ACCESS WEB APPS
BUILDING SOLUTIONS FOR SHAREPOINT AND OFFICE 365

PRESENTED BY: JULIAN KIRKNESS
KIRKNESS ASSOCIATES SOFTWARE
CONTENTS

Chapter 1 – Introduction ........................................................................................................ 3
  Who am I? ......................................................................................................................... 3
  Access 2013 Web Apps ................................................................................................. 4
  What can you do as a beginner? .................................................................................. 4

Chapter 2 – Editing Views, Automating with Macros ....................................................... 11
  Editing the Views ............................................................................................................. 11
  Creating a Settings Table .............................................................................................. 15
  Now let’s use Macros to make use of this setting ..................................................... 15
  Summary ....................................................................................................................... 18

Chapter 3 – Capturing User Information, Sending Reminders / Email Alerting ............. 19
  Create the Users Table ................................................................................................. 19
  Discovering the ‘Current User’ .................................................................................... 20
  Sending an Email to Alert a User about an Activity ................................................... 23

Chapter 4 – Formatting EMAILS, USING Viewer.aspx to provide alternative User Interfaces ........................................................................................................... 27
  Improving our Alert Email ......................................................................................... 27
  Adding the Completed Field ...................................................................................... 31
  Building the Task Manager ....................................................................................... 32
  Setting the View up for use outside the app ............................................................ 35
  The Competed View in a Browser ............................................................................. 37

Chapter 5 – Managing and Deploying Access Web App Solutions ................................ 38
  Saving your App as an app package ......................................................................... 38
    Save as Snapshot ........................................................................................................ 38
    Save as New App ....................................................................................................... 39
    Save for Deployment ................................................................................................. 39
  On Deploy Macros – Macros which only run on deployment .................................... 40
  Deploying your APP .................................................................................................... 43
  Deploying via the App Catalogue .............................................................................. 43
  Preparing an app for sale in the App Store ............................................................... 44
  Pricing Options for AWAs on the App Store ............................................................ 46

Chapter 6 – Making our App Mobile with PowerApps .................................................. 49
  What is PowerApps? .................................................................................................... 49
  Preparing our AWA Database for External Access .................................................... 50
Creating Your First PowerApp.................................................................51
Step 1: Creating a Connection ..............................................................51
Step 2: Your First App – Using Start from your data................................53
Sorting and Filtering.............................................................................56
What to do next....................................................................................58
CHAPTER 1 – INTRODUCTION

WHO AM I?

My Name is Julian Kirkness and I have been an Access user since version 1 (released in 1992) at which time I was looking for suitable technology to build an HR System for sale to clients. I finally released a product (and company) in 1996 - based on Access 95. The business grew over the years and I finally sold it in 2010.

Recently, I found myself in a similar situation - looking around for technology to build another product and business. Of course any new software product needs to be web based (or mobile) these days, as well as being easy to host in the cloud.

To my surprise I found myself considering Access 2013 and the newly introduced Web Apps. In the end, I was the first person to publish Access Web App based solutions for sale in the Sharepoint App Store. I became an Access MVP in July 2014.
ACCESS 2013 WEB APPS...

I am not a Sharepoint expert by any means, but with the platform forming the foundation of many user’s IT strategy, a relatively straightforward means of producing database applications in this environment has long been overdue.

Access 2010 introduced rudimentary features for building web database applications but these were limited to connecting to data in Sharepoint Lists and provided limited functionality. However, with the 2013 release, Microsoft has provided the ability to build web databases connected to SQL Server / SQL Azure and host them in Sharepoint. This is huge because it allows for robust storage of data - and yet users need know nothing of the complexities of SQL Server in order to create applications!

So the main reasons I chose to use Access Web Apps to build my new apps (and business) can be summarised as follows:

- Very fast development
- Robust deployment and security with Sharepoint Online and SQL Azure
- Opportunity to distribute and sell through the Sharepoint App Store
- Simple management of updates and fixes via the app store

At the moment, I believe that Access Web App (AWA) technology is the only platform for building web applications which fulfils all these criteria and I am still delighted with the decision to use the technology.

WHAT CAN YOU DO AS A BEGINNER?

As you can see in the above screenshot, you can create databases either from scratch or you can use one of the templates shipped with the product - the latter is probably a good place to start - if nothing else to let you play with the tools and see some examples of what you can achieve. However, I’m going to walk you through the process of creating a simple app to record your contacts and their related activities (emails, calls etc).

First click on Custom Web App in Access 2013 or 2016 (you can also create a new app from within the Site Contents page in your Sharepoint site).
I am going to create an App called Contacts App for Blog – and you will eventually be presented with the Access designer in the Add Table option:

Tables can be created from a template or from scratch - here's an example of a Contacts table I've created myself:
Saving the table causes Access to automatically create a new form and menu option, as follows:
So within a few minutes you can quickly have a fully functioning (if rather simple) web application which is potentially available to everyone in your organisation!

In a few more minutes, I created another table (Activities) in which I record notes of conversations, emails etc. with my contacts – the key here is that when creating the Activities Table, we simply create a Lookup field to connect it to Contacts:

Here I have shown the table design along with details of the Contact lookup field – notice that I’ve selected “Delete corresponding records...” – this sets up a Cascade Delete in the resulting SQL database so that Activities are deleted when their associated Contacts are deleted.

As soon as the new table is saved, Access will again create the new views and menu options – and because the tables are linked it has also created something called a ‘Related Items Control’ on the Contacts view:
As you can see, we now have a list of activities relating to each contact and the ability to edit them (simply clicking on them) or creating new ones (Add Activities) - using a pop up screen as below:
Everything we have done so far has been automatically set up for us by Access – but we still have a fully functioning App capable of recording dealings with our customers. We are going to leave it there for this first chapter – but if you want to follow along, why not create your own copy of this app for yourself.

In the next chapter, we will be looking at how to customise the screens that Access has provided for us and how to build some automation with macros.

It is quite possible to build fully featured applications with AWAs and over the next few chapters I hope to show you how. To give you an idea of what is possible though here is an example screen from our KasCur PRO CRM app:
CHAPTER 2 – EDITING VIEWS, AUTOMATING WITH MACROS

So far we have built a simple database in Access 2013 / 2016 Web Apps (AWA) to record details of our customers with a related table holding Activities associated with them. In this article, we are going to edit the standard Views (forms) produced by Access and create a couple of Macros to automate things.

EDITING THE VIEWS

Up until now, we let Access create it’s standard views for each table for simplicity. Now it’s time to tidy these up. The Contacts view looked like this:

In order to edit this, you need to open the app in Access 2013 or 2016 (for example by clicking Customise in Access from the tools charm on the top right of the browser window) and then select edit for the Contacts List view:
Here you are able to change the layout using simple drag and drop techniques. We can also change fonts, label text, add headings etc. Here is an example of how you may want to improve this particular view:
This change took about two minutes to do and looks as follows in the browser:

Notice that I have added some section headings - but that they are not very clear. Ideally, I would like to give these a background colour – but unfortunately there isn’t an option for this in the screen designer (the BackColor ‘property’ is only available to be set through a macro). This is the first place where you can use a Macro to achieve some simple UI changes automatically. Each View has two Actions which can trigger Macros – On Load and On Current. In this case I will create an On Load Macro for the View to change the background colour of the headers in question...
Note that we’re using the SetProperty Macro Action for each of 3 labels. This macro will run every time the View is loaded setting the colours accordingly. Here’s what the screen looks like now:
I think you’ll agree that this layout is more pleasing than the default and was achieved in just a few minutes. In our commercial apps on the app store, we allow our customers to choose the colour of these dividers to tie in with their Sharepoint style. This is then stored in the system and applied to each View as it’s loaded. Let’s see how to do this...

CREATING A SETTINGS TABLE

The first step is to create a table to hold this (and any other) system wide setting – we’ll call it Settings – click on New Table and then ‘add a new blank table’ and then add a single field:

![Table with fields](image)

Save this new table as Settings – Access will create a new table and associated views as before. Now save and run the app and enter a (Hex) colour code in the first empty record in the Settings table:

![App interface](image)

We are only ever going to hold one record in this table so it makes sense to make sure that others can’t be added – this would be achieved by removing (in design view) any ability to add a new record – or delete the existing one. To do this you can delete the add and delete Action Bar buttons by selecting them and pressing delete.

NOW LET’S USE MACROS TO MAKE USE OF THIS SETTING

In itself, this won’t change the colour of the headings on the screen – to do this we need something to look up this value in the database. This type of work is handled by Data Macros – these are similar to ordinary Macros but give you the ability to access and manipulate data. Technically, they are converted into SQL Stored Procedures and are processed on the database...
server. Here’s a data Macro (created from Advanced/Data Macro) which will look up this value and return the colour code:

Again this is very simple – but to actually use this value we also need to modify the View’s On Open Macro to apply this value:
Now, using this macro, the colour of the screen headers will be set to whatever colour the user wishes, for example:

**List**   **Datasheet**

<table>
<thead>
<tr>
<th>Filter the list...</th>
<th>![Add New Action]</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FBE9FA</td>
<td>#FBE9FA</td>
</tr>
</tbody>
</table>

Gives us a screen looking like this:
SUMMARY

Although this chapter has only touched upon a relatively simple use of Macros and Data Macros, I think we have shown, in principle, how these are used to access data and use the results to achieve something useful. Hopefully you will have some of your own ideas and will do some experimenting.

In the next chapter, I will show how we can use Macros in more sophisticated fashion to send emails to users to remind them of tasks that need doing. This will involve creating a ‘Users’ table, linking this to Activities and then creating the macros required to send out required reminders.
CHAPTER 3 – CAPTURING USER INFORMATION, SENDING REMINDERS / EMAIL ALERTING

In the previous two chapters, we built a simple database in Access 2013 Web Apps (AWA) to record details of our customers with a related table holding Activities associated with them. In this chapter, I will show how we can use Macros in more sophisticated fashion to send emails to users to alert them of tasks that need attention. This will involve creating a ‘Users’ table, linking this to Activities, and then creating the macros required to send out required emails.

CREATE THE USERS TABLE

This new table will hold the names and login emails (passed from Office 365) of all users. We are assuming in this article that you are using an Office 365 account rather than a Sharepoint Online account because the latter cannot pass an email address to us. If you are on Sharepoint online, then you can use the Sharepoint ‘Display Name’ in the macros to identify the user – feel free to contact me for advice about this.

As before, create a new blank table and add fields as follows:

After saving the table as ‘Users’, the next step is to update the Activities table to include a Lookup to Users:
Then complete the details of the lookup as follows:

![Lookup Wizard](image)

You are now able to link each Activity to a User (as well as a Contact), as follows:

![Activities](image)

**DISCOVERING THE ‘CURRENT USER’**

When a user logs into an Access Web App from Office 365, you are able to create a macro to access their login email address using the UserEmailAdress() macro function. What we are going to do now is create an ‘On Start Macro’ (a special macro which runs when the system loads) which will call a Data Macro. In the macros we are going to do the following:

1. Get the user’s email address in On Start and pass it to the Data Macro.
2. Look for a matching record in the users table and if found set a variable called CurrentUserID – this will be the value of the ID field in their user record.
3. If there is no record for them, create a new User record populating it with their Email address and Sharepoint Display Name (using the function UserDisplayName()). Once the record is saved set the CurrentUserID.

To do this we will have to use a Data Macro (called by the On Start Macro) to lookup a record, create the new record if required and, once it is saved, look it up again to find out what the ID field value is (because the ID field is only populated by Access on saving a record). We need to start with the Data Macro:

The first section of the macro tries to find the user and if found returns a value to the calling macro (On Start) of the user’s ID. Note the user of a Parameter for the macro called UserEmail:

This totally caters for the situation where the User already exists in the Users table and will return UserID to the calling macro (Stop Macro). The next section will create a new record if not:
So, there we have our data macro (if you’re following this then save it as FindorCreateUser).

In order for this to work, we now need to call the data macro from the On Start Macro (which is a special macro which can be edited from the Advanced section on the Ribbon) – here is what is needed:

Now when a user logs onto the app, a User record will be created for them (if none existed) and they will be available to be used in the User field on Activities.
SENDING AN EMAIL TO ALERT A USER ABOUT AN ACTIVITY

In this book we are only going to look at a simple scenario for sending emails when a button is clicked to send details of an Activity to a user. Once again, it is a Data Macro which will do the heavy lifting. Let us provide a feature so that the system will email details of an Activity when someone clicks a button on the Activities View. A scenario here may be that someone creates an activity and assigns it to someone else.

We are going to create a Data Macro which will build the text of the email from the fields from the activity and then sends it:

![Data Macro](image)

```sql
/* First lets build the text for the body of the email - note that you can create HTML Email from Access Web Apps and we have used a little in this example. You will also see that we have built the text using a number of steps with tests to make sure that the fields have a value - if we didn't do this and there were a null field then the text would not be created. */
SetLocalVar
   Name EmailBody
   Expression = "<b>An Activity has been assigned to you from your Contacts App</b>"l</p></p>"

If [Activity] Is Not Null Then
   SetLocalVar
      Name EmailBody
      Expression = [EmailBody]+"Activity: "+[Activity]"</p></p>"
End If

If [Contact] Is Not Null Then
   SetLocalVar
      Name EmailBody
      Expression = [EmailBody]+"Contact: "+[Contact]"</p></p>"
End If

If [Note] Is Not Null Then
   SetLocalVar
      Name EmailBody
      Expression = [EmailBody]+[Note]"</p></p>"
End If
```

The next section of the Data Macro now looks up the email address to send the alert to and then sends the email:
Finally, to make this work, we need to add a button to the Activities screen and call the data macro (SendActivityAlert):

```plaintext
/* Now find the assigned user's email... */
Look Up A Record In Users
  Where Condition  = [Users].[ID]=[UserID]
SetLocalVar
  Name  SendTo
  Expression = [Users].[User login email]
/* And send the email... */
SendEmail
  To  ={SendTo}
  Cc
  Bcc
  Subject  An activity needs attention
  Body   ={EmailAddress}

Note that I have tidied up the View and also user the background colour setting technique shown last time to colour the heading.

The On Click macro for the button is as follows:
So there we have it – this is what the final Activities View (Activities List) looks like:

And here is an example of the email sent from this record:

An Activity has been assigned to you from your Contacts App
Activity: Call Julian
Activity: 1
Call to arrange a meeting to discuss the new HR software
Note that this email is not very attractive to look at yet – I propose to look at how we can enhance this in the next chapter. I will also be covering how we can use alerts to remind users of multiple incomplete Activities and look at ways to user the CurrentUserID we have learnt how to find above.

As an exercise, can you spot the mistake(s) in the above email – and can you spot the cause in the Data Macro?
CHAPTER 4 – FORMATTING EMAILS, USING VIEWER.ASPX TO PROVIDE ALTERNATIVE USER INTERFACES

So far we have built a simple database in Access 2013 Web Apps (AWA) to record details of our customers with a related table holding Activities associated with them (a VERY simple CRM).

In this chapter, I will show how we improve the email we created in the last article. Then we will look at how you can build views for users to whom you don’t want to give full access to the application.

IMPROVING OUR ALERT EMAIL

Some readers may have noticed that the email we sent in the previous chapter wasn’t really complete – for example, the second Activity ‘field’ simply shows a ‘1’ rather that the name of the activity. Also, full marks to those who spotted the deliberate (?) mistake, this field’s caption should have had the caption ‘Contact’:

An Activity has been assigned to you from your Contacts App
Activity: Call Julian
Contact: 1
Call to arrange a meeting to discuss the new HR software

The reason for the contents showing a ‘1’ is that we passed the SendActivityAlert macro the contents of the control [ContactAutoComplete], which contains the ID of the Contact (although it displays the Name):
The second ‘mistake’ with the Contact is that the parameter in the SendActivityAlert macro was created as a ‘Short Text’ value when it should have been ‘Number (No Decimal)’ – so here is what the corrected data macro would look like:
First lets build the text for the body of the email - note that you can create HTML Email from Access Web Apps and we have used a little in this example.

You will also see that we have built the text using a number of steps with tests to make sure that the fields have a value - if we didn't do this and there were a null field then the text would not be created.

```plaintext
/*
SetLocalVar
    Name   EmailBody
    Expression  = "<b>An Activity has been assigned to you from your Contacts App</b></p></p>"

If  [Activity] Is Not Null  Then
    SetLocalVar
        Name   EmailBody
        Expression  = [EmailBody]+"Activity: "+[Activity]+"</p></p>"
End If

If  [Contact] Is Not Null  Then
    Look Up A Record In  Contacts
        Where Condition  = [Contacts].[ID]=[Contact]
        SetLocalVar
            Name   ContactName
            Expression  = [Contacts].[Contact Name]
        SetLocalVar
            Name   EmailBody
            Expression  = [EmailBody]+"Contact: "+[ContactName]+"</p></p>"
End If
*/
And the resulting email:

An Activity has been assigned to you from your Contacts App
Activity: Call Peter about the proposal
Contact: Peter John Smyth
Call about the proposal - go over the features of the solution and get his agreement that these meet his requirements.

So, that’s tidied up that process up nicely. However, the activities part of the app is not that useful – there is no way to indicate whether an activity has been completed for example. In the next section, we are going to add a ‘Completed’ field and then add this to the activities view. Finally, in this chapter, we are going to build a task list app which a user can load directly into their browser (separately from the main app).
ADDING THE COMPLETED FIELD

The first step is simple to add the field to the Activities table:

Note that I have chosen to add this as a Yes/No field and, by default, Access will use a checkbox control to display this information. Here is the modified screen design:

Note that the quickest way to place the new field on the View is simply to drag it from the Field List to the right.

We now have the data structure we need to create a simple task list app which, in this case, I am going to show you how to create ‘externally’ to the main app. In this case this is not really necessary – but in many apps you will find that you need some users able to access all data, administer the app etc., and others with more limited functionality – this is an approach you can use for this...
BUILDING THE TASK MANAGER

First create a new ‘Blank View’ from the Advanced section on the Ribbon and set it’s Record Source to Users:

Now populate the View with Fields, a Heading label and a Related Items Control (RIC) as follows:

Note that the ID field is only required to link the RIC to the parent record and therefore can be Hidden (in the properties of the control). Close the view and save it as ‘TaskListView’.

In the RIC, we are going to create two tabs, one for incomplete tasks (a ToDo list) and one for completed tasks. The best way to do this is to create two queries:
**qryIncompleteActivities:**

To create this query (our first in this app), go to the advanced section of the Ribbon and choose new Query. Now add the Activities Table to the query and, when added double click first on the ‘*’ in the list of field names – this chooses all the fields in the query and they will all be in the result – and then double click on the Complete field. So that this field is NOT included in the result (it’s already there once) remove the tick in the Show tickbox and enter the Criteria No as shown below. Then close and save the query as ‘qryIncompleteActivities’.

![Query Diagram](image)

**qryCompleteActivities:**

This is essentially exactly the same process except that you should use a Criteria Yes in the Complete column:
Now let’s go back to our TaskListView and ‘connect up’ the RIC – here’s the first tab setup process (we’ll call in ToDo List):

Now let’s set its properties:
Note that to select the popup view ‘Activities List’ you will need to have clicked the ‘show all views’ tickbox (not shown above) first. I have purposely left the second column blank to allow more space for the text in the Activity field.

Now create the second tab ‘Completed Tasks’ for yourself – here I suggest you sort the RIC by Activity Date Descending and tick ‘Hide Add Link’.

**SETTING THE VIEW UP FOR USE OUTSIDE THE APP**

As I mentioned at the beginning, we are going to design this feature so that it loads into its own web page rather than being part of the main app. One of the downsides of doing this is that the On Start Macro we created won’t run – this means that we will have to duplicate some of this functionality in the On Load event of our form. Specifically, we will need it to:

- Work out who the user is
- Filter the data so that it only shows their data
- Set the Colour of the Header

We can copy the code from the On Start Macro, and the On Load event from another View for the Colour setting – here is the Macro we need:
So now we have completed the page – so how do we go about loading it independently of the rest of the application...

Here is a typical URL for the contacts app:

https://kirknessassociates-xxxxxxxxxxxxx.sharepoint.com/contacts%20app%20for%20blog/default.aspx#Tile=Contacts&View=Contacts%20List

In order to change this to display our page, it should be changed to:

I have highlighted what should be replaced and by what.

THE COMPETED VIEW IN A BROWSER

<table>
<thead>
<tr>
<th>My Task List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Name</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Contact</th>
<th>Activity Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Julian</td>
<td>Julian Patrick Kirkness</td>
<td>20 March 2016</td>
</tr>
<tr>
<td>Call Peter about the proposal</td>
<td>Peter John Smyth</td>
<td>27 March 2016</td>
</tr>
</tbody>
</table>

Add ToDoList

We have now completed a somewhat simple app with two user interfaces, some basic alerting, etc. In the next chapter we are going to look at how to deploy and manage Access Web App solutions, including the option of releasing them on the App Store.
CHAPTER 5 – MANAGING AND DEPLOYING ACCESS WEB APP SOLUTIONS

So far, this book has focussed on actually building an AWA based solution. In this chapter, we are going to look at the options available to you for deploying these apps either internally or to the wider world including a brief introduction to selling apps on the Sharepoint App Store.

SAVING YOUR APP AS AN APP PACKAGE

An ‘app package’ is a file which contains a deployable (can be loaded into Sharepoint) copy of your app – and can optionally include your data. When saving an app (from File) there are 3 main options:

**Save Database As**

- Save current app as an App Package (*.app).
- **Learn more**

<table>
<thead>
<tr>
<th><strong>Snapshot</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This package is a complete backup of the app including all data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Deployment</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This package may be submitted to sites including the Office App Marketplace and an internal corporate catalog, as a new app or an upgraded version of an existing app.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>New</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This is a package that can be used as a basis for creating a new app.</td>
</tr>
</tbody>
</table>

Let’s look at the three options and then we’ll consider their pros and cons in terms of deploying your solution.

**SAVE AS SNAPSHOT**

This option is basically backing up your app. The app is saved with all data and the same App ID as the original app. This can be re-installed in a different team site if you wish. I would recommend periodically saving a snapshot of your apps during development.
SAVE AS NEW APP

Save as New App is similar to the above in that it saves a regular ‘Unlocked’ copy of your app – but this time saving the data is optional. In this case, the saved app is given a new App ID and so could be re-installed alongside the original app if you wanted to.

SAVE FOR DEPLOYMENT

For my work, this is the key option. Save for Deployment creates a ‘Locked’ version of your app (which means that, once re-installed, users cannot “Customise in Access” or change the app in any way).

Locked apps produced in this way are generally cleared of all data and ID fields are reset. The exception to this is that the developer is able to ‘Lock’ some tables in their design which will then be exported with data – and users will not be able to edit the contents. This means that you can keep control of key look up tables in your design. Tables are locked from their properties which are accessible by right clicking the table:

The really key advantage to apps produced this way is that they can be upgraded easily – and will automatically apply any schema or other changes by simply taking the Upgrade option and applying a new copy of the file (which must have a ‘higher’ version number).
On the surface, this seems the perfect option for managing Development, Test and Deployment. However, there is a significant downside at the moment. This is that Locked apps distributed in this way cannot be connected to by external applications (such as Power BI etc). This is because the facilities to open connections to the database are only available in the Access client and the app cannot be opened in Access (because it is locked).

This means that if external connections are required your options are really limited to working on your live system.

Finally, I would point out that this is the only usable option if you want to protect your design and / or need to be able to upgrade your solution at many installations. For my work, selling apps on the app store, this is the only option that makes sense.

**ON DEPLOY MACROS – MACROS WHICH ONLY RUN ON DEPLOYMENT**

If you make changes to your schema which require data to be moved, then there is an option for that called an On Deploy Macro. This is a special type of Data Macro which only runs when the app is installed for the first time or upgraded (and is most appropriate when using Locked apps produced using Save for Deployment).

On Deploy macros can call other macros and in my published apps I use this technique on a regular basis and my On Deploy process is slowly growing (because you cannot force updates on users, the Macro may therefore have to deal with several versions worth of changes at once). Here is an example:
This is a fairly complex example in my HR application which has been on sale now for about 18 months and has had 30+ updates since launch. Most updates do not require an On Deploy macro but on occasion this is needed. For example, one update included a change where Emergency Contacts for employees were moved from a field on the employee record to a separate table – here is the On Deploy Macro which did the work (and is called from the above):
This demonstrates a useful technique. The ‘Look Up A Record’ action determines whether the new Employee Contacts table contains any records and if so aborts the macro – this ensures that the macro will only run if required.
DEPLOYING YOUR APP

Once you have your .app package you are able to install it in other sites. To do this you simply go to Site Contents, pick Add an App and then select Access App from ‘Apps you can add’:

From here you can browse to your .app package and load it into your site.

Assuming you are using Save for Deployment then you can also upgrade from Site Contents:

Again, browsing to the new (locked) app package will enable you to apply an update.

If your app is deployed via the App Store or your internal App Catalogue (more in a moment) then you will see an indication in Site Contents that an upgrade is available and you simply need to follow instructions to apply the upgrade.

DEPLOYING VIA THE APP CATALOGUE

Each Sharepoint installation will have an App Catalogue and you are able to load your app into this to make it available for others to load (from the ‘From your Organisation’ option):
PREPARING AN APP FOR SALE IN THE APP STORE

Before you can post apps to the App Store, you need a Microsoft Seller Account. I am not going to go into the process of getting this account in this book – I am just going to look at what is required in order to make an .app file ready for submission.

The app file created directly by Save for Deployment would almost certainly be rejected by the app store – but for only one reason which is the lack of an app specific Icon. The app package needs to include a 96x96 Icon in .png format which needs to be the same image as that entered into the submission page when you submit the app for approval.

You also need to make sure that if your app displays a version number on screen to the user that this is the same as the version number you enter in the submission and also the same as the number you entered when you created the app file.

So in order to change the app package (which is actually a re-named .zip file) you need to rename it to YourAppName.zip (where YourAppName is the name of the file you exported) and then expand it – you should end up with a folder like this:
The Icon you create for your app needs to be copied into this folder as ‘accessappicon.png’:

Depending upon the geography in which you want to deploy your app, you may also need to edit the appmanifest.xml file. For my apps, produced on a UK based installation, I add in another ‘Locale’ to the file for the US. In Access Web Apps this only effects date formats but having the two options means that users can choose between UK (and most of the rest of the world’s) date formats and US formats:
<xml version="1.0" encoding="utf-8">
<App
    ProductID="b80549cb-366f-4ea7-9c41-6287699e05b2"
    Type="AccessLocked"
    Version="1.6.2.0"
    SharePointMinVersion="16.0.0.0"
    Name="KasPer Pro"
xmlns="http://schemas.microsoft.com/sharepoint/2012/app/manifest">
    <Properties>
        <StartPage>~appWebUrl/default.aspx</StartPage>
        <Title>KasPer Pro</Title>
        <WebTemplate
            Id="ACCSVC#1" />
        <SupportedLocales>
            <SupportedLocale
                CultureName="en-GB" />
            <SupportedLocale
                CultureName="en-US" />
        </SupportedLocales>
    </Properties>
    <AppPermissionRequests />
    <AppPrerequisites>
        <AppPrerequisite
            Type="Auto Provisioning"
            ID="Database" />
        <AppPrerequisite
            Type="Capability"
            ID="{7CC11180-92E4-49F4-AF68-A55CA440E761}" />
    </AppPrerequisites>
    <AppPrincipal>
        <Internal
            RedirectUrl="access:AuthRequestComplete" />
    </AppPrincipal>
</App>

Generally the rest of this file should be left as-is but is nonetheless interesting as it includes version numbers, the type of app (AccessLocked) etc.

Once you have your Icon and (if required) edited the app manifest then you should create a new .zip file from the contents of this folder and rename it to YouAppName.app – it is then ready to be submitted to the store.

PRICING OPTIONS FOR AWAAS ON THE APP STORE

The screenshot below shows some of the options available to you for pricing apps on the Sharepoint Store:
In addition to the options shown in this example, you can charge on a monthly per-user subscription basis (this is not shown in this shot because they don’t allow you to change a one-time purchase into a subscription).

So, you are able to offer per user pricing, subscription pricing and trial periods – right? Yes and No – with Access Web Apps you are able to configure these options for your store entry but beware – your app WILL NOT ENFORCE ANY LICENCE OPTIONS.

This means that if you offer your app for a trial period that it will continue to work forever – you have given your app away. Similarly, subscription pricing is not enforced nor is the count of the number of users.

For this reason, for my main HR app I have built licencing into the app itself and provide it free on the app store with a 14 day trial period (again this in managed by the app). This approach is quite straightforward to implement (in a Locked app) but does remove one of the great advantages of selling through a store which is that of customer billing and VAT / Sales Tax handling.

I would therefore recommend that careful consideration is given to this before deciding to develop an App for commercial sale on the store (unless Microsoft resolve these issues).
Hopefully you have found this chapter interesting and useful. In the next (and final) chapter in this book I am going to look at how we can make our app mobile with Microsoft’s new PowerApps platform, an ideal companion product to AWA.
CHAPTER 6 – MAKING OUR APP MOBILE WITH POWERAPPS

So far we have focussed on actually building an AWA based solution. In this chapter, now that we have our basic application, we are going to look at the options available to you for making your Apps mobile with the new Microsoft PowerApps.

PowerApps is currently available as a Public Beta – by invitation. Request your invitation at:

https://powerapps.microsoft.com

WHAT IS POWERAPPS?

PowerApps is a tool designed for Power Users and developers to build mobile apps from your corporate data fast. The tool uses a model similar to Excel to build expressions and formulae and so should feel familiar to many:

Apps created in PowerApps run inside the PowerApps clients on Windows, iOS and Android.
PREPARING OUR AWA DATABASE FOR EXTERNAL ACCESS

As we discussed in the previous chapter, AWA applications can be either Locked (distributed using Save for Deployment) or unlocked. It is only possible (at the moment) to open the database up to external access in Unlocked apps and this is handled in the Access client. This is how it’s done:

Click on Manage Connections:
From here you can click on Enable Read-Write Connection and either From My Location or From Any Location, as required. Once this is done, you can click View Read-Write Connection Information.

I have redacted some of this information – but this can be then used to create a Connection String which can then be entered into PowerApps to connect it to your database:

Server=tcp:XXXXXXXXyw6.database.windows.net;Database=db_XXXXXXXX_51e9_4bc9_8575_8e36ebaade20;User ID=db_XXXXXXXX_51e9_4bc9_8575_8e36ebaade20_ExternalWriter@yqaqwyeyw6.database.windows.net;Password=-XXXXXX^/dD-Rm;Trusted_Connection=False;Encrypt=True;

At this point you should use Notepad to take the connection information for your copy of the app and create a Connection String.

CREATING YOUR FIRST POWERAPP

STEP 1: CREATING A CONNECTION

The first thing you need to do to connect PowerApps to your AWA database is to create a Connection:
Choosing SQL Azure from the list of available connections, enter your connection string and then click Connect:
Unfortunately, at the moment, you don’t seem to be able to name your connection so you may end up with several called SQL Azure which could become confusing!

STEP 2: YOUR FIRST APP – USING START FROM YOUR DATA

PowerApps gives you the choice to build an app from scratch, build it automatically from a Data Source or build from a template. As we are talking about making mobile apps to extend the scope of your AWA application, we’re going to choose Start from your data:
Having clicked, you can then choose which data you want to connect to:

The next step, having chosen SQL Azure, is to choose the Dataset – click Default and then select a Table:
In this example we’re going to select [Access].[Contacts] which is the name AWA gives to our Contacts table from within our Web App. *Note that at this point it doesn’t seem possible to connect to queries in the database – just Tables.*

PowerApps now goes away and creates a default 3 page app based upon this single table with Browse, Detail and Edit screens:

So, we now have a mobile app connected to the Contacts table from our AWA – you can run it right here in the PowerApps client by clicking the play Icon at the top of the screen.
This app can be customised and improved from here in the PowerApps designer – there are many improvements you may want to make – for example changing which fields are shown in the Browse screen, changing the sorting of the records, etc...

**SORTING AND FILTERING**

If we select the ‘Gallery’ in the browse page (the control which display the grouped fields) we can see it has an Items property which includes an expression:

Here’s the expression:

```
Sort(If(IsBlank(TextSearchBox1.Text), '[Access].[Contacts]', Filter('[Access].[Contacts]', TextSearchBox1.Text in Text(Address))), Address, If(SortDescending1, Descending, Ascending))
```

The syntax of the Sort expression is:

```
Sort( Table, Formula [ , SortOrder ] )
```

So, looking at the expression from our app, we can see that the screen will present data from the Contacts table if there is no entry in the Search box and Filtered records if there is.
However, the search and sort are both referring to the Address field and I would prefer to use Contact Name for this – here’s what we need to change:

```plaintext
Sort(If(IsBlank(TextSearchBox1.Text), '[Access].[Contacts]', Filter('[Access].[Contacts]', TextSearchBox1.Text in Text('Contact Name'))), 'Contact Name', If(SortDescending1, Descending, Ascending))
```

If you make this change you will see, even in design view, that the order will change straight away.

Another interesting part of this expression is that the Sort is either Ascending or Descending depending upon the value of a ‘Context Variable’ called SortDescending1. This variable is set by clicking the ‘⇅’ button on the top bar in the app. Selecting this button reveals how this variable is updated:

We have the following expression in the OnSelect property of the image:

```plaintext
UpdateContext({SortDescending1: !SortDescending1})
```

Again, looking at the syntax:

```plaintext
UpdateContext({ ContextVariable1: Value1 [, ContextVariable2: Value2 [, ..., ... ] ] })
```
We can see that variable SortDescending1 is reassigned a value !SortDescending1. The “!” symbol represents a logical NOT and therefore this expression is toggling the value of the variable from 0 to 1 and back – and this controls the sort with no further code.

WHAT TO DO NEXT

If you would like to give PowerApps a try, then you need to apply for an invitation to try it out as mentioned at the beginning of this article. In addition to this brief introduction, you will also find the help for PowerApps pretty good – it is available directly by clicking the ? on the PowerApps designer and here:

https://powerapps.microsoft.com/en-gb/tutorials/

I hope you have found this brief introduction a useful extension to this book and hope you enjoy PowerApps.