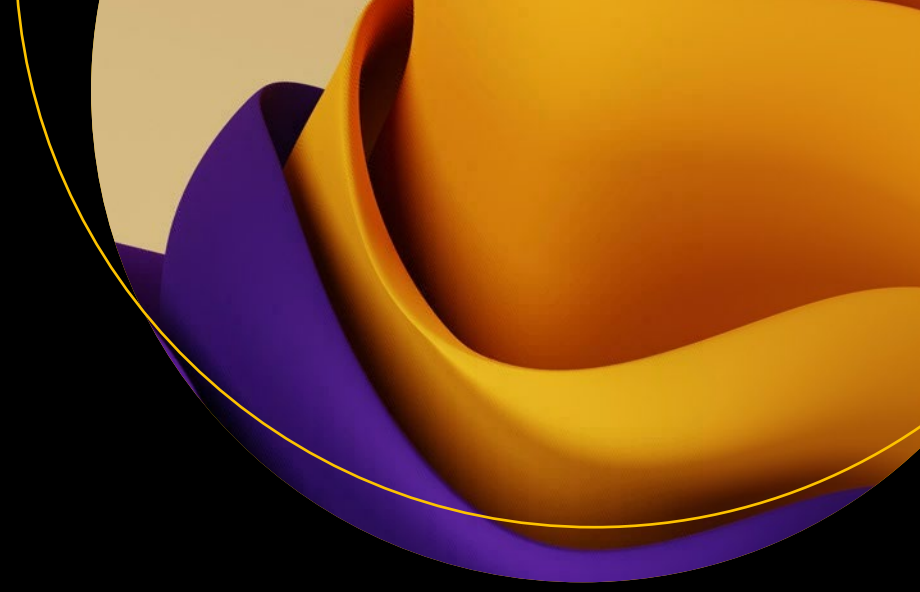


Creating Business Applications with Microsoft 365

Techniques in Power Apps, Power BI, SharePoint, and Power Automate

—

Second Edition

—

Jeffrey M. Rhodes

Apress®

Creating Business Applications with Microsoft 365

Techniques in Power Apps, Power BI, SharePoint, and Power Automate

Second Edition

Jeffrey M. Rhodes

Apress®

Creating Business Applications with Microsoft 365: Techniques in Power Apps, Power BI, SharePoint, and Power Automate

Jeffrey M. Rhodes
Colorado Springs, CO, USA

ISBN-13 (pbk): 978-1-4842-8822-1 ISBN-13 (electronic): 978-1-4842-8823-8
https://doi.org/10.1007/978-1-4842-8823-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Smriti Srivastava
Development Editor: Laura Berendson
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image by Philip Oroni on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

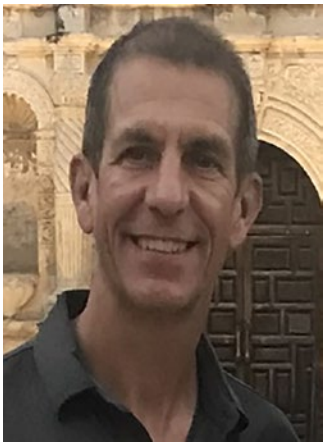
Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to my late parents, who steadfastly supported me my whole life: Brigadier General (ret.) James M. Rhodes, Jr., Catherine Harrington Rhodes (mother), and Sylvia Dunlevy Rhodes (stepmother).

Table of Contents

About the Author



Jeffrey Rhodes is a founder and Chief Technical Officer of Platte Canyon Multimedia Software Corporation, a leader in developing commercial e-learning software. He graduated at the top of his class at the Air Force Academy, where he earned a bachelor's degree in electrical engineering. Jeff received a master's degree in economics from the London School of Economics, which he attended under a British Marshall Scholarship. He is the author of *Creating Business Applications with Office 365: Techniques in SharePoint, PowerApps, Power BI, and More*; *Programming for e-Learning Developers: ToolBook, Flash, JavaScript, and Silverlight*; and *VBTrain.Net: Creating Computer and Web Based Training with Visual Basic .NET*. He also co-wrote *The ToolBook Companion*. He lives in Colorado Springs with his wife Sue and is the proud father of his sons Derek and Michael.

About the Technical Reviewers

Fabio Claudio Ferracchiati is a senior consultant and a senior analyst/developer using Microsoft technologies. He works for BluArancio (www.bluarancio.com). He is a Microsoft Certified Solution Developer for .NET, a Microsoft Certified Application Developer for .NET, a Microsoft Certified Professional, and a prolific author and technical reviewer. Over the past ten years, he's written articles for Italian and international magazines and coauthored more than ten books on a variety of computer topics.

Arun Sharma is a techno-strategic leader and carries deep experience in development consulting, cloud, AI, and the IoT space. He is currently associated with Fresh & Pure, an agritech startup, in the position of Director Program Management–Automation and AI. He has expertise in various technologies (Microsoft Azure, AWS, Ali Cloud), IoT, ML, microservices, bots, Dynamics 365, PowerPlatform, SAP Crystal Reports, DevOps, Docker, and containerization. He has more than 20 years of experience in a wide range of roles such as GM–Paytm, Delivery Manager at Microsoft, Product Manager at Icertis, Lead and Architect Associate at Infosys, Executive Trainer at Aptech, and Development Consultant at CMC. He has managed CXO-level relationships on strategic levels, sales, cloud consumption, consulting services, and adoption with medium- and large-size global customers.

Acknowledgments

As I finish this, my fifth book, my mind is on my awesome family. My mom, dad, and stepmother are unfortunately no longer with us. But I still think of them often. My older brother Jim inspires me every day with how he tackles numerous health challenges, while still keeping a positive outlook and DJ'ing a weekly music program. My sister Joni is the leader of our family and has a heart of gold. She hosts every big event and is always there when you need her. She and my brother-in-law Sean are our travel buddies as well. Figure 1 is one of my favorite family pictures from when we were young.

Figure 1. *My sister Joni, my dad, my brother Jim, my mom, and me*

I spent most of my teenage years with my stepmother, Sylvia, and we became very close. We lost her too early. Figure 2 is from my high school graduation, which was extra special because my dad handed out the diplomas as the Wing Commander of the Air Force Base, which is kind of like being the mayor when stationed overseas.

Figure 2. *My dad, myself, and my stepmother Sylvia at my high school graduation*

My wife Sue is my best friend and biggest supporter. I feel so fortunate that I met her at the Air Force Academy, and we've been happily married for over 34 years. Our sons Derek and Michael have made our lives infinitely richer. We are very proud of the honest and caring men they have become. We are equally proud of our daughter-in-law, Alexis. Her kindness and fun-loving nature are contagious.

Thank you all for the love and support. My life would be empty without you!

Introduction

This book is geared toward power users and what I call business developers. While some of the applications and techniques require a degree of understanding programming, my objective is to make the solutions accessible to the non-computer scientist.[1] The joy of writing an application or visualization that makes your and/or a co-worker's life better is something I hope you all can experience. Sometimes, the solutions can help hundreds or thousands of users, and other times, like the example later where I first used Access and then Power BI to help my wife detect errors in spreadsheets, just have a single user. Either way, I hope these techniques help you and your organization with these powerful tools.

Changes in This Edition

Well over half of this edition is brand new. With the huge expansion of Teams, SharePoint – in my view – has largely faded into the background. Its primary purpose now is a file storage location for Teams and as data storage via lists for Power Apps and Power Automate. In addition, Microsoft 365 is now most often deployed in the Azure cloud rather than being hosted on-premise. With that security model, it is much less common to allow custom scripting in SharePoint pages, not to mention that SharePoint pages themselves are less used compared to Teams posts, tabs in Teams channels, etc. So I have removed all the JavaScript and jQuery custom scripting examples. SharePoint Designer workflows and InfoPath forms are deprecated as well, so those are gone. Microsoft Flow has been renamed Power Automate, and I've greatly expanded its coverage and added numerous new examples of how to effectively use Power Apps and Power BI. I've also added Teams content, such as sending chats/posts and scheduling Teams Meetings via Power Automate. I've thoroughly updated the remaining content as well.

[1] Which I am as well. My undergraduate degree is in electrical engineering while my master's is in economics. While I took some Pascal and FORTRAN in college, I discovered my love of programming using a language called *OpenScript* in a now-defunct e-Learning authoring environment called *ToolBook*.

Audience Level

While some programming expertise will be helpful in understanding the code in Power Apps, the actions in Power Automate, and the custom columns in Power BI, I do not assume that you are a programmer. I start the book with a new "*Programming in the Power Platform"* chapter that introduces these tools and compares them to more traditional programming in .NET and HTML/JavaScript. Anyone who is willing to learn and feels at home in front of a keyboard can truly benefit.

CHAPTER 1

Programming in the Power Platform

The Power Platform is a great environment but does things a bit differently than traditional development tools. This chapter[1] illustrates this as we take on the challenge of a simple form to compose and send an email in .NET, HTML/JavaScript, Power Apps on its own, and Power Automate and Power Apps in combination. That will allow us to examine core programming concepts and explain how the Power Platform implements them, often in ways different than those of you familiar with other environments might expect. If you are not familiar with these other environments, however, feel free to jump ahead to the Power Apps section.

Core Programming Concepts

The basic idea of programming is that it is made of objects that have Properties, Methods, and Events. *Properties* are things that an object *has*. These are attributes like height, width, color, visible, and text. *Methods* are things that an object can *do*. They are actions or capabilities available in the object. A *media player* object, for example, would have events like play, rewind, pause, fast forward, and stop. *Events* are actions recognized by an object that your code can respond to. A button object has a *Click* event. A *media player* object fires an event when the media it is playing reaches its end, allowing you to jump to the next item in the playlist or prompt the user for what to do next.

[1] I wrote a whole book years ago with an e-Learning programming challenge implemented in ToolBook, Flash, JavaScript, and Silverlight. All of these except JavaScript are now defunct.

J. M. Rhodes, *Creating Business Applications with Microsoft 365*,
https://doi.org/10.1007/978-1-4842-8823-8_1

In most of our examples, we will have *Textbox* objects that contain the *To* email address and *Subject* of the email and a *RichTextBox* object that contains the *Body* of the email. These values will be *Properties* of their corresponding objects. We will then handle the *Click* event of the *Send Email* button and write code to set the properties of the appropriate *Mail* object, finally calling a *Send Email* method of that object to actually send our email. When we get to Power Apps, we will see that the event is called *OnSelect* instead of *Click*. Power Automate doesn't actually have an interface, so we will use a modified version of our Power Apps example to let Power Automate do the heavy lifting.

.NET

We will start with a standard .NET Windows Forms application using Visual Basic[2] with Visual Studio as a development environment. Figure 1-1 shows Visual Studio with the *Toolbox* on the left for dragging on controls and the design surface in the middle (with separate design and code files as discussed shortly). The *Solution Explorer* at the upper right shows all the files, while the *Properties* window below it gives an interface for setting properties for whatever object is selected. In this case, I've selected the *Send Email* button and am setting its *BackColor* property.

[2] I like Windows Forms for this purpose as opposed to *Windows Presentation Foundation* since its relationship between methods, properties, and events is more straightforward. As for Visual Basic, its syntax is closer to the syntax in the Power Platform. Plus, I am an old-time VB rather than C# guy.

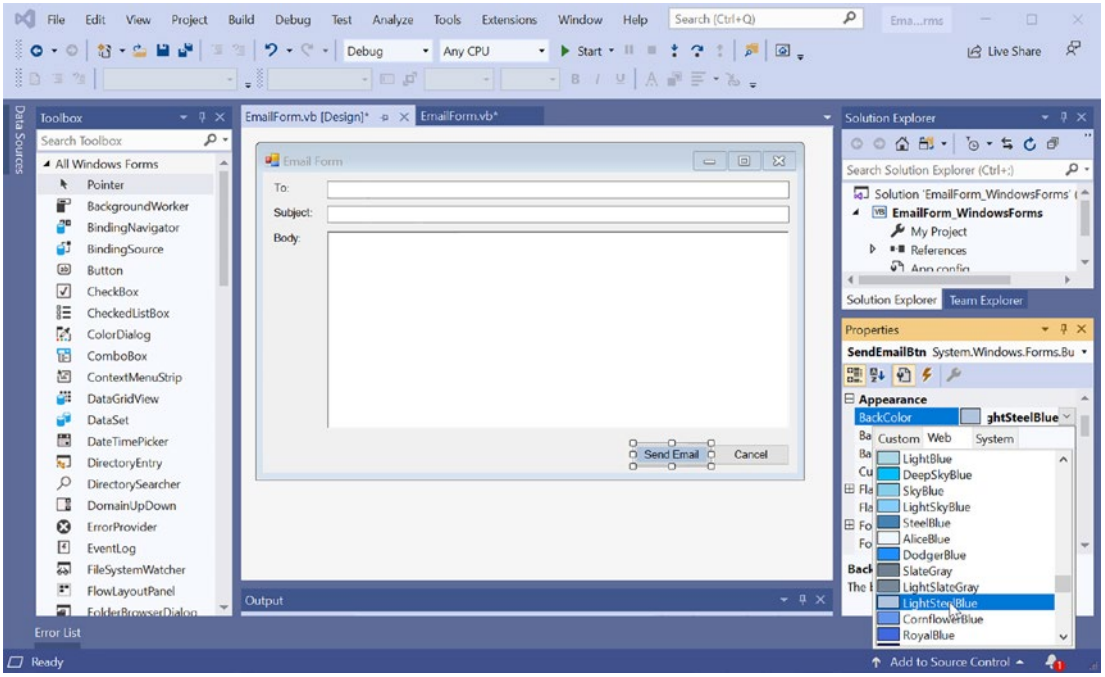


Figure 1-1. *Windows Forms Application in Visual Studio Showing the Setting of a Property*

The code is contained in the EmailForm.vb file and is shown in Listing 1-1.

Listing 1-1. Sending an Email in Windows Forms and Visual Basic.Net

```
Imports System.Net.Mail

Public Class EmailForm
    Private Sub SendEmailBtn_Click(sender As Object, e As EventArgs)
Handles SendEmailBtn.Click
        Dim mailId As New MailMessage()
        Dim clientId As New SmtpClient("localhost")

        With mailId
            .From = New MailAddress("info@plattecanyon.com")
            .To.Add(New MailAddress(ToBox.Text))
            .Subject = SubjectBox.Text
            .Body = BodyBox.Text
```

```
        End With

        Try
            clientId.Send(mailId)
            Message.Text = "Your email has been sent."
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub

    Private Sub CancelBtn_Click(sender As Object, e As EventArgs) Handles
CancelBtn.Click
        Me.Close()
    End Sub
End Class
```

The *Imports System.Net.Mail* line tells the code that we want to use functions within that *Namespace*. It allows us to use a shorter *MailMessage()* syntax rather than repeat the whole *System.Net.Mail.MailMessage()* part every time. All the action happens in our *Click* handler. Visual Studio automatically names our subroutine *SendEmailBtn_Click*. The important part is the *Handles SendEmailBtn.Click*. This means that this code gets called in response to the *Click* event for the button. We then create two objects: *MailMessage* and *SmtpClient*. The latter is what sends the actual email. For more elaborate implementations, we would pass it the name of the mail server and some credentials, but here, we just tell it to use our local web server (*localhost*). With our mail message, we then set its *From*, *To*, *Subject*, and *Body* properties. Notice how we read from the *ToBox*, *SubjectBox*, and *BodyBox* objects on the form and specifically read their *Text* properties. Notice also how naming the objects (as opposed to leaving them TextBox1, TextBox2, etc.) makes our code much more readable and easier to maintain. This will carry over to Power Apps and Power Automate as well. The *Try - Catch* syntax is used when we think our code may generate an error. Since sending an email can easily generate an error, this is appropriate. It allows us to display a message to the user instead of just crashing upon any failure. We call the *Send* method and pass our *mailId* object as a parameter. If this succeeds, we make it to the next line and we set the *Text* property

of our *Message* label to "Your email has been sent."[3] If *Send* fails, the *Catch* line give us an *Exception* object. We display the *Message* property of that object. If the user clicks the *Cancel* button, we call the *Close* method of the form.

HTML/JavaScript

Our next example won't actually send the email since that can't be done directly from JavaScript,[4] Instead, we will launch our default email program, as shown in Figure 1-2.

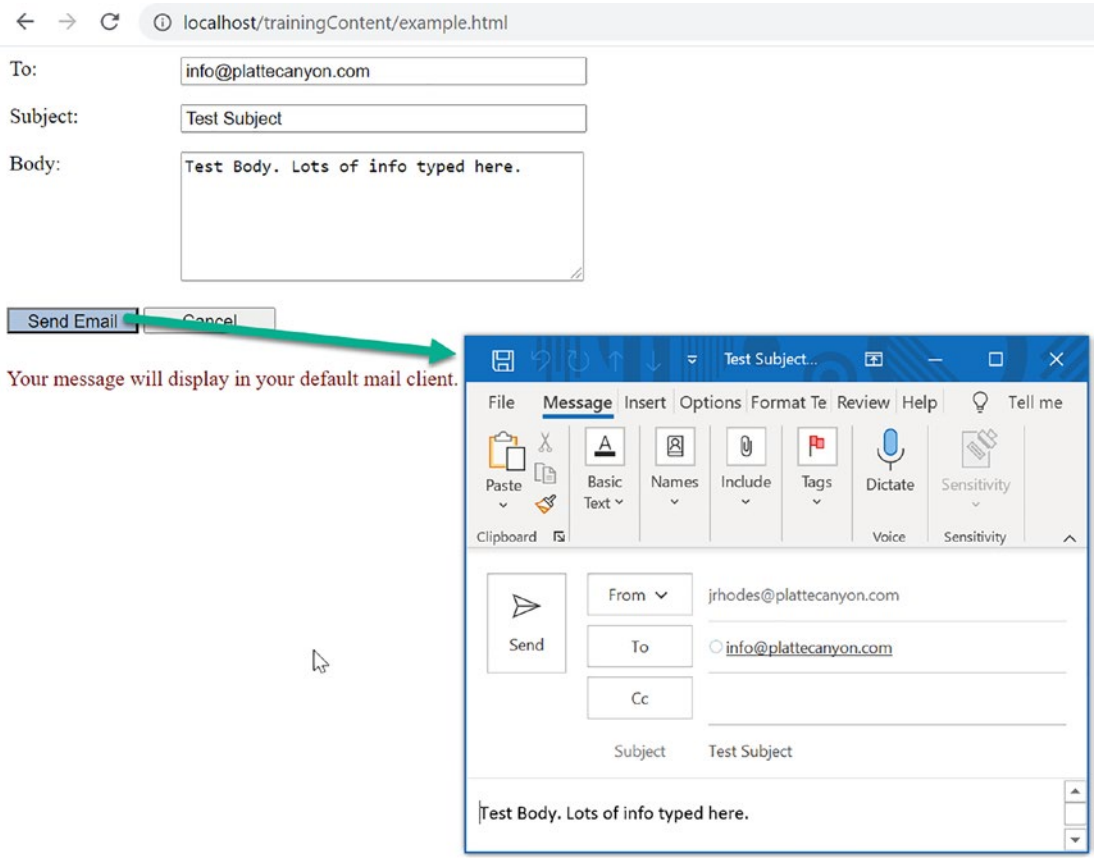


Figure 1-2. *Generating an email from an HTML and JavaScript page*

[3] Note how we can set the *Text* of the label directly. We will see that when we get to Power Apps, this is not possible in that environment. We will use variables instead.

[4] We could call code like Listing 1-1 from JavaScript or by "posting" to a web server if desired.

The gist of the solution is that we build a link like this: *mailto:info@plattecanyon.com?subject=Test Subject&body=Test Body. Lots of info typed here.* We then launch the link to bring up the default mail client like Outlook. Listing 1-2 shows the implementation.

Listing 1-2. HTML, styles, and JavaScript for generating an email

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Send Email</title>
    <style>
        label {
            width: 100px;
            position: absolute;
            left: 10px;
        }

        input {
            width: 300px;
            position: absolute;
            left: 140px;
        }

        textarea {
            width: 300px;
            position: absolute;
            left: 140px;
        }

        button {
            width: 100px;
            height: 20px;
            position: relative;
            top: 100px;
        }
```

```
        span{
            position: relative;
            top: 125px;
            color: maroon;
        }
        #SendEmailBtn {
            background-color: lightsteelblue;
        }
    </style>
    <script>
        function SendEmail() {
            var toValue = document.getElementById("ToBox").value;
            var subjectValue = document.getElementById("SubjectBox").value;
            var bodyValue = document.getElementById("BodyBox").value;
            var mailString = "mailto:" + toValue + "?subject=" +
subjectValue + "&body=" + bodyValue;
            window.open(mailString, "emailWindow");
            document.getElementById("message").innerHTML = "Your message
will display in your default mail client."
        }
        function CloseWindow() {
            window.close();
        }
    </script>
</head>
<body>
    <label>To:</label><input id="ToBox" type="text" />
    <br />
    <br />
    <label>Subject:</label><input id="SubjectBox" type="text" />
    <br />
    <br />
    <label>Body:</label><textarea id="BodyBox" rows="6" cols="20">
</textarea>
    <br />
```

```
    <button id="SendEmailBtn" type="button" onclick="SendEmail()">Send
Email</button> <button id="CancelBtn" type="button"
onclick="CloseWindow()">Cancel</button>
    <br />
    <span id="message"></span>
</body>
</html>
```

Let's start with the HTML at the bottom. HTML *input* objects are for a single line, so we use that for the *To* and *Subject* information. For the *Body*, we use a *textarea* object. As you might expect, we use *button* objects to send the email or cancel. We use a *span* to display our message after the user clicks the button. To get all the objects to line up correctly, we use the styles near the top of the listing.[5] When we list just the name of the object as in *input*, that applies to all the objects of that type on the page. Note the *width*, *position*, *left*, *top*, etc. are all *properties* of the corresponding objects. When we include the # sign as in *#SendEmailBtn*, that limits it to just an object with that *id*. So the *lightsteelblue* background color only applies to the *Send Email* button.

Now let's talk code. We handle the *onclick* event by listing the function we will call in response: *onclick="SendEmail()"*. We look in the script section[6] for the corresponding code. In it, we build variables to hold info we need: *toValue*, *subjectValue*, and *bodyValue*. This introduces us to a common challenge in programming (and one we will tackle later in a later chapter in Power Apps): how to get from the *name* of an object to the *reference* for the object. To read the *value* property of the *ToBox* object, we need to get from the string "ToBox" to the object *ToBox*. That is the purpose of the *document.getElementById("ToBox")* syntax.[7] Once we have our three variables, we concatenate them into our *mailString* variable. We then call the *open* method of the *window* object to launch our mail client. For our cancel functionality, we call its *close* method instead.

[5] We would normally put the styles in a Cascading Style Sheet (CSS) file that we could share among multiple pages.

[6] Like the CSS file, we would typically put the JavaScript in a separate .js file.

[7] The recurring need for "selectors" to go from *id*, *class*, or other information to object references is one reason for the popularity of the jQuery JavaScript library. For example, our code would be *$("#ToBox").value* if we were using jQuery. I covered jQuery in depth in the previous edition of this book in the context of adding functionality to SharePoint pages with JavaScript. Since that is much less common in SharePoint Online, I have removed those examples from this edition.

To display our message, we get the object reference to our *span* object and then set its *innerHTML* property to be the text we want.

Power Apps

Now that we've seen a couple of standard environments, let's tackle this task in Power Apps. We start by selecting a Canvas app from blank, as shown in Figure 1-3.

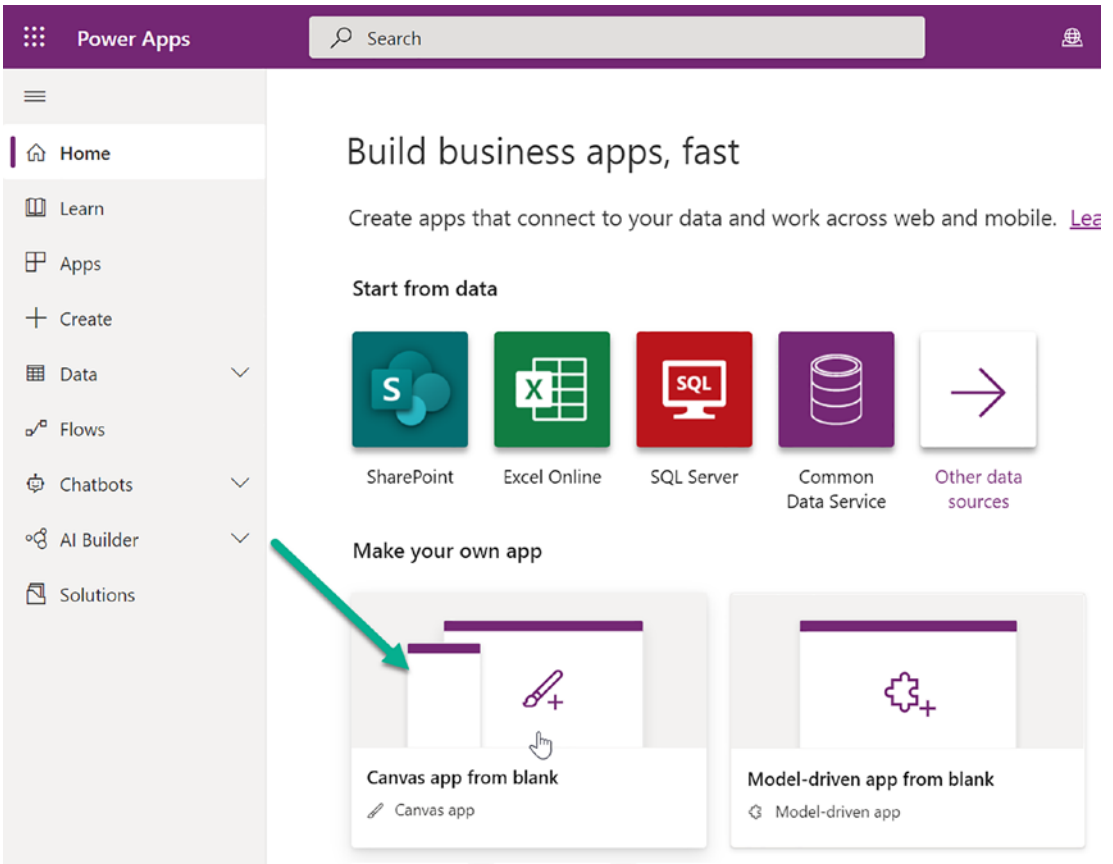


Figure 1-3. *Building a Canvas app from blank*

We choose Tablet as our format and then start adding objects. We use *labels* and *text input* controls in a similar way to our previous examples. To allow a formatted body, we use a *Rich text editor*, as shown in Figure 1-4. We choose it instead of the *HTML text control* since the latter only allows the display but not the entry of text. We continue by adding two buttons and arranging our objects on the screen.

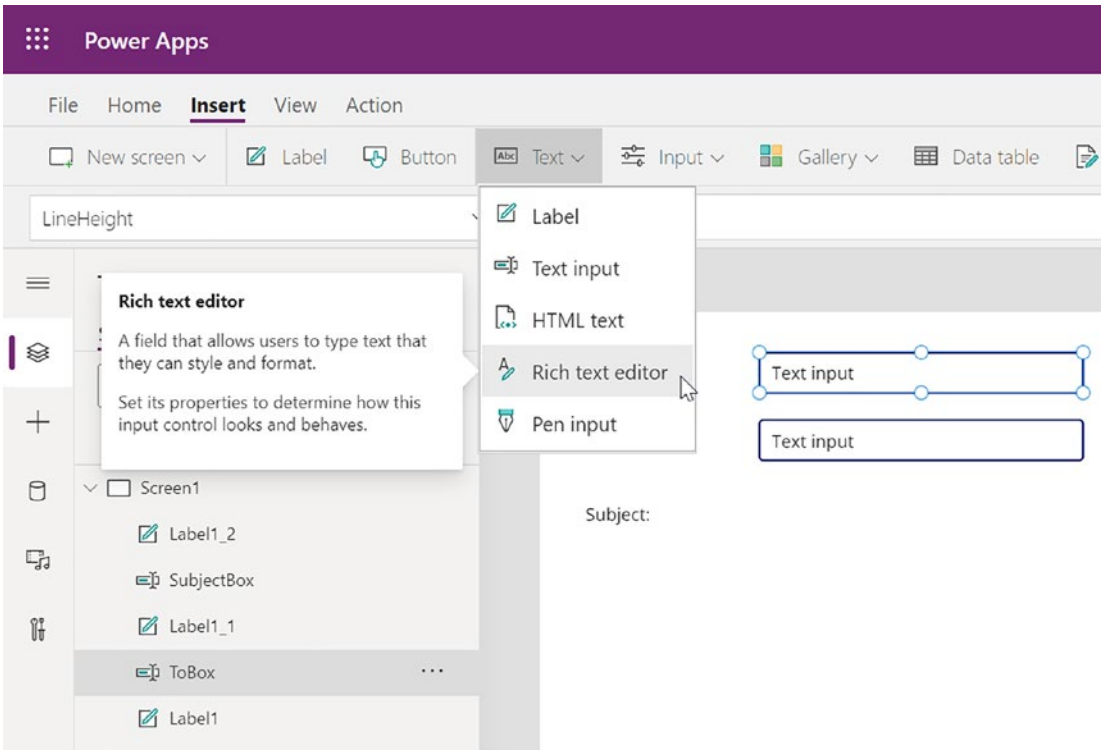


Figure 1-4. *Adding a Rich text editor in Power Apps*

One of the most interesting and powerful aspects of Power Apps is how the entire development is accomplished in the browser, as opposed to a specialized program like Visual Studio. As we see in Figure 1-5, the drop-down list of choices when we select an object is a mix of *properties* we can set and *events* we can handle. In addition, there is typically a property sheet on the right-side of the screen where we can also set properties. In Figure 1-5, we can set the *Text* of the button in either location.

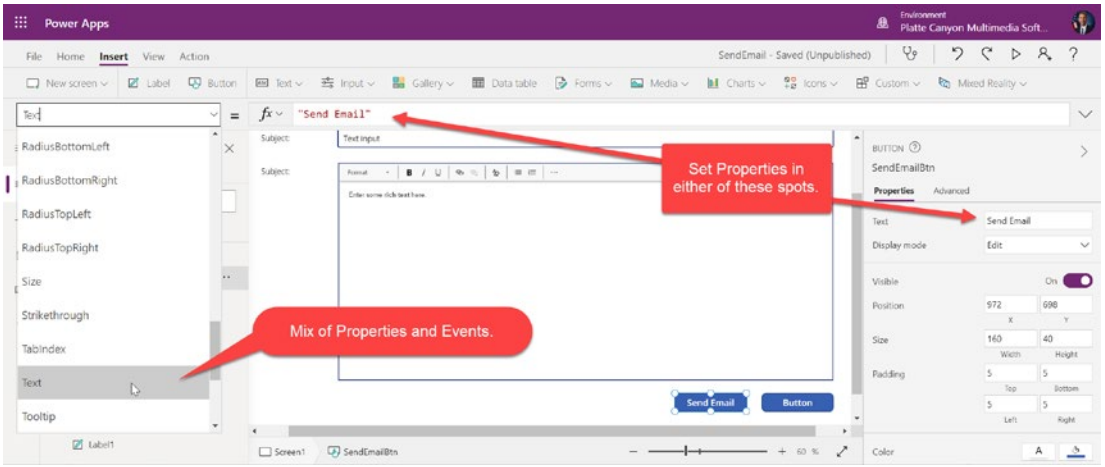


Figure 1-5. *Setting a property in Power Apps*

As we lay out the rest of the screen, we see that Power Apps controls have a *Default* property, as shown in Figure 1-6. This is what shows up when the user first enters the screen. If it is blank, then the *HintText* property displays to give an indication to the user as to what to do with that control.

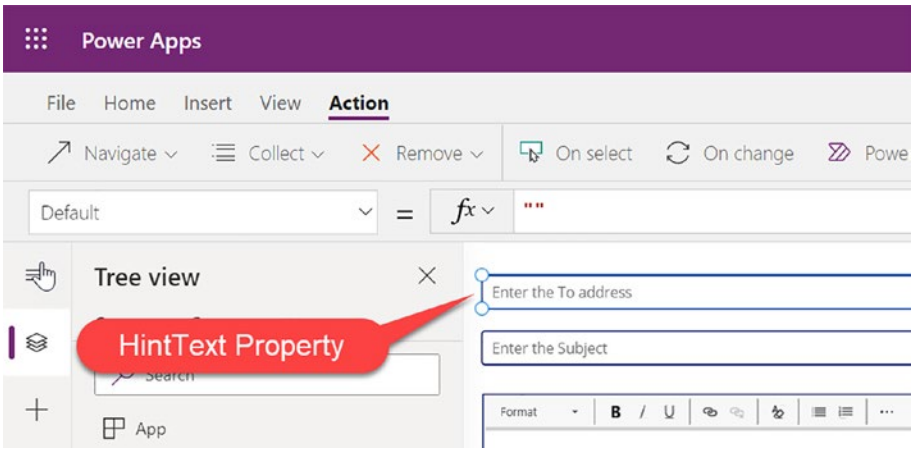


Figure 1-6. *Default and HintText properties of a Text Input control*

Our next task is to add the Office 365 Outlook connector so we can send email. Figure 1-7 shows us the steps: go to *Data*, *Add data*, and then search for *Outlook*.

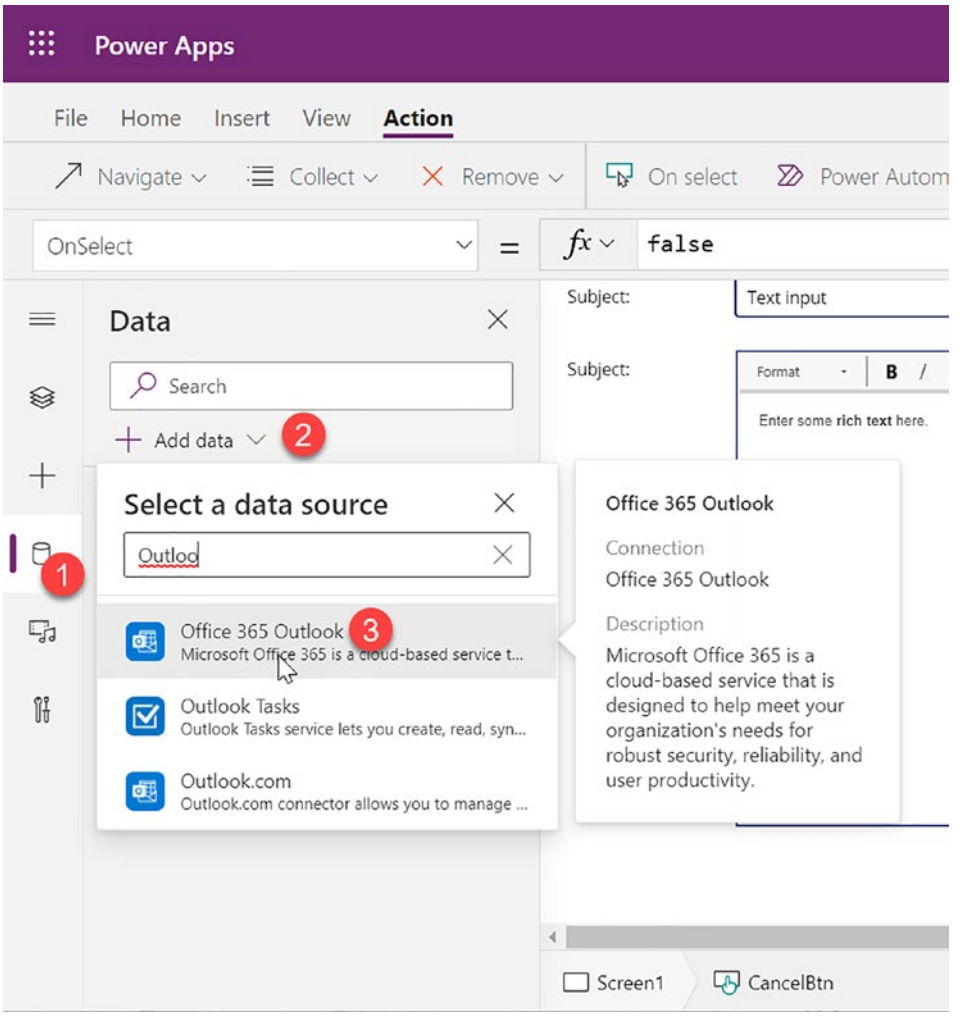


Figure 1-7. *Adding the Office 365 Outlook Connector to Power Apps*

We are now ready to implement the rest of the solution. We select our *SendEmailBtn* object and choose its *OnSelect* event (which is analogous to *Click* and *onclick* that we saw in our previous examples). Figure 1-8 shows how Power Apps gives us both the parameters of the *SendEmailV2* method but also Intellisense where it displays a list of the available properties and methods upon typing the "." after the object name. Notice how we enter the code directly in the entry box (rather than in another file) so that code goes side-by-side with property values. You can drag down the box to make it bigger to see all your code.

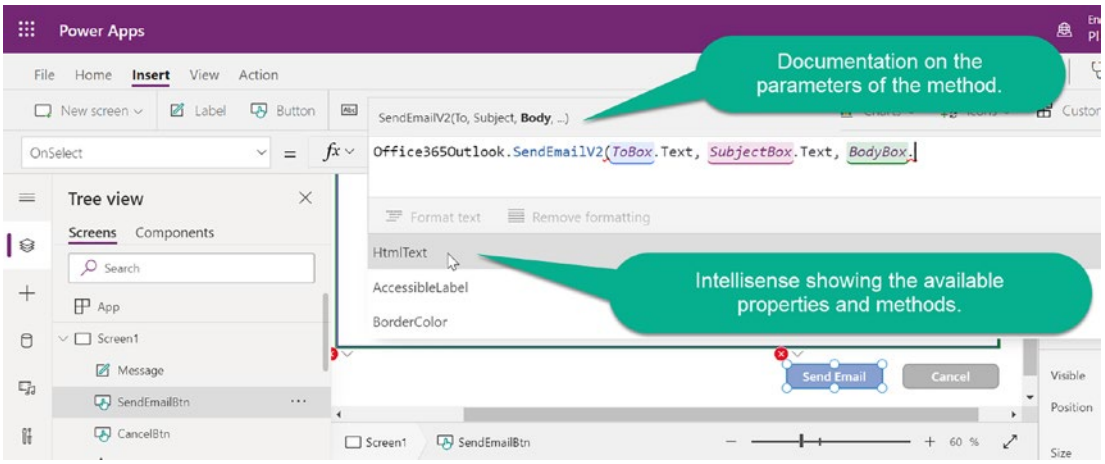


Figure 1-8. *Method documentation and Intellisense inside Power Apps*

Let's look at the message we want to send after the email is sent. Unlike .NET and JavaScript, we can't directly set a property of one object from another. Instead, we set the *Text* property to be a variable, as shown in Figure 1-9.

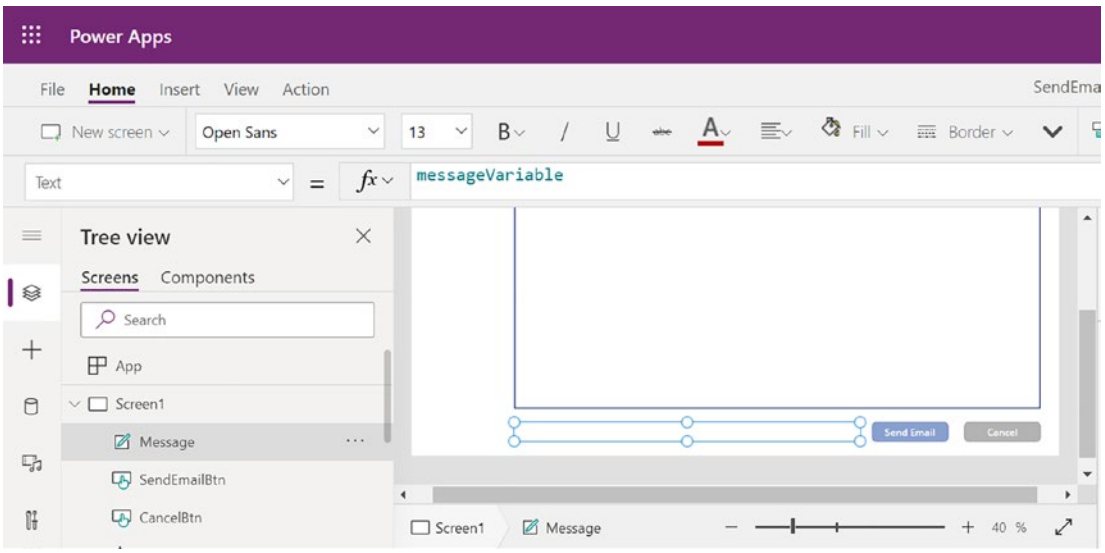


Figure 1-9. *Setting the Text property to a variable in Power Apps*

We then can change the value of that variable when the user clicks the *Send Email* button. Here is the entire *OnSelect* code:

```
Office365Outlook.SendEmailV2(
    ToBox.Text,
    SubjectBox.Text,
    BodyBox.HtmlText
);
Set(
    messageVariable,
    "Your email has been sent."
)
```

Notice how the ability to use the *Office365Outlook* connector makes it easy to send email. Our .NET example needed a mail server and associated credentials, while sending email was not even possible with our HTML and JavaScript solution. With PowerApps, it was just one line of code. We then use the *Set* command[8] to give our *messageVariable* the value of "Your email has been sent." This makes that text display, as shown in Figure 1-10.

[8] This is an example of a *global* variable, meaning its value will be available on another screen within Power Apps. I could have used a *context* variable instead by using this syntax: *UpdateContext({messageVariable: "Your email has been sent."})*. A context variable can only be referenced from the screen where it was created.