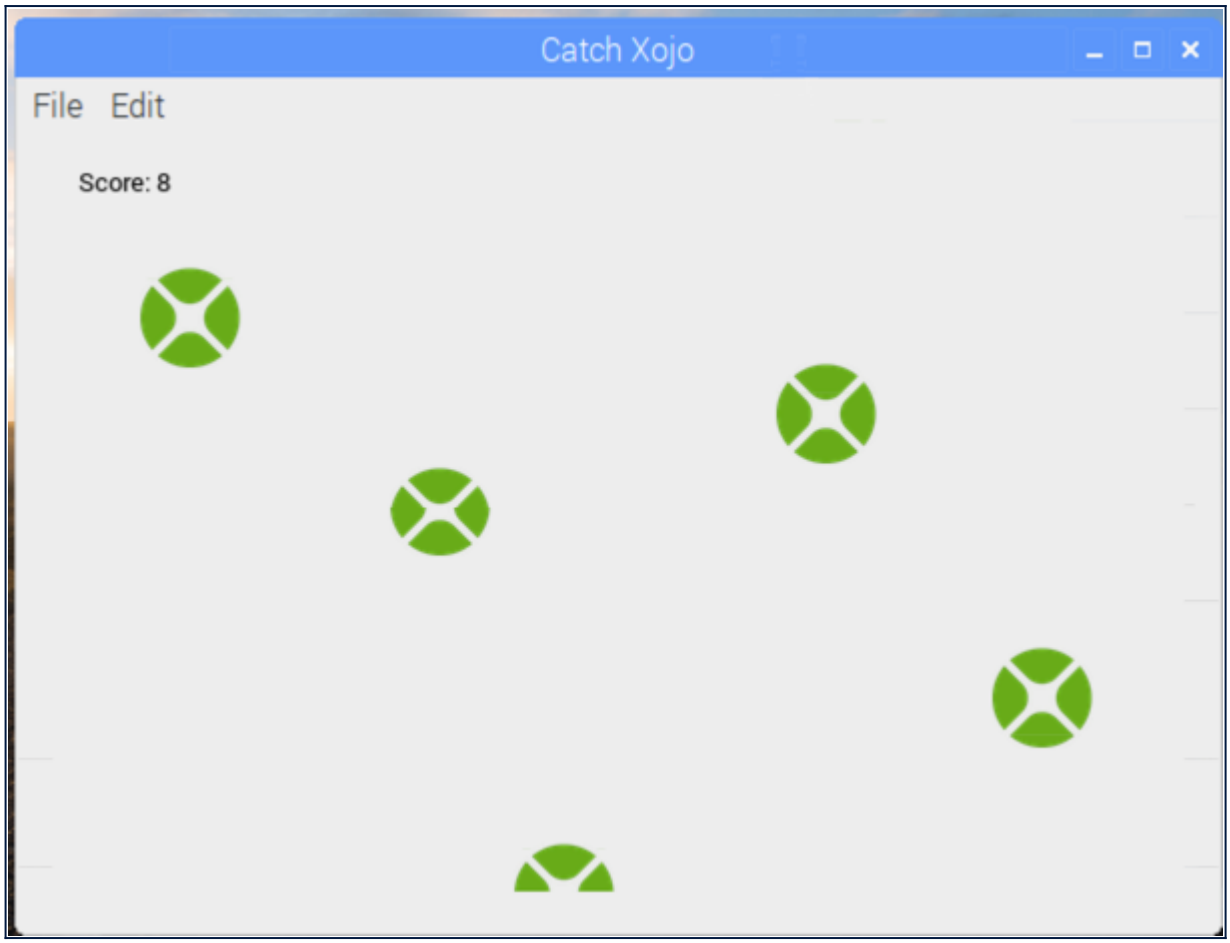
The Canvas control has just an event for this, so add the MouseDown event handler to it and enter this code:

```
For i As Integer = ItemsToCatch.Ubound DownTo 0
  If ItemsToCatch(i).IsCaught(X, Y) Then
    ' When caught, increase the score and remove the item
```

```
    Score = Score + 1
    ItemsToCatch.Remove(i)
    Me.Invalidate(False)
  End If
Next
```

This code is similar to the Paint event code. It also loops through all the items in the ItemsToCatch array and calls the IsCaught method to see if it was caught. If it was caught, the score is increased and it is removed from the array so it no longer gets drawn.

And that's it! You've made your first simple game that you can play on the Raspberry Pi. You can run it on the Pi using the Remote Debugger or by building and transferring the app.



Improvements

Here are suggestions for improvements and changes you can make to the game:

- To increase the difficulty, add new items more frequently than 1 per second.
- Or, instead have the difficulty increase as the game progresses. You could increase the frequency of new items each time an existing one is caught.
- Have the game end when 3 items reach the bottom and are not caught.
- Advanced: Try changing the project so that the items move in different directions rather than just from the top

of the screen to the bottom. Maybe you can even have them bounce around when they reach the edges!

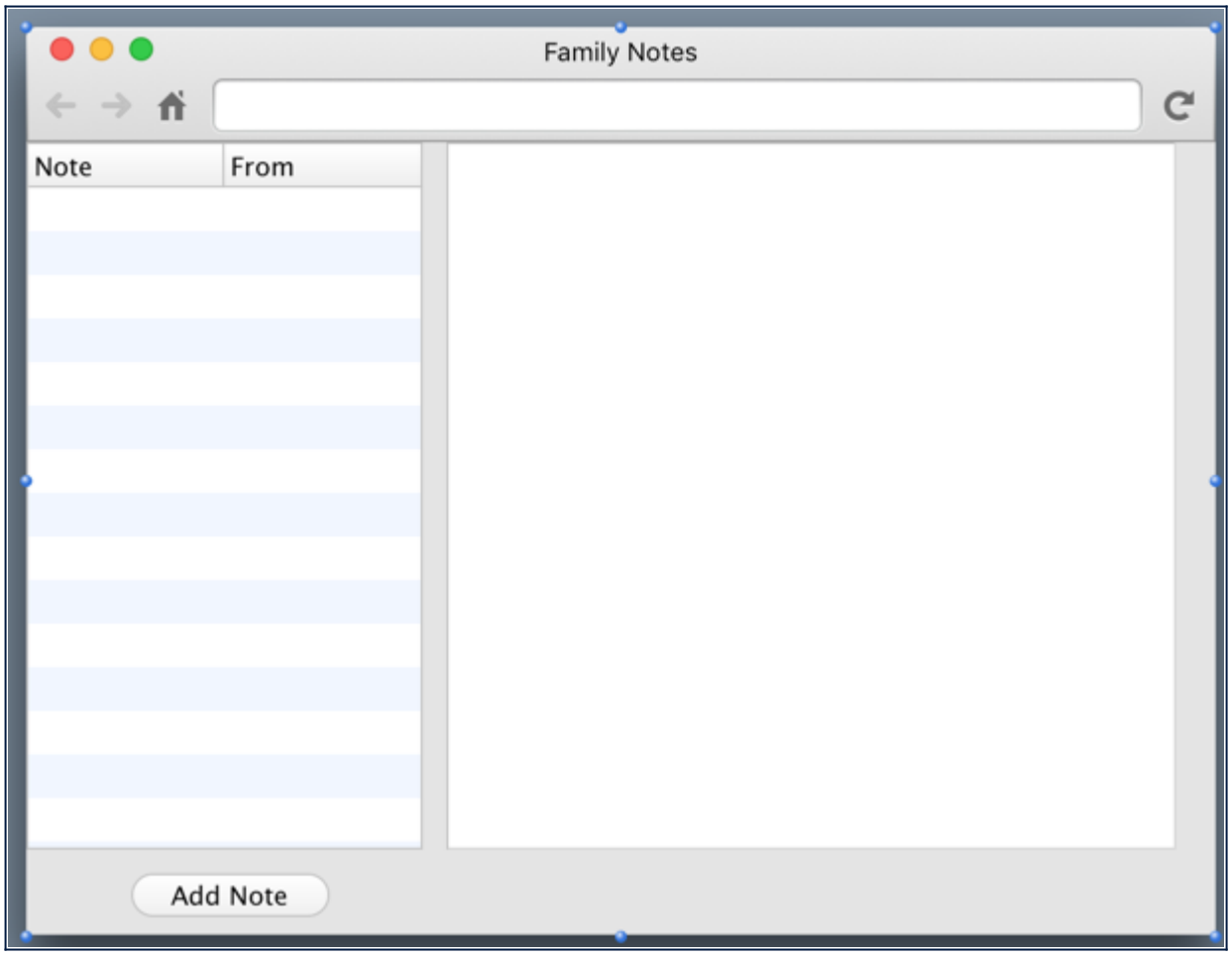
Web Project - Family Notes

With Xojo you can also make web apps that use the Raspberry Pi as a simple web server that other devices, such as computers, phones and tablets can connect to.

To create a web app you design the layout using the built-in controls much like you would for a desktop app. For this project, you will create a "Family Notes" web app. This app is meant to run on your home wifi and be accessible to devices connected to the wifi. With it, members of you family can leave messages to one another, similar to how you might use a magnetic white board attached for a refridgerator.

Create the Project and User Interface

Start by creating a new web project and give it the name "FamilyNotes". For its user interface, this project uses a WebPage and a WebDialog. Starting with the WebPage, you want to have a ListBox, a TextArea and a Button arranged to look like this:



Change the names for each of the controls as shown below:

- ListBox1: NoteList
- TextArea1: NoteArea

- Button1: AddNoteButton

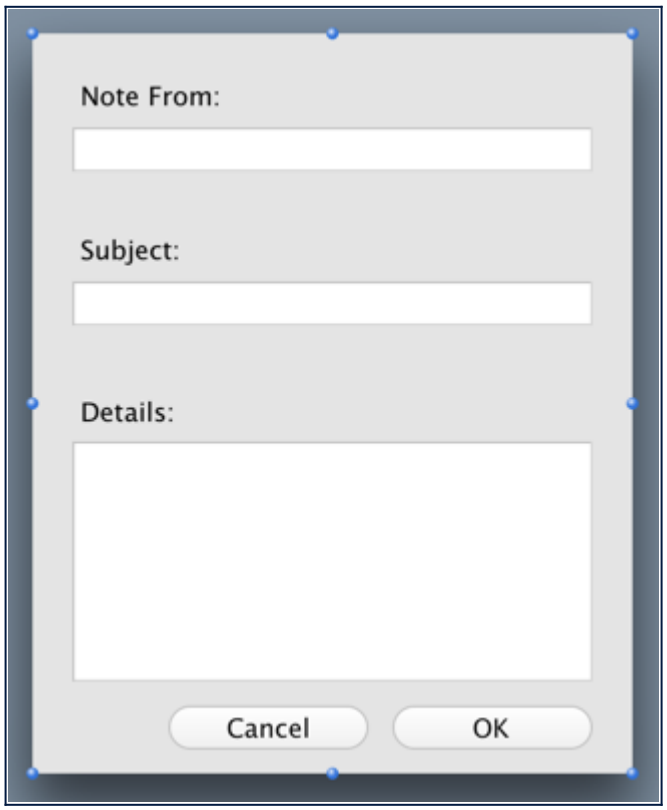
To change the heading on the ListBox, click on the ListBox and then select the "pencil" icon in the lower right. This opens a popup editor where you can double-click on the header row to change the header text.

In order for the controls and the WebPage to be properly positioned on devices of all sizes, you'll also want to adjust a few properties. First, on WebPage1 change the Min Width to 320 and the Min Height to 200. This allows the web page to display correctly on phone-sized devices.

Next you want to change the "locking" for the controls so they resize when the web page size changes. You change the locking by clicking on the locks in the Locking area of the Inspector when the control is selected. Here is how you want the locking set for each of the controls:

- NoteList: Click the left, top and bottom locks so they are "locked". The right lock should be unlocked.
- NoteArea: Click all four locks so they are locked.
- AddNoteButton: Click the left and bottom locks so they are locked. The other locks should be unlocked.

The next part of the user interface to create is the dialog for adding a new note. Select Insert -> Web Dialog and create a web dialog with three Labels, two TextFields, one TextArea and two Buttons arranged to look like this:



Change the names of the controls as shown below:

- TextField1: FromField
- TextField2: SubjectField
- TextArea1: DetailsArea
- PushButton1: CancelButton
- PushButton2: OKButton

Now that you are finished with the UI, you can now move on to writing code.

Add Code

The notes that are created will be saved in an array of Dictionaries that are periodically saved as a JSON file. Since everyone that connects to the web app needs to access the notes, the array of dictionaries should be global to the entire web app. Create a public property on the App object:

- Notes() As Xojo.Core.Dictionary

The web page needs code to display the notes in the array and to display the dialog to add new notes. Start by adding a method, LoadNotes:

```
Sub LoadNotes()
  ' Display all the notes

  NoteList.DeleteAllRows

  App.Notes.Sort(AddressOf DateCompare)

  For Each n As Xojo.Core.Dictionary In App.Notes
    NoteList.AddRow(n.Value("Subject"), n.Value("From"))
    NoteList.RowTag(NoteList.LastIndex) = n.Value("Details")
  Next
End Sub
```

This method loops though all the notes in the Dictionary and adds them to the ListBox. Notice that it also adds the note details to the RowTag so that when a row is clicked, it can be easily retrieved to display in the text area on the web page.

The Sort method call uses something called a "Delegate", which is simply a reference to another method. In this case the method is used to sort the dictionary so that the most recently added notes appear first. This is the code for the DateCompare method:

```
Function DateCompare(value1 As Xojo.Core.Dictionary, value2 As Xojo.Core.Dictionary)
as Integer
  ' Used to sort the notes so that the most recent notes
  ' appear first.
  Dim d1 As Double = value1.Value("Timestamp")
  Dim d2 As Double = value2.Value("Timestamp")

  If d1 < d2 Then Return 1
  If d1 > d2 Then Return -1
  Return 0
End Function
```

With this code in place, you can now have the web page call it when it is displayed. Add the Shown event handler to the web page and call the LoadNotes method:

```

```

```
LoadNotes
```

It's great that the code is in place to load the notes, but there are no notes yet since you have not created a way to add them. Notes are created using the EditNoteDialog so you now need to add a way to display the dialog. The first step is to actually put the dialog on the page. To do this, select the web page and then drag EditNoteDialog from the Navigator onto the web page. It will appear on the "shelf" area at the bottom. Click on it and change its name to "EditDialog".

The user clicks the "Add Note" button to display the EditDialog, so add the Action event handler to the "Add Note" button with this code:

```

EditDialog.Show
```

While you are still on the web page, the last code to add it to tell the ListBox to reload all the notes after the EditDialog is closed. To do this, add the Dismissed event to EditDialog with this code:

```

LoadNotes
```

Now you are ready to add code to EditNoteDialog. The first thing to do is to add the Action event to the Cancel button with this code to close the dialog:

```

Self.Close
```

Next, add the Action event to the OK button. The code here will create a new dictionary, add the values from the fields to it, append the dictionary to the global array, and close the dialog:

```
Dim n As New Xojo.Core.Dictionary
n.Value("From") = FromField.Text
n.Value("Subject") = SubjectField.Text
n.Value("Details") = DetailsArea.Text
n.Value("Timestamp") = Xojo.Core.Date.Now.SecondsFrom1970

App.Notes.Append(n)

Self.Close
```

Each time the dialog is opened for the same user, the fields will contain the values that were previously entered. It makes sense to keep the same value for the FromField, but it probably makes sense to clear the other two. Add the Shown event to the dialog with with this code:

```

SubjectField.Text = ""
DetailsArea.Text = ""
```

You are just about done. The last thing to do is to make sure the notes do not get lost should you quit the web app.

One way to do this is to periodically save the array of dictionaries to a JSON file. You don't really need to save every time a change is made, but you do want to save it periodically. A good way to ensure that changes get saved is to save the JSON file whenever a session is disconnected. This can happen when the user closes the page/tab or when they refresh the page.

Go to the Session object in the Navigator and add the Close event with this code to call a method you will add next:

```
SaveNotes
```

Now you can add the SaveNotes method to the Session:

```
Sub SaveNotes()
  Dim file As Xojo.IO.FolderItem
  file = Xojo.IO.SpecialFolder.Documents.Child("FamilyNotes.json")

  Dim outputStream As Xojo.IO.TextOutputStream
  outputStream = Xojo.IO.TextOutputStream.Create(file, Xojo.Core.TextEncoding.UTF8)

  Dim jsonText As Text = Xojo.Data.GenerateJSON(App.Notes)

  outputStream.Write(jsonText)
  outputStream.Close

End Sub
```

The companion method is a way to load the notes when the web app first starts. You can do that by adding the Open event to the App object with this code to call a LoadNotes method you will add next:

```
LoadNotes
```

Now you can add the LoadNotes method to the App:

```
Sub LoadNotes()
  ' Load notes for the user from a json file

  Dim file As Xojo.IO.FolderItem
  file = Xojo.IO.SpecialFolder.Documents.Child("FamilyNotes.json")

  If file.Exists Then

    Dim inputStream As Xojo.IO.TextInputStream
    inputStream = Xojo.IO.TextInputStream.Open(file, Xojo.Core.TextEncoding.UTF8)
    Dim jsonText As Text = inputStream.ReadAll

    Dim jsonArray() As Auto
    jsonArray = Xojo.Data.ParseJSON(jsonText)
```

```
    For Each d As Xojo.Core.Dictionary In jsonArray
      Notes.Append(d)
    Next
  End If
End Sub
```

And that is it. You can run the project on your own computer to test it. When you do this it will automatically start your default web browser with the app open. You can open the app in other browser or other devices on your local network by entering the local IP address of the computer. For example, if the local IP address is 10.0.1.199, then you can go to an iPhone and enter this as the URL in Safari: http://10.0.1.199:8080

Deploy the Web App

To use the Pi as a web server for this app, you will want to build the web app as a Standalone app for ARM 32-bit.

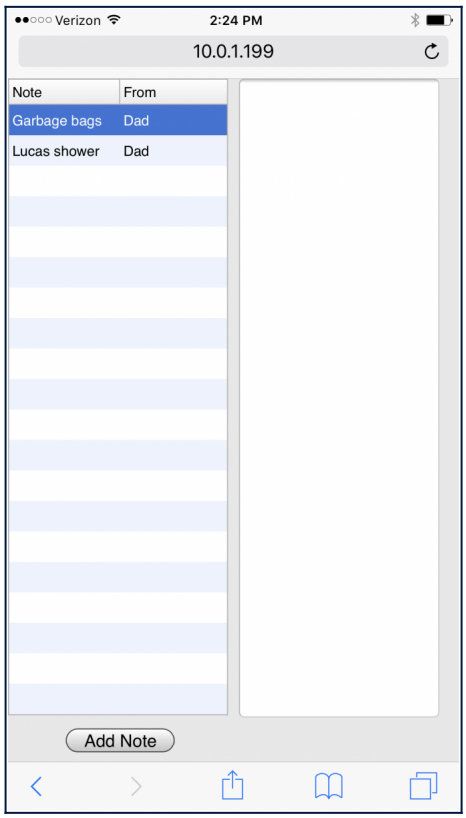
Click on Shared in the Build Settings section of the Navigator and change the Deployment Type setting to "Stand Alone". In the Linux build setting change the Architecture to "ARM 32-bit". Now you can click the Build button (this does require a Xojo license) to build the web app.

You can transfer the folder containing the app to the Pi and then navigate to it in the Terminal to launch it with this command:

```
./FamilyNotes
```

Once it is running, you can connect to the web app by using the local IP address of the Raspberry Pi, such as: http://10.0.1.199:8080

This is the app shown on Safari on an iPhone:



Improvements

- Use Styles to improve the look and feel of the app.
- Add a way to remove notes from the list.
- Currently you have to refresh the page in order to see any new notes that were added. You can instead have the app automatically update the page when new notes are added. You need to loop through all the connected sessions and call the LoadPages method for each webpage that is connected.

Interfacing Hardware with GPIO

GPIO is the General Purpose Input/Output port on the Raspberry Pi. You can use this 40-pin port to connect external hardware to the Pi. You can connect all kinds of things to the Pi using its GPIO port, including LEDs, motors, displays and pretty much anything you can think of. Sites such as AdaFruit and CanaKit have lots of hardware and gadgets you can connect to the Pi.

wiringPi with Xojo

Xojo uses wiringPi to communicate with the GPIO port. In order to use wiringPi with Xojo you'll need to install the open-source wiringPi library on your Pi. Instructions for this are here:

Install wiringPi Library

In your Xojo projects, you'll need to use the Xojo GPIO library which provides access to most of the wiringPi functionality for you to use in your Pi projects.

The GPIO library is available on GitHub:

https://github.com/xojo/GPIO

To add the GPIO library to your projects, download it from GitHub, open the project and select the WiringPiXojo module in the Navigator. You can then copy and paste it to your own Xojo projects.

The WiringPiXojo module provides support for

- GPIO pin control
- RGB LED
- LCD panel
- Tones
- Servos
- Pulse-width modulation

wiring Pi Docs

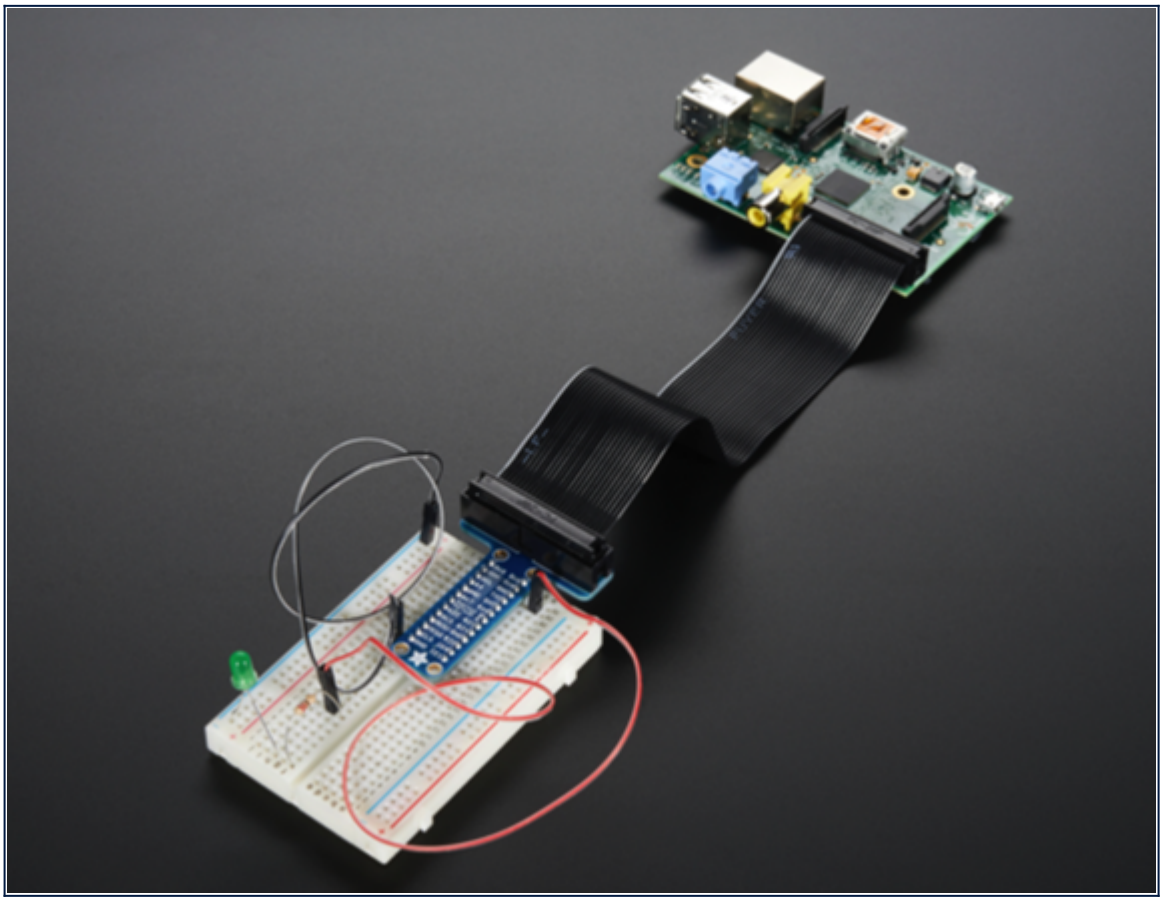
The official docs for wiringPi will not be replicated here, but links are provided for your reference:

- Setup methods
- Core Functions
- Raspberry Pi Specifics
- Timing
- Priority, Interrupts and Threads
- Serial (not yet implemented in Xojo GPIO module; use the Serial class instead)
- SPI Library
- I2C Library
- Shift Library
- Software PWM Library
- Software Tone Library

Connecting to GPIO

In most cases you will not connect wires directly to the GPIO port as that is pretty tricky and error-prone. Instead you will want to connect the pins on the GPIO port to a breadboard by using a cobbler and ribbon cable to make it much easier to do your wiring.

A breadboard is the place where you will wire your circuit. The ribbon cable and cobbler are essentially used to extend the GPIO port pins to the breadboard. On the breadboard you can wire everything without soldering. You just plug things into the holes on the breadboard (no need to solder). Everything is also numbered and labelled which makes it much easier to make sure you are hooking things up properly.



GPIO Pin Numbering

There are several ways that you can refer to the pins on the GPIO port. You typically do not use the specific pin number (0-40) but instead refer to the Broadcom (BCM) pin numbers. In a chart as shown below, these are the numbers you see that are referred to as "GPIO xx".


support.canakit.com
CanaKit electronics
Raspberry Pi GPIO Header Quick Reference
3.3V
5V
I²C SDA GPIO 2
I²C SCL GPIO 3
5V
GND
GPIO 4
GPIO 14 TXD UART
GND
GPIO 15 RXD
GPIO 17
GPIO 18 CLK PCM
GPIO 27
GND
GPIO 22
GPIO 23
3.3V
GPIO 24
SPI MOSI GPIO 10
GND
SPI MISO GPIO 9
GPIO 25
SPI SCLK GPIO 11
GPIO 8 CE0 SPI
GND
GPIO 7 CE1
ID SD
ID SC
GPIO 5
GND
GPIO 6
GPIO 12
GPIO 13
GND
PCM FS GPIO 19
GPIO 16
GPIO 26
GPIO 20 DIN PCM
GND
GPIO 21 DOUT
See http://www.canakit.com/quick-start/pi for more information

GPIO Project - Blinking LED

In this project, you will create a simple circuit with an LED and then make the LED blink using a Xojo app.

Parts

In order to build the circuit, you'll need some parts:

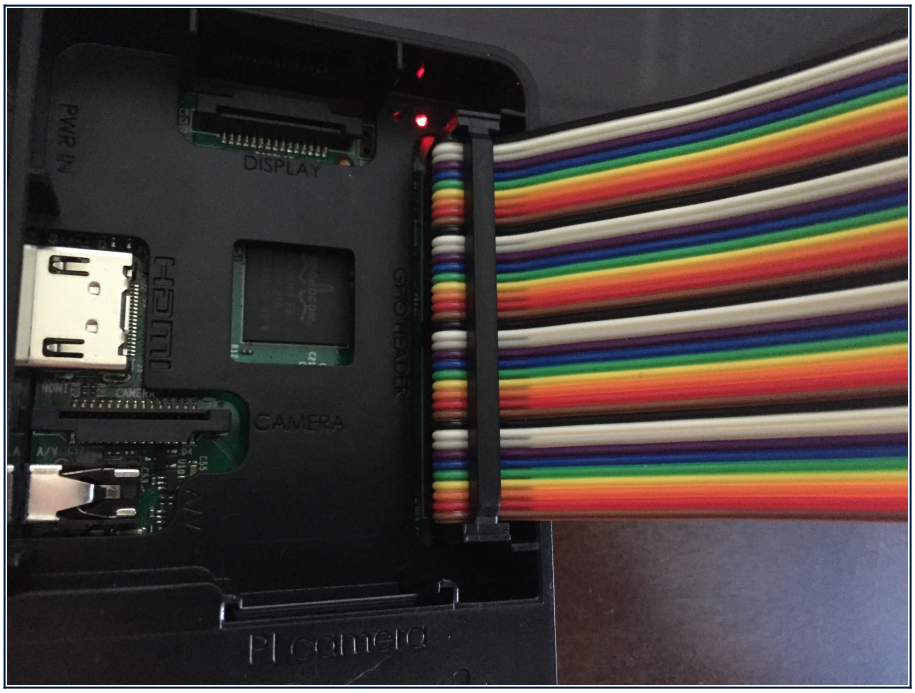
- 1 ribbon cable with Raspberry Pi GPIO connectors (AdaFruit link)
- 1 cobbler (AdaFruit link)
- 1 breadboard (AdaFruit link)
- 3 jumper wires (AdaFruit link)
- 1 LED (AdaFruit link)
- 1 10k resistor (SparkFun link)

In order to make the LED blink with Xojo, you'll need to install the wiringPi library and grab the WiringPiXojo module from GitHub:

- https://github.com/xojo/GPIO

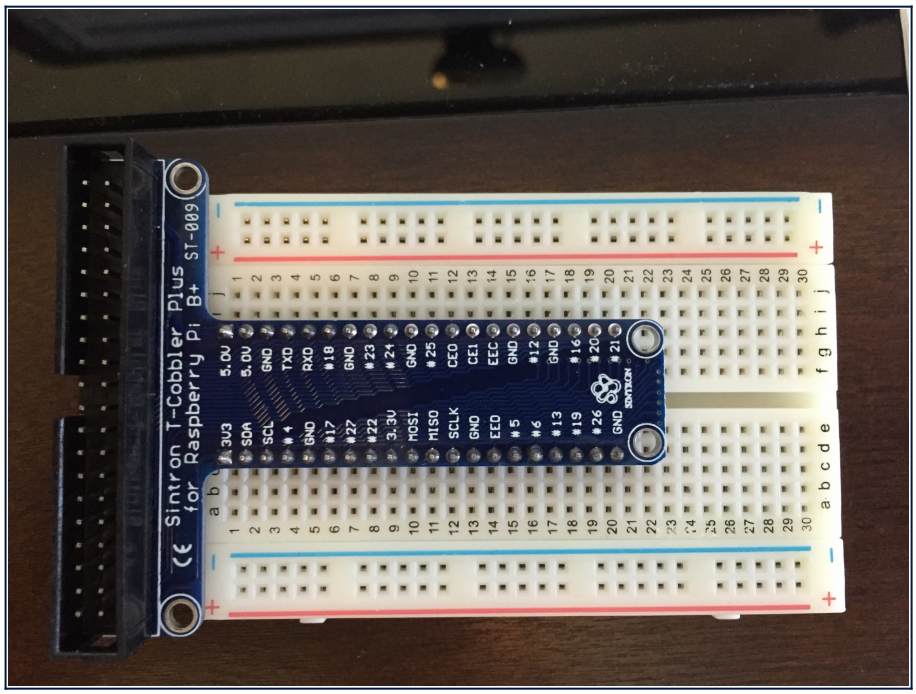
Connect Ribon Cable to Pi

First, connect your ribbon cable to the Raspberry Pi. The white/black side is typically facing the side of the Pi that does not have connectors. There is not a lot of room to work so take your time and make sure the pins are all aligned before you push it down.



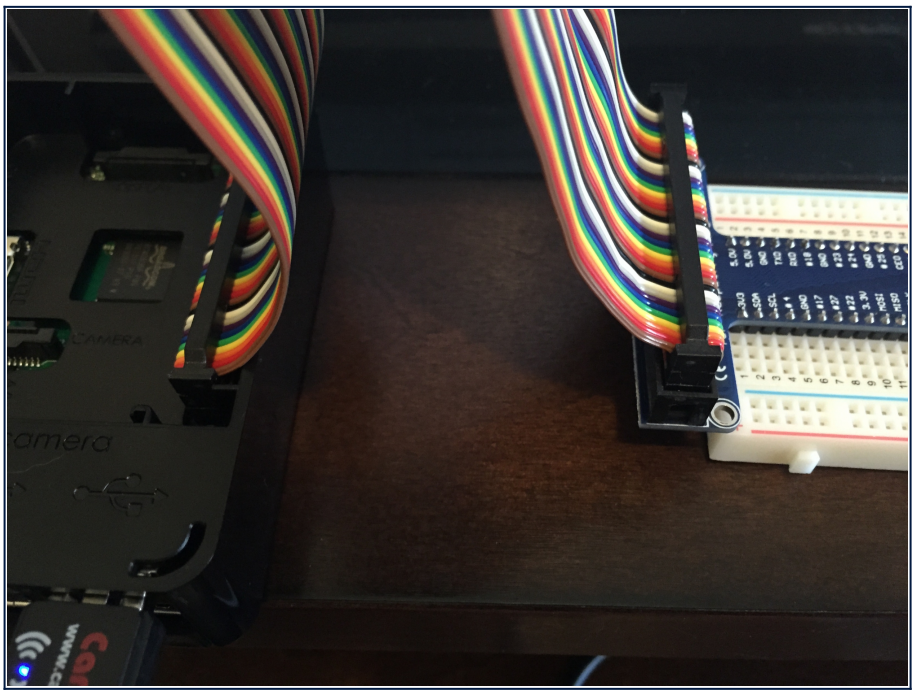
Setup the Breadboard

First, you'll want to plug your cobbler into the breadboard. The cobbler has to be in the center so that the left and right pins are separated by the center channel of the breadboard. In this picture, a "t-cobbler" is used and it is plugged near the top of the board to maximize available space on the board for wiring.



Carefully make sure all the pins are lined up with the holes on the breadboard and push it down. It may take a bit of force, but you want to get the cobbler to be flush with the breadboard.

The last step is to connect the cobbler to the Raspberry Pi. Plug the other end of the ribbon cable into the cobbler. It's likely the cobbler will have a slot in it so that the ribbon cable only fits in one direction.



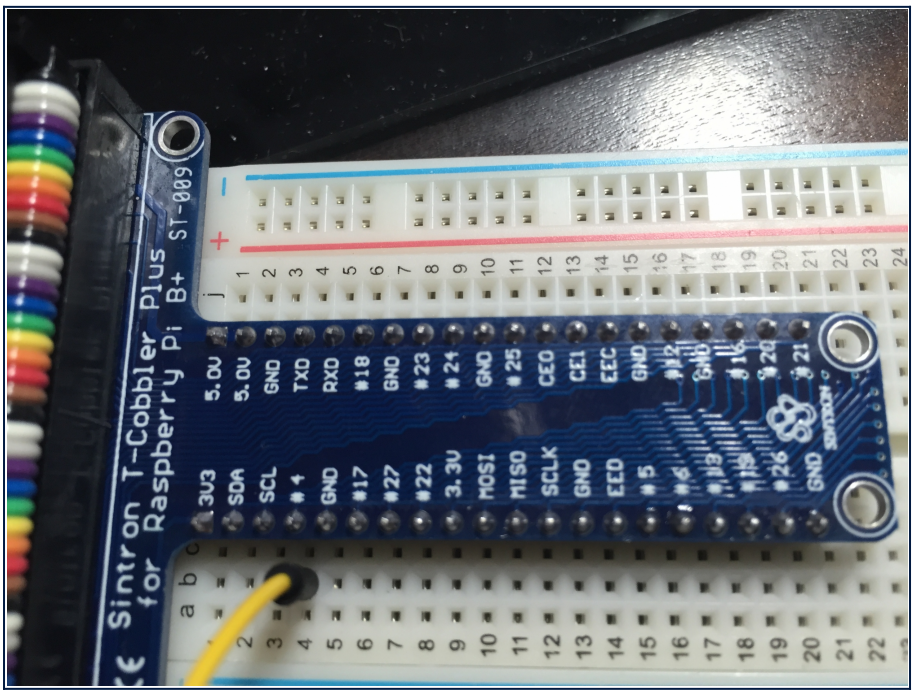
With the breadboard connected, you now have access to the pins on the GPIO port. As you can see in the photo above, this cobbler has each of the pins labelled so you can easily tell what they are for without having to count them.

Each row on the breadboard that matches up to a pin on the cobbler is "connected" to the pin on the cobbler. Any wires you connect on that row will act as if they are connected to pin on the GPIO port. For example, looking at the cobbler, you can see that the pin marked "#17" is aligned with row 6 on the cobbler.

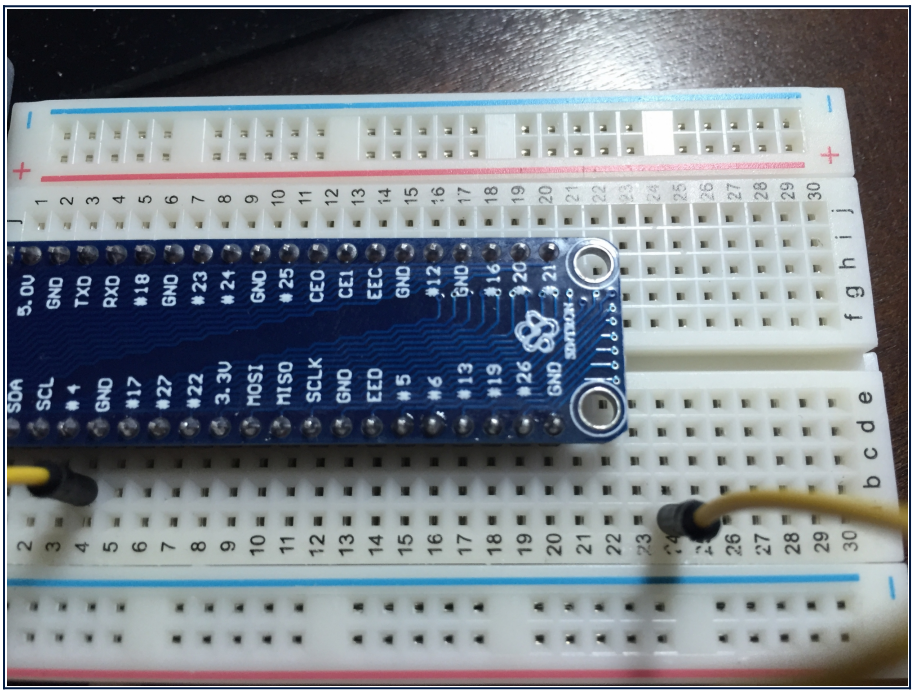
Wire the Circuit

Now that you have the breadboard hooked up to the Raspberry Pi, you can start on building the blinking LED circuit. In this step you'll use 3 wires, 1 LED and a resistor.

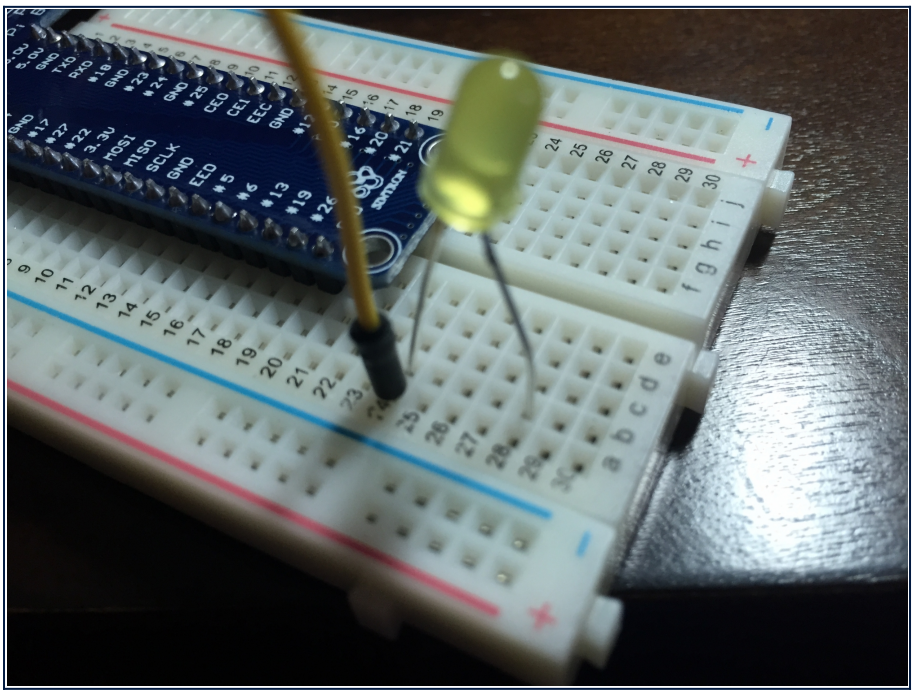
1. Connect a wire to the pin marked "#4". Yellow is used in this example.



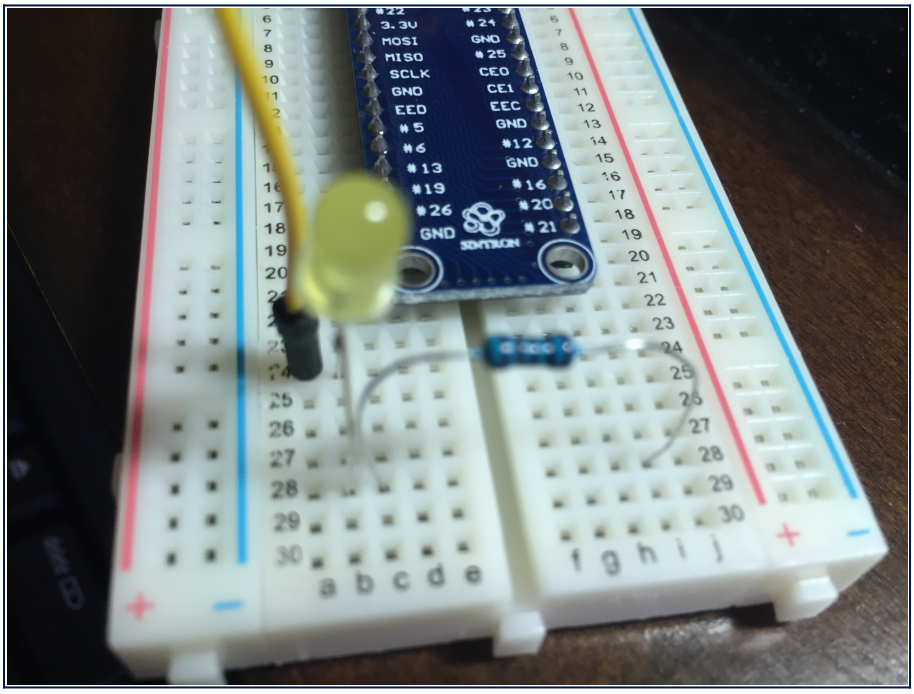
2. Connect the other end of the wire to an open spot on the board away from the cobbler. Here row 24 is used.



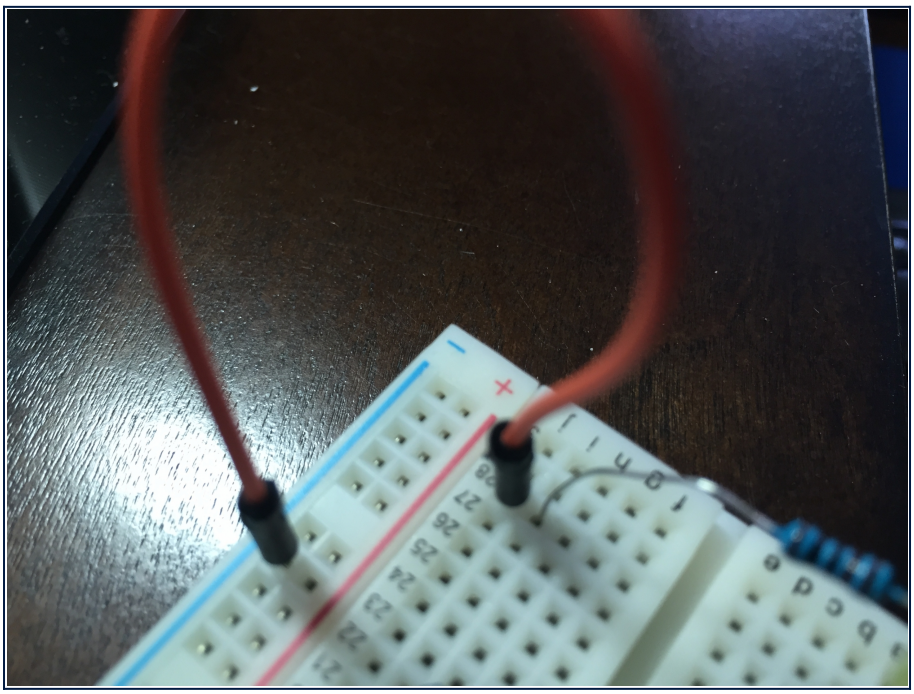
3. Grab the LED. Note that one of the wires coming from the LED is longer than the other. The longer end is the positive (+) connector. Plug the long end into a hole on the breadboard adjacent to the wire (yellow) and the short end to an open row on the breadboard.



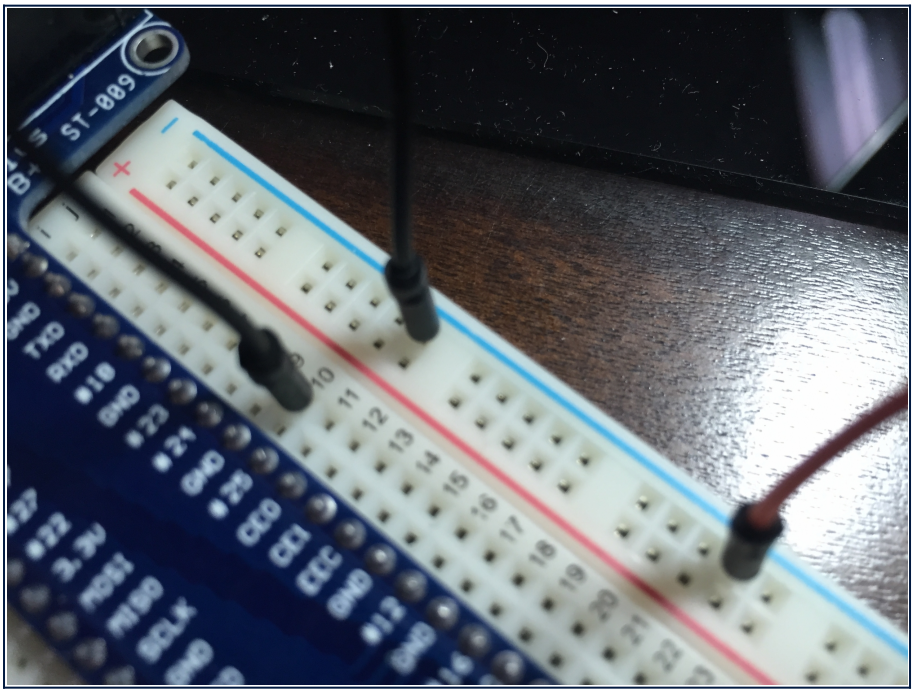
4. A resistor is needed to prevent the power provided by the Pi from blowing out the LED. Take the 10k resistor and connect one end so that it is adjacent to the negative wire of the LED. Connect the other end of the resistor to an open spot on the board. If you cross over the center of the board to the other side, you can take advantage of the same row, which is what is done here.



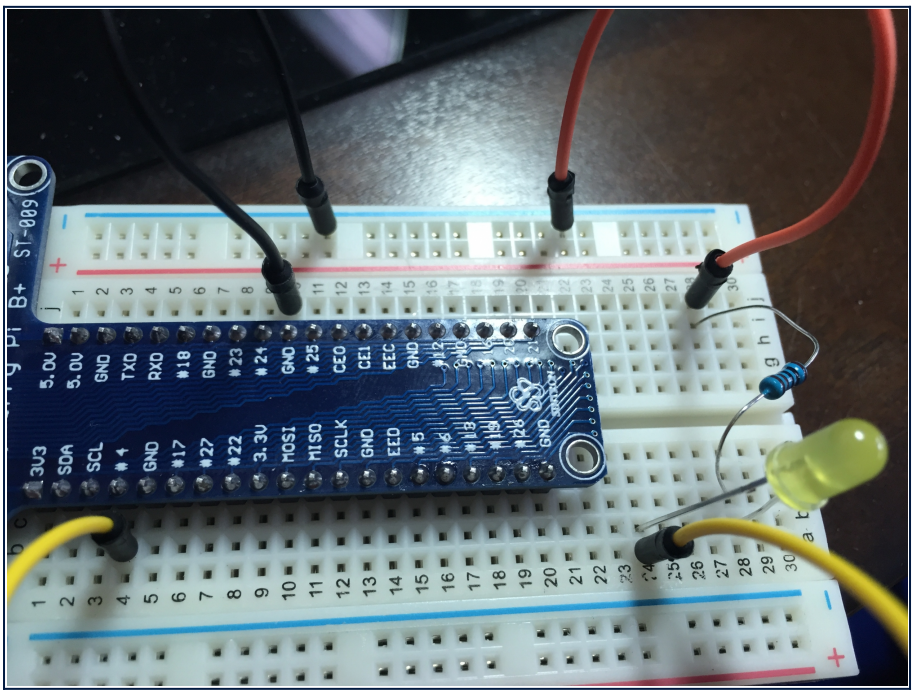
5. Now connect another wire adjacent to the unconnected resistor wire. Connect this wire to the negative (-) column on the breadboard. This is usually the column on the outer edge.



6. Lastly, connect a wire from the negative column of the breadboard to a GND pin on GPIO. There are several GND pins; you can use any one that is easily reachable.



7. This completes the circuit.



You can quickly test the circuit by moving the yellow wire from pin #4 to pin 3v3. This directly connects the circuit to 3.3 volt power. The LED should immediately light up. Switch it back to the #4 pin before moving on to creating the Xojo app.

Create the Xojo app

To test your circuit, you'll use a simple Xojo app that will alternate between ON (HIGH) and OFF (LOW) on pin #4. When the pin is ON (HIGH), the LED will illuminate.

1. Create a new Console project and call it LEDBlinker.
2. Add the GPIO module to the project.
3. Add this code to the App Run event handler:

```
GPIO.SetupGPIO

Const kLEDPin = 4 ' "#4" on the pinout

' Set the pin to accept output
GPIO.PinMode(kLEDPin, GPIO.OUTPUT)

' Blink LED every 1/2 second
While True
  ' Turn the pin on (give it power)
  GPIO.DigitalWrite(kLEDPin, GPIO.ON)
  App.DoEvents(500)

  ' Turn the pin off (no power)
```

```
  GPIO.DigitalWrite(kLEDPin, GPIO.OFF)
  App.DoEvents(500)
Wend
```

Build the app.

⚠ Before running the app, make sure you've selected **Linux** in Build Settings and its architecture to **ARM 32-bit** in the Inspector.

Transfer and Run the Xojo app

You can use the Remote Debugger to run and test this app on the Pi. Since this is a console app, you'll need to manually start the app on the Pi Terminal once it has been transferred by Xojo. To do this, navigate to the DebugLEDBlinker folder in the Remote Debugger Stub Download Location and then start it using this command:

```
sudo ./LEDBlinker
```

⚠ The sudo command is needed to access GPIO. The GPIO.SetupGPIOSys method does provide access to some GPIO functionality without requiring sudo.

The LED should start blinking. To stop the app, press Control-C.

Note that if you press Control-C while the LED is on, it will remain on.

Starting with Raspbian Jessie, you no longer have to use sudo to access GPIO. In order to not require sudo, you have to first set an environment variable in Terminal before you run the app:

```
export WIRINGPI_GPIOMEM=1
```

The Xojo project is included with the Xojo examples and is located here:

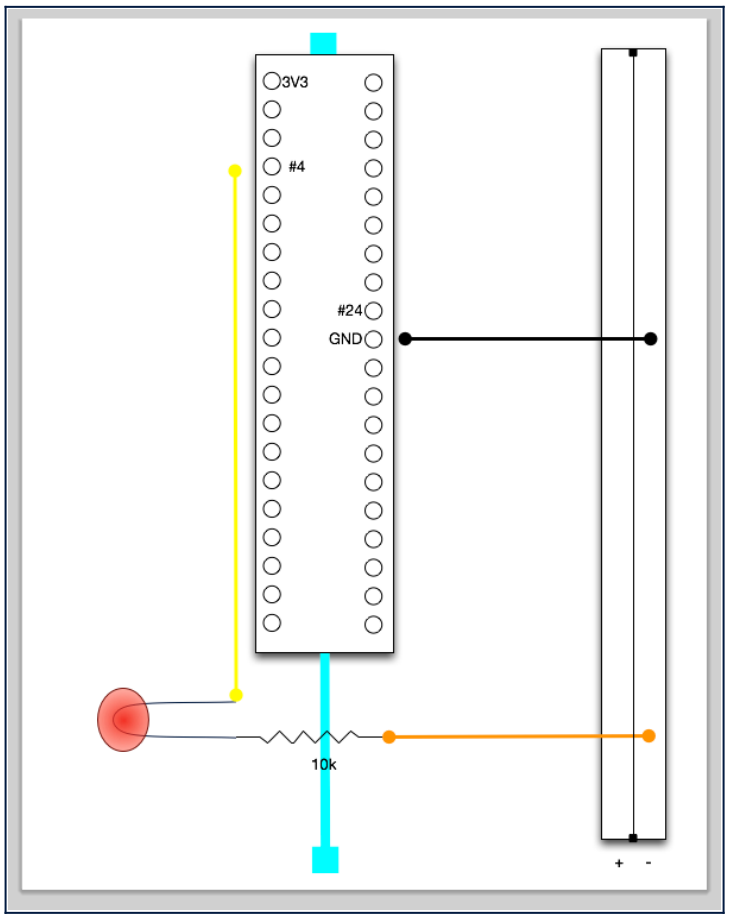
- Examples/Platform-Specific/Linux/RaspberryPi

Video

A video of the above steps is available here:

Circuit Diagram

Here is a simple circuit diagram that describes the wiring for the blinking LED circuit:


3V3
#4
#24
GND
10k
+
-

GPIO Project - Digital Clock

In this project you will learn how to hook up a HD44780-based LCD character display. This tutorial uses a 20 character by 4 line display, but most of these types of displays are hooked up the same and will work with the provided code. You'll then create a Xojo app to show the time and date on the LCD.

Parts

- LCD character display (HD44780, 20x4 lines)
- Potentiometer
- 16 jumper wires

Wire the Circuit

The LCD consists of 16 pins, of which 12 are used. The potentiometer is used to control the display's contrast. These are the overall steps:

1. Plug the LCD into your breadboard. Make sure you have plenty of space to work with. The pins must not overlap with any of the pins in the GPIO cobbler connector.
2. Wire up power and ground. The LCD uses 5v power. There are several power and ground connections so you'll want to use the power and ground rail on the side of the breadboard.
3. Connect the potentiometer. You can plug this in anywhere on the board with space.
4. Wire the LCD to the GPIO pins. The specific pins you use does not matter, but if you use different pins than described here, you'll need to update the code to use the pins you connected.
5. Wire the LCD to the potentiometer.

With this all hooked up, you should see the LCD light up and ought to be able to turn the potentiometer to adjust the contrast so that you can see the boxes where the characters are displayed. The LCD pins are numbered left to right when looking at the LCD with the pins at the top. These are the specific pin connections:

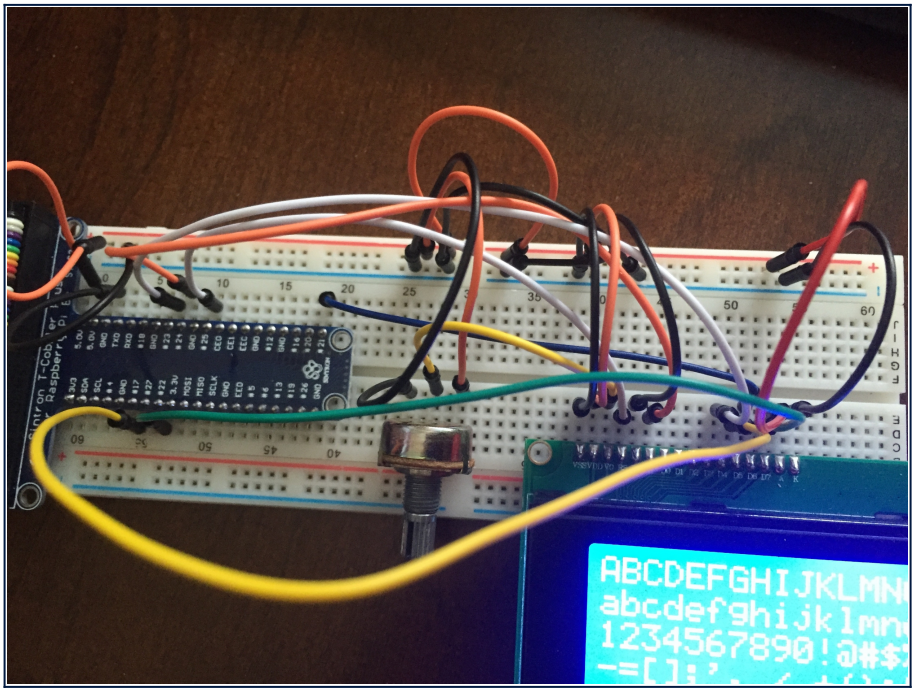
LCD Pin	GPIO Pin
#1 (GND)	GND
#2 (VCC)	5.0v
#3 (Vo)	To potentiometer (see below)
#4 (RS)	#25
#5 (RW)	GND
#6 (EN)	#24
#7, #8, #9, #10	not used

LCD Pin	GPIO Pin
#11 (D4)	#23
#12 (D5)	#17
#13 (D6)	#21
#14 (D7)	#22
#15 (LED+)	5.0v
#16 (LED-)	GND

A potentiometer is used to adjust the contrast of the LCD. It is connected like this:

Pot Pin	LCD Pin / GPIO Pin
#1	GPIO GND
#2	LCD #3 (Vo)
#3	GPIO 5.0v

When everything is connected correctly you will see the LCD light up and you'll be able to adjust the contrast with the potentiometer. You will not see any characters, although you ought to be able to see the individual character boxes.



Create the Xojo App

You'll use the Xojo GPIO library to display text on the LCD. This library has the GPIO.LCD module which contains methods you can call to easily intialize and display text on the LCD.

A Conole app witht his code in the Run event handler displays text on the LCD:

```
' Display text on 4-line LCD
GPIO.SetupGPIO

Const kRSPin = 25
Const kEPin = 24
Const kD4Pin = 23
Const kD5Pin = 17
Const kD6Pin = 21
Const kD7Pin = 22

Dim lcd As New GPIO.LCD(kRSPin, kEPin, kD4Pin, kD5Pin, _
  kD6Pin, kD7Pin)

lcd.Clear
lcd.Home
lcd.SetMessage("ABCDEFGHIJKLMNOPQRSTUVWXYZ", 1)
lcd.SetMessage("abcdefghijklmnopqrstuvwxyz", 2)
lcd.SetMessage("1234567890!@#$%^&*()", 3)
lcd.SetMessage("-=[];',./_+{}:""<>?", 4)
```

Using the code above as a starting point, you can now create a Xojo app that updates the LCD to display the current

time and date. This code in the Run event of a Console app updates the time and date about every 1/2 second:

```
' Display time and date on 4-line LCD
Using Xojo.Core

GPIO.SetupGPIO

Const kRSPin = 25
Const kEPin = 24
Const kD4Pin = 23
Const kD5Pin = 17
Const kD6Pin = 21
Const kD7Pin = 22

Dim lcd As New GPIO.LCD(kRSPin, kEPin, kD4Pin, kD5Pin, _
  kD6Pin, kD7Pin)

While True ' Loop forever
  lcd.Clear
  lcd.Home
  Dim d As Date = Date.Now
  Dim currentTime As Text = d.ToText(Locale.Current, Date.FormatStyles.None,
Date.FormatStyles.Long)
  Dim currentDate As Text = d.ToText(Locale.Current, Date.FormatStyles.Long,
Date.FormatStyles.None)

  lcd.SetMessage(currentTime, 1)
  lcd.SetMessage(currentDate, 2)

  App.DoEvents(500) ' sleep about 1/2 second to reduce CPU usage
Wend
```

You can run this on the Pi using the Remote Debugger or by building and transferring the app.

What's Next

You've now created a few types of apps for your Raspberry Pi and even made a few hardware projects. But this only scratches the surface. The Raspberry Pi really is the "ultimate gadget" and what you can do with it is limited only by your imagination.

You can download the source code for all the projects in this book here: Raspberry Pi Book Project Source

> You'll want to use Xojo 2017 Release 1 or later with these projects.

You can use Xojo for free to make and test your Raspberry Pi apps. When you are ready to build apps to install on the Pi (for yourself or others) you can purchase a Xojo license in the online store.

Here are some other projects and references that will help you on your journy of Raspberry Pi discovery:

- More Xojo projects on the Einhugur blog
- Program Raspberry Pi 2 B Electronics with Xojo Book by Eugene Dakin
- Official Raspberry Pi site
- MagPi, the official Raspberry Pi magazine
- Xojo Raspberry Pi forum channel
- Official Raspberry Pi forum
- CanaKit
- AdaFruit